Fachhochschule Köln
Cologne University of Applied Sciences

Mehrprozessbetrieb für mobile kontextsensitive Anwendungen

Konzeption und prototypische Implementierung auf Basis eines bestehenden Rahmenwerks

Multitasking for mobile context-sensitive applications

Design and prototype implementation based on an existing framework

BACHELORARBEIT

ausgearbeitet von

Benjamin Krumnow

zur Erlangung des akademischen Grades

BACHELOR OF SCIENCE

vorgelegt an der

FACHHOCHSCHULE KÖLN
CAMPUS GUMMERSBACH
FAKULTÄT FÜR INFORMATIK UND
INGENIEURWISSENSCHAFTEN

im Studiengang

MEDIENINFORMATIK

Erster Prüfer: Prof. Dr. Kristian Fischer
Fachhochschule Köln

Zweiter Prüfer: Prof. Dr. Erich Ehses
Fachhochschule Köln

Gummersbach, im August 2010

Adressen: Benjamin Krumnow
Berliner Strasse 80-82
51063 Köln
Benjamin.Krumnow@googlemail.com

Prof. Dr. Kristian Fischer
Fachhochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
fischer@fh-koeln.de

Prof. Dr. Erich Ehses
Fachhochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
ehses@fh-koeln.de

Kurzfassung

Mobile Systeme haben sich fest in den Alltag des Menschen in der westlichen Welt integriert. Sie helfen bei der Erledigung von Aufgaben der Benutzer und versuchen, deren Bedürfnisse bestmöglich zu erfüllen. Um dies zu erreichen, wurden verschiedene Lösungen entwickelt, dazu gehören kontextsensitive Anwendungen, die sich dadurch auszeichnen, dass sie ihr Verhalten nach den Belangen der Benutzer, bestimmt durch die jeweilige Situation, anpassen können. Zur Entwicklung solcher Anwendungen gibt es Rahmenwerke, um Problemen, bedingt durch die Komplexität dieses Bereiches, vorzubeugen und die Entwicklung zu erleichtern. Dabei spielt auch die Fähigkeit dieser Rahmenwerke, mehrere Anwendungen gleichzeitig zu bedienen, gerade bei steigender Popularität von kontextsensitiven Applikationen, eine entscheidende Rolle.
Diese Arbeit beschäftigt sich mit der Konzeption und Umsetzung des Multiprozessbetriebes für kontextsensitive Anwendungen auf Basis eines bestehenden Rahmenwerkes. Innerhalb der Arbeit werden Anforderungen für den Multiprozessbetrieb ermittelt, auf deren Basis ein Konzept zur Erweiterung des Rahmenwerkes erstellt wird. Anhand des Konzeptes wird eine Implementierung des Rahmenwerkes auf der Android-Plattform durchgeführt, welche danach mittels der Durchführung eines Testszenarios mit mehreren Anwendungen evaluiert wird.

Abstract

Mobile systems have firmly been integrated in the everyday life of people in the western world. They help complete tasks of the users and try to fulfill their needs the best way possible. To accomplish this, different solutions have been developed, including context-sensitive applications. Context-sensitive applications are characterized by their ability to adapt their performance to the needs of the user in different situations. Frameworks are used to prevent or help with problems that come with the development of context-sensitive software. With increasing popularity of context-sensitive applications, it is important that these frameworks are able to operate with multiple applications at the same time.
This work deals with the conception and the implementation of the multi-process operation for context-sensitive applications, based on an existing framework. Within the work, requirements of the multi-process operation are determined and, based on these requirements, a concept to expand the framework is created. With help of this concept the framework is implemented on the android-platform. A test scenario with multiple applications is used to evaluate this implementation afterwards.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

In diesem Kapitel werden die grundsätzlichen Ideen, das Vorgehen und die Zielsetzung für diese Arbeit vermittelt. Des Weiteren wird eine Motivation zur Erstellung dieser Arbeit formuliert. Abschließend folgt eine Übersicht und kurze Beschreibung der einzelnen Kapitel.

1.1 Motivation

Die soziale Verankerungen des Personal Computers hat sich in den letzten Jahrzehnten drastisch verändert. Hierbei haben mobile Systeme eine große Bedeutung eingenommen, da sie die Menschen, besonders in der westlichen Welt, tagtäglich begleiten[1]. Während dieses Entwicklungsprozesses entstanden auf den mobilen Geräten etliche Anwendungen, die den Menschen bei seinen täglichen Aufgaben unterstützen sollen. Um dies zu erreichen, bieten neuartige Anwendungen die Möglichkeit das eigene Verhalten den Gegebenheiten der jeweiligen Situation des Nutzers anzupassen. Anwendungen, die über diese Fähigkeit verfügen, werden oft als *kontextsensitive* Anwendungen betitelt. Allerdings steigt der Aufwand für die Entwickler solcher Anwendungen, da diese sich neben der Entwicklung der eigentlichen Software, zusätzlich mit der Erfassung und Modellierung der jeweiligen Situation auseinandersetzen müssen. Um diesen Prozess zu erleichtern wurden Rahmenwerke entwickelt, die den Entwicklern diese Aufgaben abnehmen oder für sie vereinfachen. Bei voranschreitender Entwicklung und Verbreitung dieser Rahmenwerke ist mit einem starken Zuwachs von kontextsensitiven Anwendungen zu rechnen. Durch die steigende Anzahl könnte in Zukunft eine Vielzahl an kontextsensitiven Anwendungen auf einem Gerät verfügbar sein, welche teilweise auch gleichzeitig Informationen von dem jeweils verwendeten Rahmenwerk benötigen. Dies führt zu neuen Problemen, da beispielsweise nur begrenzte Kapazitäten auf einem mobilen Gerät vorhanden sind, aber es entstehen auch neue Möglichkeiten, wenn kontextsensitive Anwendungen in die Lage versetzt werden eine gemeinsame, erweiterte Informationsbasis zu schaffen und zu nutzen.

[1] vgl. [Bit09]

1.2 Zielsetzung

Diese Bachelorarbeit hat das Ziel die Fragestellung zu klären, wie sich das Rahmenwerk für mobile, kontextsensitive Applikationen aus der Arbeit von [Mül10] für paralell laufende Anwendungen, auf mobilen Geräten mit mulitprozessfähigen Betriebssystemen, umsetzen lässt.
Um diese Fragestellung zu beantworten, soll ein Konzept erstellt werden, das als Grundlage für Implementierungen des Rahmenwerkes zur Unterstützung des Multiprozessbetriebes von kontextsensitiven Anwendungen verwendet werden kann.
Zur Erstellung des Konzeptes ist eine Auseinandersetzung mit den Aspekten und den Problemen der vorliegenden Domäne erforderlich. Hierdurch sollen Anforderungen für den Multiprozessbetrieb des Rahmenwerkes ermittelt werden, auf deren Grundlage sich die konzeptionelle Modellierung des Rahmenwerkes stützen soll.
Des Weiteren soll eine eigene Implementierung angefertigt werden, um zu überprüfen, ob eine erfolgreiche Umsetzung des Rahmenwerkes möglich ist und die Zielstellung erreicht wurde. Für diese Überprüfung ist es notwendig, dass mindestens zwei Anwendungen zum gleichen Zeitpunkt das Rahmenwerk verwenden und Daten von diesem beziehen können.

1.3 Eingrenzung

Da in dieser Arbeit ein bereits bestehendes Rahmenwerk für kontextsensitive Anwendungen verwendet wird, kann auf den bereits vorhandenen Erkenntnissen aufgebaut werden. Dies bedeutet, dass grundsätzliche Überlegungen, wie beispielsweise die Modellierung des Kontextes, nicht erneut behandelt werden, insofern es nicht unbedingt erforderlich ist. Die Konzeption beschränkt sich aufgrund der Thematik auf das mobile Gerät. Es werden also keine konzeptionellen Überlegungen bezüglich des Kontextservers[2] gemacht. Des Weiteren werden keine Aspekte der Sicherheit und des Datenschutzes behandelt, da diese nicht zum Kern der Arbeit gehören.

1.4 Vorgehen

Um das Ziel dieser Arbeit zu erreichen, werden im ersten Schritt die Grundlagen geklärt, die für die Thematik dieser Arbeit relevant sind. Mittels dieser Grundlagen werden über die Anwendung geeigneter Methoden, sowohl Probleme, als auch schon bestehende Lösungen der Domäne ermittelt. Die Erkenntnisse aus diesem Prozess werden für die Ableitung von Anforderungen verwendet, welche die Basis für eine Konzeption stellen. Der

[2] siehe Kapitel 3

nachfolgende Schritt ist die konzeptionelle Betrachtung des Rahmenwerkes für die Umsetzung des Multiprozessbetriebes. Hierbei sollen mögliche Änderungen vorgenommen werden, wenn dies erforderlich ist. Um das resultierende Konzept zu überprüfen, wird eine Implementierung auf einem Multiprozesssystem durchgeführt und abschließend mittels mehrerer parallel laufender Anwendungen getestet. Die daraus entstehenden Erkenntnisse werden im letzten Schritt zusammengefasst.

1.5 Aufbau der Arbeit

In diesem Kapitel wurden das Ziel, Vorgehen und die Grundidee für diese Arbeit formuliert. In dem nachfolgenden Kapitel 2 werden Grundlagen behandelt, auf denen die weiteren Kapitel aufbauen. Dabei werden für die Thematik relevante Begriffe besprochen und eingegrenzt. Danach schließt der Grundlagenteil, durch die Behandlung des Rahmenwerkes in Kapitel 3 ab. Hierbei wird zum einen das Rahmenwerk, welches in dieser Arbeit verwendet wurde, vorgestellt und zum anderen gezeigt, wo diese Arbeit innerhalb dieses Rahmwerkes ansetzt.
Mittels Kapitel 4 erfolgt dann die eigentliche Behandlung der Fragestellung. Es werden Szenarien vorgestellt, die zur Ermittlung von Aspekten des Multiprozessbetriebes mit kontextsensitiven Anwendungen dienen. Anhand von Analysen der Szenarien werden Probleme und positive Aspekte festgehalten. Diese werden dann in Kapitel 5 verwendet, um die Anforderungen an das zu entwerfende System abzuleiten und durch eine Priorisierung auszuwählen. Auf dieser Basis wird in Kapitel 6 das bestehende Konzept und weitere verschiedene Ansätze auf Umsetzung des Multiprozessbetriebs geprüft. Des Weiteren werden Aspekte für die Implementierung durch eine Auseinandersetzung mit Anforderungen auf einer technischen Ebene ermittelt. Das daraus entstandene Konzept und die gewonnenen Erkenntnisse werden in Kapitel 7 verwendet, um Komponenten für eine Implementierung auszuwählen. Für die Auswahl der Komponenten wird im Vorfeld eine Plattform mit der Fähigkeit des Multiprozessbetriebes bestimmt. Danach erfolgen eine Dokumentation der prototypischen Implementierung, sowie eine Evaluation des Prototyps, anhand der Durchführung eines Testszenarios. In Kapitel 8 werden die Arbeitsprozesse und Ergebnisse dieser Bachelorarbeit zusammengefasst. Die Arbeit endet mit einem Ausblick auf weitere mögliche Themenbereiche.

2 Grundlagen

Um die in dieser Arbeit behandelte Thematik erfassen zu können, ist es notwendig zu verstehen, was kontextsensitive Anwendungen sind und welche Bedeutung ein Multiprozessbetrieb für mobile Anwendungen hat. Über die Klärung der Begrifflichkeiten dieser zwei Bereiche, soll ein besseres Verständnis und eine Annährung an das Thema erreicht werden.
Zur Festlegung der Bedeutung von kontextsensitiven Anwendungen, beginnt dieses Kapitel mit einer Untersuchung des Begriffes *Kontext*, welcher dann mittels der Aussagen von Autoren für diese Arbeit definiert wird. Mit dem gleichen Vorgehen wird danach die Bedeutung von kontextsenstiven Anwendungen bestimmt. Im Anschluss daran werden verschiedene Kategorisierungen und unterschiedliche Ansätze verschiedener Autoren vorgestellt. Abschließend wird der Mulitprozessbetrieb für mobile Anwendungen betrachtet und das Kapitel zusammengefasst.

2.1 Kontext

Um sich der Bedeutung von kontextsensitiven Anwendungen zu nähern, soll hier als erstes der Begriff Kontext behandelt werden.
Beim heranziehen von Literatur für Begriffsdefinitionen stößt man auf die Bedeutung *"enge Verknüpfung"* oder *"Zusammenhang"*[1] für den Kontext. Zur Klärung des Begriffes kontextsensitiv, ist diese Definition allerdings zu ungenau. Mittlerweile haben einige Autoren diesen Begriff behandelt, wobei unterschiedliche Aussagen und Definitionen entstanden sind, welche hier verwendet werden sollen, um diesen Begriff für die vorliegende Arbeit festzulegen.

Erste Ansätze diesem Begriff näher zu kommen, sind Aufzählungen der relevanten Aspekte. Schilit et al. definieren den Kontext wie folgt:

"Three important aspects of context are: where you are, who you are with, and what resources are nearby" [SBN94a]

[1] Duden - Deutsches Universalwörterbuch, [Dud07]

In dieser Definition wird der Begriff Kontext also durch unterschiedlichen Artefakte bestimmt, die sich dabei auf eine Person beziehen. Im Hinblick auf eine mobile Anwendung ist dies also ein benutzerzentrierter Ansatz, da Aspekte bei denen der Benutzer im Mittelpunkt steht zur Bestimmung des Kontextes verwendet werden. Ähnlich hierzu bestimmen Ryan et al. [RN97], den Kontext durch die Umgebungstemperatur, Identität, Zeit und Ort des Benutzers. Somit liegt der Unterschied dieser Ansätze in der Berücksichtigung verschiedener Artefakte zur Bestimmung des Kontextes.
Bei der Definition von Brown[Bro96], besteht der Kontext aus einer Kombination von Elementen in der Umgebung, über die sich der Computer des Benutzers bewusst ist. Somit nimmt die Definition von Brown, im Vergleich zu den zwei vorher genannten Ansätzen, eine eher systemzentrierte Sicht ein. In der Arbeit von Franklin und Flachsbart[DF98] wird der Kontext als die Situation des Benutzers verstanden, wobei wieder eine benutzerzentrierte Sicht eingenommen wird.
Im Vergleich zu den hier genannten Definition verfolgen Dey und Abowd[DAK00] einen allgemeineren Ansatz:

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [DAK00]

Dieser Ansatz verzichtet im Gegensatz zu Ryan et al. und Schilit et al. auf die Aufzählung einzelner Artefakte zur Bestimmung des Kontextes, da diese nach Angaben der Autoren nie vollständig sein können. Somit wird keine Unterscheidung zwischen system- und benutzerzentrierten Ansätzen gemacht, da alle relevanten Informationen zur Kontextbestimmung mit einbezogen werden.

Basierend auf den hier aufgeführten Aussagen, ist der Kontext eine Information zur Bestimmung der Situation des Benutzers, die sich aus mehreren Teilinformationen zusammensetzt. Dabei ist jede Teilinformation[2] wichtig für das Bestimmen der Situation.
Für diese Arbeit, im Hinblick auf eine mobile Anwendung, ist der Kontext eine übergeordnete Information, abgleitet aus einer beliebigen Menge von relevanten Teilinformationen, die eine Situation des Benutzers für die mobile Anwendung charakterisiert.

[2] auch Kontextinformation genannt

2.2 Kontextbewusstsein

Nachdem nun die Bedeutung des Begriffes Kontext für den Rahmen dieser Arbeit geklärt wurde, steht die Klärung des Begriffes sensitiv aus, um bestimmen zu können, was kontextsensitive Anwendungen sind.
Gleichbedeutend wird für den Begriff kontextsensitiv auch *kontextbewusst*[3] verwendet. Die Begriffe bewusst und sensitiv werden also gleichbedeutend mit dem Kontextbegriff verwendet. Dabei steht der Begriff sensitiv für empfindlich[Dud10] und bewusst für geistig wach oder auch willentlich[Dud07]. Demnach besitzen kontextbewusste oder kontextsensitive Anwendungen die Fähigkeit, für Kontexte empfänglich zu sein bzw. diese wahrzunehmen. Um dies nun noch etwas genauer spezifizieren zu können, sollen auch hier die Aussagen weiterer Autoren mit einbezogen werden.

Schilit und Theimer [SBN94b] beschreiben Kontextbewusstsein als die Fähigkeit von mobilen Anwendungen, Veränderungen der Umgebung zu erkennen und darauf zu reagieren. Im Vergleich zu Schilit und Theimer beschreiben Hull et al. [HR97] Kontextbewusstsein als die Fähigkeit von Geräten, Aspekte der Umgebung zu erkennen, zu interpretieren und darauf zu reagieren. Die Definition von Pascoe et al.[Pas98] beinhaltet nicht nur die Beeinflussung des Zustandes der Umgebung, sondern auch die des Systems bzw. des Gerätes an sich.

"Context-awareness is the ability of a program or device to sense various states of its environment and itself" [Pas98]

Der Ansatz von Salber et al. [SD98] legt den Schwerpunkt auf die Flexibilität der möglichen, zu erkennenden Formen von Kontextinformationen in Echtzeit. Dazu gehören beispielweise örtliche, intentionale, emotionale und informelle Informationen. Ähnlich, wie schon bei den Definitionen des Kontextes, werden mehrere Aspekte aus- bzw. eingeschlossen, um diesem Begriff näher zu kommen. Allerdings besteht dabei die Gefahr Anwendungen auszuschließen, die eigentlich zu den kontextsensitiven Anwendungen gehören. Die Definition von Dey und Abowd [DAK00] versucht alle kontextsensitiven Anwendungen zu beinhalten und ist deshalb etwas allgemeiner gehalten als die vorherigen Definitionen.

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task." [DAK00]

[3] oder im englischen Sprachgebrauch *context-awareness*

Eine ähnliche Definition stammt von Kurt Rothermel [Rot], welche die hier erwähnten Aussagen umfasst:

"Eine Anwendung ist kontextbezogen, wenn ihr Verhalten durch Kontextinformationen beeinflusst wird." [Rot]

Die hier getroffenen Aussagen verwenden oft unterschiedliche Aspekte, um eine kontextsensitive Anwendung als solche zu bestimmen. Da allerdings so die Möglichkeit besteht, eine Reihe von anderen Anwendungen zu vernachlässigen, sollen für diese Arbeit die Ansätze von [Rot] und [DAK00] verwendet werden. Demnach werden also Anwendungen als kontextsensitiv oder -bewusst bezeichnet, sobald sie Kontexte verwenden, um ihr Verhalten auf eine Situation anzupassen, in der sich der Nutzer befindet.

2.3 Kategorien von kontextsensitiven Anwendungen

Da nun verständlich gemacht wurde, was kontextsensitive Anwendungen sind, soll geklärt werden, in welchen Ausprägungen sie vorkommen. Dies ist für ein besseres Verständnis dieser Arbeit wichtig, da hierdurch deutlich wird, welche verschiedenen Arten von Anwendungen von einem Rahmenwerk berücksichtigt werden müssen. Um diese Ausprägungen erkennen zu können, haben unterschiedliche Autoren Kategorisierungen von kontextsensitiven Anwendungen vorgenommen. Verschiedene Kategorisierungen dieser Autoren sollen hier vorgestellt werden, von denen eine zur Unterscheidung von kontextsensitiven Anwendungen in dieser Arbeit verwendet werden soll.

2.3.1 Arten von Kategorien

Kategorien von kontextsensitiven Anwendungen helfen bei der Klassifizierung dieser Anwendungen. Ein Ansatz zur Klassifizierung sind dienst- bzw. aufgabenbezogene Kategorien. Hierbei werden Anwendungen danach unterschieden, welche Aufgaben sie erledigen bzw. welche Dienste sie anbieten. Solch eine Kategorie stammt von Bauer et al.[BH05]. Dabei werden nur ortsbezogene Dienste behandelt, da nach Ansicht der Autoren, die Ortsinformation die wichtigste Kontextinformation ist. Die Tabelle 2.1 zeigt eine Übersicht der einzelnen Kategorien nach Bauer et al.
Eine andere Variante ist die Klassifizierung der Anwendungen nach den Leistungsmerkmalen bzw. Fähigkeiten, die sie besitzen. Anders als ein konkreter Dienst oder eine Aufgabe, beschreibt ein Leistungsmerkmal, welche Funktion eine Anwendung anhand von Kontextinformation ermöglicht. Dies erlaubt eine umfangreichere Klassifizierung von kontextsen-

sitiven Anwendungen, als das vorher genannte Modell, da keine konkreten Aufgaben bzw. Dienste formuliert werden müssen. Dabei gibt es unterschiedliche Ausprägungen wie beispielsweise von Schilit et al. [SBN94a], die eine Unterscheidung der Anwendungen durch die Art, wie Informationen präsentiert werden oder welche Form von Aktionen ausgeführt werden können, vornehmen. Ein anderer Ansatz hierzu stammt von Pascoe et al., der sich nur auf die Kernfähigkeiten der Anwendungen konzentriert[4] und somit beispielsweise keine Unterscheidung bei der Informationspräsentation benötigt.

Die beiden hier vorgestellten Arten von Kategorisierungen verfolgen das gleiche Ziel, allerdings bergen die Kategorisierungen nach den Fähigkeiten ein größeres Potenzial für diese Arbeit, da sie es durch ihre Unterteilung ermöglichen mehr Anwendungen zu erfassen, während dienstbezogene Kategorisierungen eher nur einen spezifischen Bereich behandeln. Da das in dieser Arbeit verwendete Rahmenwerk[5] einen generischen Charakter anstrebt, ist also die Verwendung einer weiter gefassten Kategorisierung vorteilhaft. Aus diesem Grund soll nun eine fähigkeitsbezogene Kategorisierung vorgestellt werden, um auf diese in späteren Kapiteln zurückgreifen zu können.

2.3.2 Kategorisierung nach Dey und Abowd

Für diese Arbeit wird die Kategorisierung von Dey und Abowd verwendet, da sie zum einen die Ansätze der Autoren Pascoe et. al und Schilit et. al. vereint. Des Weiteren bietet diese Kategorisierung den Nutzen, eine Vielzahl von kontextsensitiven Anwendungen zu klassifizieren. Dey und Abowd bestimmen dabei drei Fähigkeiten, die Anwendungen besitzen können, damit sie zu den kontextsensitiven Anwendungen zählen.
Presentation of information and services to a user – Beschreibt die Fähigkeit Informationen anhand des Kontextes angemessen zu präsentieren. Hierbei ist jegliche Art von Information mit eingeschlossen, auch das Anbieten von Diensten.

Automatic execution of a service – Beschreibt die Fähigkeit von Anwendungen, unter Berücksichtigung des bestehenden Kontextes, automatisch Dienste auszuführen.

Tagging of context to information for later retrieval – Die letzte Fähigkeit bezieht sich auf die Anreicherung der Umgebung mit digitalen Informationen.

[4] vgl.[DAK00]
[5] siehe Kapitel 3

CATEGORY	SUB-CATEGORY	MAIN VALUE
Tracking services	People tracking	Tracking private people or personal
	Object tracking	Tracking objects (products, vehicles, material etc.)
Navigation services	Regular routing services	Localizing navigating towards fixed objects
	Specialized routing services	Localizing and navigating towards specialized product and services providers
	Indoor routing services	Localizing and navigating indoors
Information services	Regular information services	Delivering local information
	Interactive information services	Delivering local information including direct reply mechanisms
Communication services	Private communication services	Easing communication of distributed friends, family members and unknown others with same interests
	Business communication services	Easing communication of distributed employees
Entertainment services		Providing added entertainment value by adapting to location
Transaction services	Location based advertising	Location based initiation of economic
	Location based billing transactions	Location based execution of economic transactions

Tabelle 2.1: Kategorien nach Lokalisation basierenden Diensten von Bauer et al. [BH05]

2.4 Begriffe des Multiprozessbetriebes

Abschließend soll in diesem Abschnitt erläutert werden, was ein Multiprozessbetrieb für mobile Systeme bedeutet und was ein Multiprozesssystem ausmacht. Hierfür sollen zuerst einige Anmerkungen zum Multiprozessbetrieb im Allgemeinen gemacht werden.

Multiprozessbetrieb bezeichnet die nebenläufige Abarbeitung von Tasks, innerhalb einer oder verschiedener Anwendungen[BJ09]. Parallel dazu werden auch die Begriffe *Multitasking* [BJ09], *Mehrprozessbetrieb* oder *Multiprozessumgebung* verwendet. Dies soll nicht

verwechselt werden, mit dem Begriff *Multiusersystem*, welcher das Arbeiten mehrerer Benutzer auf einem System bezeichnet.

Diese Aussagen können nun auf mobile Systeme angewendet werden. Das iPhone OS wurde bis einschließlich Version 3.0 als Singletaskingsystem entworfen. Dies bedeutet das System kann nur eine Anwendung zu einem Zeitpunkt ausführen. Allerdings können Grundfunktionen, wie Telefonie, SMS E-Mail [Mül10] oder auch das Abspielen von Musik [Woo10] parallel zu einer laufenden Anwendung betrieben werden. Bezüglich der hier verwendeten Definition, bietet das iPhone OS also eine Multiprozessumgebung, gilt deshalb aber nicht automatisch als Multiprozesssystem. Deshalb soll für diese Arbeit gelten, dass mobile Systeme als Multiprozesssysteme bezeichnet werden können, wenn sie einen Multiprozessbetrieb für jede Anwendung auf dem mobilen Gerät ermöglichen.

2.5 Zusammenfassung

In diesem Kapitel wurden grundlegende Begriffe für diese Arbeit erklärt und festgelegt. Des Weiteren wurde eine Kategorisierung vorgestellt, mit der nun kontextsensitive Anwendungen klassifiziert werden können. Um die Grundlagen für diese Arbeit zu vervollständigen, soll im nächsten Kapitel das für diese Arbeit verwendete Rahmenwerk vorgestellt werden.

3 Einführung in das Rahmenwerk

Dieses Kapitel dient zur Einführung in das Rahmenwerk, welches für die Durchführung dieser Bachelorarbeit verwendet wird. Ziel dieses Kapitels ist es, die für diese Arbeit relevanten Aspekte hervorzuheben. Somit wird eine Grundlage geschaffen, auf die in den weiteren Kapiteln aufgebaut werden kann, wenn es um konzeptionelle oder andere Überlegungen geht, bei denen das Rahmenwerk betrachtet werden muss. Das Kapitel beginnt mit einigen allgemeinen Bemerkungen zum Rahmenwerk und behandelt dann den konzeptionellen Aufbau. Abschließend wird ein Ausblick darauf gegeben, wo diese Arbeit innerhalb des Rahmenwerks versuchen wird anzusetzen.

3.1 Allgemein

Bei der Entwicklung von mobilen, kontextsensitiven Anwendungen stehen Softwareentwickler einer erschwerten Aufgabe gegenüber, da sie nicht nur mit der Entwicklung der Anwendung an sich, sondern auch mit den Problemen der Umsetzung von kontextsensitiven Funktionen der Anwendung beschäftigen müssen. Dazu gehören unter anderem das Abrufen von Sensoren zum Ermitteln von Kontextinformationen auf dem jeweiligen Gerät, die Entwicklung eines Modells, durch das Kontexte abgebildet werden können und die anschließende Verwaltung der Kontexte. Das in dieser Arbeit verwendete Rahmenwerk, entstanden in der Masterarbeit von Felix Müller [Mül10], schafft eine Grundlage, mit der die Entwicklung solcher Anwendungen vereinfacht werden soll. Dabei wurde das Ziel verfolgt, ein generisches[1], anwedungsunabhängiges Rahmenwerk zu erstellen, welches Zugriff und Verwaltung von Kontextinformationen für Anwendungen ermöglicht[Mül10]. Des Weiteren wurden bei der Konzeption mögliche Erweiterungen in Betracht gezogen, weshalb ein modularer Aufbau gewählt wurde. Um zu zeigen, wie diese Aspekte in dem Rahmenwerk umgesetzt werden und wie es funktioniert, wird im nächsten Abschnitt der konzeptionelle Aufbau des Rahmenwerkes behandelt.

[1] Generisch bezieht sich hierbei auf die Modellierung des Kontextes und Technologien, die für die Umsetzung verwendet werden können.

3.2 Konzeptioneller Aufbau des Rahmenwerkes

Abbildung 3.1 zeigt den Aufbau des Rahmenwerkes von [Mül10] sowie die wesentlichen Komponenten. Die erste Schicht bildet die Anwendungen ab, welche die Kontextinformationen verwenden. Diese Anwendungen sind lokal auf dem mobilen Gerät angesiedelt und benutzen dabei teilweise entfernte Dienste, meist zum Erweitern des Funktionsumfangs, welches in der Grafik durch den entfernten Dienst C dargestellt ist. Über eine Schnittstelle sind die Anwendung mit dem darunter liegenden lokalen Kontextdienst verbunden, der sich ebenfalls auf dem mobilen Gerät befindet. Diese Schnittstelle ist für die Entwickler von Bedeutung, da sie diese in ihren Applikationen verwenden.
Der lokale Kontextdienst ist eine Komponente des Rahmenwerkes und stellt Kontextinformationen für die Anwendungen bereit. Dies bedeutet Anwendungen können über ihn Kontexte oder Sensorwerte abfragen. Die Sensoren des jeweiligen Gerätes müssen dafür mit dem lokalen Kontextdienst verbunden sein. Hierzu werden wiederum Kontextquellen verwendet, eine weitere Abstraktionsebene die es ermöglicht, die spezifische Implementierung zum Abfragen der einzelnen Sensoren vor dem lokalen Kontextdienst zu verbergen. Somit wird es ermöglicht, weitere Kontextquellen hinzuzufügen oder zu modifizieren, ohne dafür Änderungen an dem lokalen Kontextdienst vornehmen zu müssen. Dies ist wichtig, da sich ständig Änderungen an den Sensoren ergeben, entweder durch unterschiedliche Implementierungen der darunter liegenden Plattform oder durch neuartige Sensoren. Die Kommunikation zwischen diesen Komponenten wurde auch hier über Schnittstellen geregelt. Wie in der Grafik zu sehen ist, handelt es sich bei den Sensoren nicht nur um Hardwaresensoren des mobilen Gerätes, sondern auch um andere externe Informationsquellen.
Die letzte Komponente in dem Gesamtgebilde ist der entfernte Kontextserver. Hier werden die Kontexte modelliert und für den lokalen Kontextdienst bereitgestellt. Zur Abbildung der Kontexte werden ontologiebasierte Modelle verwendet. Die Verlagerung der Kontextabbildung auf den entfernten Kontextserver gegenüber der Abbildung auf dem lokalen Gerät bringt den Vorteil, dass auch bei umfangreichen Kontextmodellen keine Grenzen durch die Limitierung von Performance und Speicher auf dem mobilen Gerät gesetzt werden. Des Weiteren wird dadurch ein geräteübergreifender Austausch von Kontextinformationen möglich.

Eine konkrete Interaktion einer Anwendung mit dem Rahmenwerk kann nach dem folgenden Prinzip ablaufen. Anwendung A möchte alle aktiven Kontexte[2] ermitteln. Über die Schnittstelle des lokalen Kontextdienstes wird eine Abfrage, ausgehend von der Anwendung, gestartet. Der lokale Kontextdienst ermittelt über die Kontextquellen, welche die

[2]Alle Kontexte die zum jetzigen Zeitpunkt zutreffen

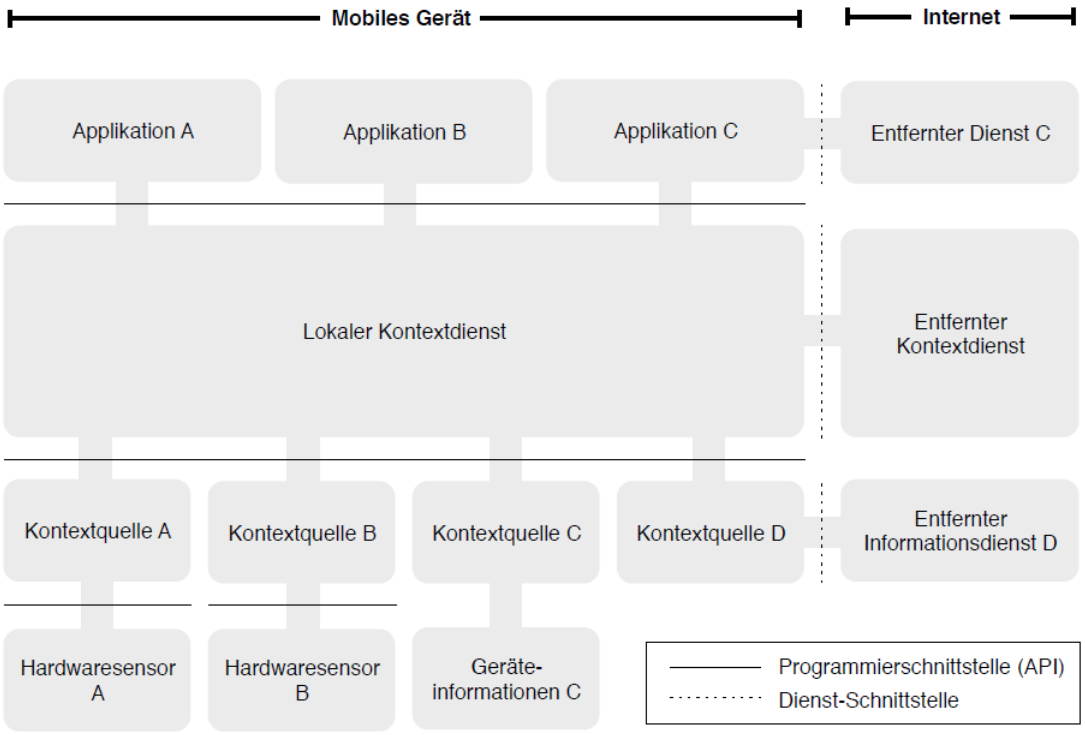


Abbildung 3.1: Konzeptioneller Aufbau des verwendeten Rahmenwerkes von [Mül10]

konkrete Implementierung zum Abfragen der Sensoren beinhalten, alle aktuellen Sensorwerte. Mit diesen Informationen stellt der lokale Kontextdienst, über das Internet, eine Abfrage an den entfernten Kontextserver. Dieser ermittelt aus den übergebenen Daten die aktuellen Kontexte und schickt sie zurück an den lokalen Kontextdienst, der sie wiederum an die Anwendung weitergibt.

3.3 Ansatzpunkt für diese Arbeit

Aus dem konzeptionellen Aufbau lässt sich ableiten, dass mehrere Anwendungen das Rahmenwerk benutzen, somit wäre die Betrachtung des Multiprozessbetriebes durch diese Arbeit überflüssig. Allerdings wurde der Prototyp in der Arbeit von [Mül10] mit einer Singletasking-Plattform umgesetzt, d.h. es konnte noch nicht evaluiert werden, ob diese Architektur den Ansprüchen eines Multiprozessbetriebes mit mehreren kontextsensitiven Anwendungen gewachsen ist. Es bietet sich also an, dieses Konzept unter Betrachtung der Aspekte des Multiprozessbetriebes zu untersuchen und gegebenenfalls zu modifizieren.

Des Weiteren muss eine Evaluation erfolgen, um sicherzustellen, dass der Multiprozessbetrieb mittels des Rahmenwerkes möglich ist. Dies kann nur durch die Durchführung eines Testlaufs, bei der mindestens zwei Anwendungen das Rahmenwerk zum gleichen Zeitpunkt verwenden, überprüft werden.
Es wurde in diesem Abschnitt klar gestellt, dass die Konzeption nach Aspekten des Multiprozessbetriebes und dessen Evaluation erfolgen muss, um diese Thematik zu behandeln. Allerdings ist noch nicht geklärt, was die Aspekte eines Multitaskbetriebes von mehreren kontextsensitive Anwendungen sind. Mit der Klärung dieser Frage beschäftigt sich das nächste Kapitel.

3.4 Zusammenfassung

In diesem Kapitel wurden Ziele, Komponenten und die Funktionsweise des Rahmenwerkes vorgestellt. Danach erfolgte ein kurzer Einblick, wie diese Arbeit versucht an das bestehende Rahmenwerk anzuknüpfen. Da alle Grundlagen besprochen wurden, kann nun mit der eigentlichen Untersuchung der Thematik dieser Arbeit begonnen werden.

4 Szenarien

Um die Fragestellung zu beantworten, inwiefern ein Multiprozesssystem für dieses Rahmenwerk benötigt wird, welche Aspekte in dem momentanen Stand der Domäne vorliegen und welche Anforderungen an ein Rahmenwerk für kontextsensitive Anwendungen in einer Multiprozessumgebung gestellt werden, kommen hier Szenarien zum Einsatz. Szenarien sind Geschichten über Menschen und ihre Aktivitäten [Car99] und werden in der Mensch-Computer-Interaktion eingesetzt, um funktionale und non-funktionale Anforderungen zu ermitteln [Pla05]. Szenarien bieten die Möglichkeit sich den oben genannten Fragen, in einem für diese Arbeit ausreichenden Rahmen, zu nähern oder diese zu beantworten, auch wenn sie keine vollständige Exerzierung eines Vorgehensmodells des Usability Engineering ersetzen können. Zusätzlich bieten sie die Vorteile, dass sie:

- ein Systemdesign spezifizieren, ohne dabei auf tiefere Details der Umsetzung durch das System eingehen müssen [Car99],
- es durch die narrative Form erlauben, reichhaltige und detaillierte Vorstellungen der Bedürfnisse der Benutzer zu erhalten [Pla05],
- zeigen, wie eine Allokation beschaffen sein sollte [Pla05] und
- für den vorhanden zeitlichen Rahmen für diese Arbeit angemessen erscheinen.

Im Rahmen dieser Arbeit werden Problemszenarien verwendet, da sie ein deskriptives Modell liefern [Pla05], anhand dessen mit der folgenden Analyse Vor- und Nachteile heraus gearbeitet werden. Daraus werden Anforderungen extrahiert, welche für den weiteren Verlauf der Arbeit verwendet werden, um das System zu gestalten.
Die Szenarien im Folgenden beziehen sich speziell auf Anwendungen für mobile Systeme, um die Anforderungen für diesen Bereich aufzudecken. Dabei wird darauf geachtet, dass eine möglichst hohe Vielfalt von unterschiedlichen Kategorien aus Kapitel 2.3 verwendet wird und eine genauere Betrachtung von Situationen für den Gebrauch parallel laufender Anwendungen stattfindet. Die Tabelle 4.1 zeigt eine Übersicht der verwendeten Kategorien für die beiden Szenarien, aufgeteilt in Kategorien des Kontextes und Kategorien der Anwendungen, nach dem Prinzip von [DAK00].

Ak=Aktivität, I=Identität, U=Umgebung, Z=Zeit P=Präsentation, Au=Ausführung, M=Makieren

ANWENDUNGSBEZEICHNUNG	FUNKTION	AK	I	U	Z	P	AU	M
Navigationssystem	Leitet den Anwender zu einem gewünschten Ziel	X	X	X		X		
Stick-e Document System	Ermöglicht das Lesen und Abrufen von digitalen Notizzetteln	X	X	X	X	X		X
Adaptierende Anwendung	Passt automatisch Systemeinstellungen an	X		X	X		X	
Friendlocator	Lokalisiert befreundete Personen und stellt diese auf einer Karte dar. Zusätzlich wird eine Kommunikation ermöglicht	X	X	X		X		
Datefinder	Vergleicht Profile von in Nähe befindlichen Personen und erleichtert deren Zusammenkunft	X	X	X		X	X	

Tabelle 4.1: Übersicht der verwendeten Anwendungen nach dem Schema von [DAK00]

4.1 Szenario: Ein durchschnittlicher Arbeitstag von Thomas

Es ist Montag um 7.00 Uhr früh, das Handy von Thomas schlägt Alarm. Er steht auf, da er sich fertig für die Arbeit machen muss. Nachdem er das Haus verlassen hat, steigt er in seinen Audi A3 und steckt sein Handy in die Halterung. Er schaltet das Navigationssystem seines Handys an und sucht in seinen favorisierten Fahrtzielen nach der Adresse seiner Arbeitstelle. Er wählt die entsprechende Adresse aus. Sein Handy zeigt ihm an, dass die Route berechnet wird, während er den Motor startet und los fährt. Nach ca. 20 Minuten zeigt sein Navigationssystem ihm an, dass er die Autobahn verlassen soll, um über einige Landstraßen dem vor ihm aufkommenden Stau auszuweichen. Fast pünktlich kommt er an seiner Arbeitsstelle an. Er nimmt das Handy aus der Halterung und schaltet die Navigationssoftware aus. Beim Betreten des Büros begrüßt er Tim, seinen Arbeitskollegen, der ihm gerade entgegen kommt, um sich einen Kaffee zu kochen. Er wechselt kurz einige Worte mit Tim und geht dann zu seinem Arbeitsplatz, an dem er seinen PC startet. Während sein Computer hochfährt, startet er seine Stick-e Document[1] Anwendung auf seinem Handy, damit er standortbezogene Notizen seiner Kollegen erhalten kann. Er liest

[1] siehe Brown [Bro96]

seine neuen E-Mails und fängt danach an, seine aktuellen Projekte zu bearbeiten. Um 10 Uhr wird Thomas über sein Handy erinnert, dass in einer halben Stunde das Meeting mit seinen Kollegen los geht. Thomas druckt noch schnell die wichtigen Unterlagen, die er für das Meeting benötigt, aus. Sein Computer meldet ihm an, dass der Druckauftrag nicht ausgeführt werden konnte. Thomas geht in den Flur zum Drucker, um das Problem festzustellen. Als Thomas sich dem Drucker nähert, teilt ihm seine Stick-e Document Anwendung mit, dass es eine Nachricht für seine aktuelle Position gibt, die ihm sein Kollege Tim hinterlassen hat.

“Liebe Kollegen, Drucker 534 ist zurzeit defekt. Bitte verwendet Drucker 535.“

Thomas geht zurück zu seinem Arbeitsplatz druckt die Unterlagen auf dem anderen Drucker und macht sich auf den Weg zum Konferenzraum. Die ersten Kollegen warten schon auf ihn, an dem runden Konferenztisch. Thomas begrüßt die Kollegen und setzt sich. Er erhält nun über sein Handy ein kurzes Signal, dass dieses alle akustischen Signale unterdrückt um Störungen zu vermeiden. Nachdem Meeting geht Thomas zurück zu seinem Arbeitsplatz. Sein Handy wechselt wieder zum vorherigen Profil. Am seinem Arbeitsplatz angekommen, beschäftigt sich Thomas mit den neuen Themenbereichen, die er in der Sitzung bekommen hat. Als er sich an dem System für sein neues Projekt anmelden möchte, bemerkt er, dass er seine Logindaten nicht mehr weiß. Zum Glück hat er sich während des Meetings eine Notiz gesetzt, die, wenn er am Arbeitsplatz ist, erscheint. Thomas ruft die vorhandenen Nachrichten für seinen Standort ab und findet Notizen zu seinem Passwort. Er arbeitet nun noch bis 17 Uhr an dem neuen Projekt, bis er sich wieder aufmacht, zu seinem Auto zu kommen. Er startet sein Navigationssystem und fährt los. Während der Fahrt signalisiert ihm seine E-Stick-Anwendung, dass er weitere Nachrichten erhalten hat, wodurch ihm auffällt, dass er vergessen hat die Software zu beenden, als er das Büro verlassen hat. Er schaltet diese in der Regel nach der Arbeit immer aus, damit sein Handyakku nicht so schnell leer ist. Zuhause angekommen macht er sich etwas zu Essen und verbringt einen ruhigen Abend vor dem Fernseher.

4.2 Analyse des Szenarios von Thomas

Die Analyse des Szenarios wurde in Tabelle 4.2 festgehalten.

Nr.	Ereignis	Positive Aspekte	Negative Aspekte
1a	An- und Abschalten des Navigationssystems		Das An- und Abschalten muss vom Benutzer vorgenommen werden
2a	Einstellen des Fahrziels	Favoriten helfen bei der schnelleren Auswahl	Die Auswahl muss vom Benutzer getroffen werden
3a	An- und Abschalten Stick-e Document-Anwendung		Standortbezogene Anwendungen müssen nicht ständig laufen, so kann das An- und Abschalten der Software zur lästigen Aufgabe des Benutzers werden
4a	Abruf der Druckernotiz	Notiz nimmt dem Benutzer die Fehlersuche ab	Eine Erkennung der Situation, würde des dem Nutzer ersparen, sich zum Drucker zu begeben
5a	Abruf Logindaten	Sensible Daten sind nicht auf dem Computer gespeichert, dadurch kann erhöhte Sicherheit erreicht werden	Wirft Sicherheitsfragen auf für das Abrufen der Daten oder bei Verlust des Gerätes
6a	Gleichzeitiges Laufen der Anwendungen Stick-e Document und Navigationssystem		Unnötiger Stromverbrauch, durch laufende Anwendungen die nicht benötigt werden. Mögliche Performanceeinbußen, durch nicht benötigte Anwendungen. Redundantes Abrufen von Standortsinformationen.

Tabelle 4.2: Auflistung der negativen und positiven Aspekte aus dem Szenario von Thomas

4.3 Szeanrio: Ein Partywochenende von Marco und Sven

Marco ist auf den Weg in die Stadt. Er hat schon um 18.23 Uhr die Bahn genommen, um pünktlich am Lenauplatz anzukommen. Er trifft sich dort vor der Bar “Zeitsprung“ mit seinem Kumpel Sven. Sven hatte die Bar letzte Woche ausgemacht und zusammen

wollten sie ein paar Bier trinken und danach vielleicht noch in der Umgebung befindliche Clubs ausfindig machen. Mit der Bahn am Lenauplatz angekommen, zückt Marco sein Handy und startet die Navigationssoftware, da er noch nie in der Bar war und sich somit sich nicht wirklich sicher ist, wo diese sich befindet. Die Navigationssoftware leitet Marco zu der gewünschten Bar. Während er zur Bar läuft, startet er seinen “Friendlocator“, um nachzuschauen ob Sven schon da ist. Über den “Friendlocator“ wird ihm Svens Position und die von anderen Personen aus seiner Freundesliste auf einer Karte angezeigt. Marco wählt Sven aus und sendet ihn eine Nachricht über den “Friendlocator“. Nachdem Sven ihm gesagt hat, dass er in 5 Minuten da ist beschließt er sich schon in die Bar zu setzen. Er bestellt die erste Runde für sich und seinen Kumpel. Sven trifft zeitgleich mit den ersten zwei Bieren ein. Beide begrüßen sich und reden in der Bar über ihre Erlebnisse der letzten Tage. Da das Lokal nicht besonders gut gefüllt ist, entscheiden sich Sven und Marco den Abend noch woanders zu verbringen. Marco startet seinen “Friendlocator“ und sucht nach Freunden in der Umgebung. Marie wohnt ungefähr 15 Minuten von der Bar entfernt und wird vom “Friendlocator“ in der Nähe angezeigt. Marco schreibt Marie an und fragt, was sie gerade macht. Marie berichtet, dass sie heute Abend eine kleine Party veranstaltet, die aber erst um 21 Uhr anfängt. Sie beschließen beide auf die Party zu gehen. Da es aber gerade erst 20 Uhr ist und sie die Zeit bis dahin überbrücken wollen, suchen sie im Internet über den Browser nach möglichen Bars in der Umgebung. Nachdem sie für sich eine attraktive Tanzbar ausgemacht haben, verlassen sie die Bar. Marco gibt in sein Navigationssystem die Adresse der Tanzbar ein. Danach minimiert er sein Navigationssystem und startet seinen “Datefinder“. Marco und Sven folgen den auditiven Angaben des Navigationssystems. Fast an der Bar angekommen erhält Marco ein Signal und wechselt vom Navigationssystem auf den “Datefinder“. Das Navigationssystem läuft weiter und gibt ihnen auditive Weganweisungen. Der “Datefinder“ zeigt ihm an, dass Melanie sich in der Nähe befindet. Marco fragt über den “Datefinder“ an, ob sie sich kennenlernen wollen. Melanie bestätigt die Anfrage, nachdem sie sich Marcos Profil angeguckt hat. Sie unterhalten sich kurz über den “Datefinder“ und machen dann die Tanzbar als gemeinsamen Treffpunkt aus. In der Tanzbar angekommen, verbringen Melanie, ihre zwei Freundinnen, Marco und Sven einen witzigen Abend. Irgendwann zu später Stunde tauschen sie ihre Kontaktdaten aus und Melanie verlässt mit ihren Freundinnen die Tanzbar. Sven ruft bei Marie an, um die Adresse zu erfragen. Marco gibt in die Adresse ins Navigationssystem ein und beide machen sich auf den Weg zu Marie. Dort endet dann auch ihr Abend nach ein paar schönen Stunden auf der Party.

4.4 Analyse des Szenarios von Marco und Sven

Die Analyse des Szenarios wurde in Tabelle 4.3 festgehalten.

Nr.	Ereignis	Positive Aspekte	Negative Aspekte
1b	Gleichzeitig laufende Anwendung Friendlocator und Navigationssoftware	Navigationssystem schaltet auf Audiosignale um, wenn es minimiert wird	Lokalisierung erfolgt zwei mal, obwohl Standortinformationen schon für eine Anwendung abgerufen wurde
2b	Gleichzeitig laufende Anwendungen Datefinder und Navigationssoftware	Navigationssystem schaltet auf Audiosignale um, wenn es minimiert wird	Austausch der Daten zwischen den Anwendungen ist nicht möglich Datefinder sucht in der Umgebung nach Personen, aber nicht an möglichen Zielorten. Redundante Standortsanfragen
3b	Abrufen von Informationen aus dem Internet und dessen Eingabe in das Navigationssystem		Informationen sind eigentlich schon bekannt, müssen aber erneut eingegeben werden

Tabelle 4.3: Auflistung der positiven und negativen Aspekte aus dem Szenario von Marco und Sven

4.5 Fazit

Die Szenarien und deren Analyse zeigen zum einen, dass sich Situationen ergeben, in denen mehrere Anwendungen parallel Kontextinformation benötigen und zum anderen, dass noch einige Nachteile bei einer gleichzeitigen Nutzung von unterschiedlichen kontextsensitiven Anwendungen existieren. Hinzu kommt, dass diese Anwendungen auch durch den Austausch oder die Wiederverwendung von Informationen profitieren können. Zum Beispiel können im Szenario des parallel laufenden “Datefinder“ und Navigationssoftware Informationen ausgetauscht und somit der Suchradius des “Datefinders“ um den Zielort erweitert werden. Zusätzlich ergibt es die Möglichkeit durch die Wiederverwendung der Standortinformationen[2], wiederholende Anfragen von unterschiedlichen Anwendungen auf die Sensoren der Geräte zu reduzieren. Hinzu kommen Automatismen, die bei einer steigenden Anzahl von kontextsensitiven Anwendungen auf einem System, dem Nutzer eine Erleichterung bei der Handhabung bringen können.

[2]Nicht nur Standortinformationen, sondern alle Informationen, die von mehreren Anwendungen benötigt werden, können zwischengespeichert und wiederverwendet werden, so lange sie aktuell sind.

4.6 Zusammenfassung

In diesem Kapitel wurden einige Vor- und Nachtteile der jetzigen Situation aufgedeckt und festgehalten. Des Weiteren konnte der Bedarf an einer Multiprozessumgebung für das Rahmenwerk ermittelt werden. Im nächsten Kapitel können nun auf Basis dieser Erkenntnisse Anforderungen an das Rahmenwerk unter Berücksichtigung der Multiprozessumgebung extrahiert werden.

5 Anforderungen

In diesem Kapitel werden aus den gewonnen Erkenntnissen die Anforderungen für das Rahmenwerk extrahiert. Zuerst werden funktionale Anforderungen, dann die non-funktionalen Anforderungen ermittelt. Danach werden die gefundenen Anforderungen, durch die noch nicht genannten Anforderungen aus der Arbeit von Felix Müller, ergänzt. Für die Anforderungen wird dann eine Priorisierung durchgeführt, so dass auf deren Basis das Rahmenwerk für den Multiprozessbetrieb entworfen werden kann.

5.1 Funktionale Anforderungen

Automatisches Starten und Beenden von Anwendungen – Das automatische Starten und Beenden von Anwendungen kann dem Benutzer eine Erleichterung mit der täglichen Handhabung von kontextsensitiven Anwendungen sein. Des Weiteren kann Energie und Performance eingespart werden, wenn nicht benötigte Anwendungen abgeschaltet und erst bei Bedarf wieder angeschaltet werden.

Aktivieren und deaktivieren des Wahrnehmen von Kontextinformationen – Es kann durchaus möglich sein, dass der Nutzer sich in Situationen befindet in denen er nicht möchte, dass Informationen über ihn gesammelt werden oder auch beispielsweise den Stromverbrauch für sein mobiles Gerät senken möchte. In diesem Fall sollte es für den Benutzer möglich sein, das Sammeln von Information auf seinem Gerät zu unterbinden.

Speichern von vergangenen Informationen – Durch sich immer wiederholende Vorgänge können Anwendungen ihr Verhalten anpassen, da diese Aktionen der Benutzer vorhersehbar werden. So wäre es beispielsweise, durch die Information, dass Thomas immer jeden Werktag, zu ungefähr der gleichen Zeit, zur Arbeit fährt, für das Navigationssystem möglich, die Adresse von Thomas Arbeitsstelle als Zielpunkt von vornherein auszuwählen oder die Auswahl der Adresse für den Benutzer zu vereinfachen. Um dies zu ermöglichen müssten Kontextinformationen über längere Zeit gespeichert werden, um Rückschlüsse auf das Verhalten der Benutzer schließen zu können.

Wiederverwendbarkeit von Daten – Daten die gerade erst ermittelt wurden, wie der aktuelle Standort des Benutzers, können auch noch wenige Augenblicke später verwendet werden, solange sie noch aktuell sind oder ein gewisses Maß an Abweichung irrelevant ist. Dies kann an dem Szenario mit Marco erkannt werden, in dem der Datefinder und das Navigationssystem gleichzeitig laufen. Das Navigationssystem benötigt ständig aktuelle Ortsinformationen, ebenso wie der Datefinder, der bei jeder Abfrage Ortsinformationen ermitteln muss. Da es für den Datefinder nicht relevant ist, ob diese Information möglicherweise eine Abweichung von mehreren Metern beinhaltet, kann die Anwendung die Ortsinformationen der Navigationssoftware benutzen, selbst wenn sich die Position des Nutzers um einige Meter seit der letzten Abfrage verändert hat. Durch das Wiederverwenden von Daten, besonders anwendungsübergreifend, können mögliche redundante Abrufe vermieden und somit Performance und Energie eingespart werden.

Anreicherung der Kontextinformationen durch die Anwendungen und Austausch von Informationen zwischen den Anwendungen – Im zweiten Szenario hat Marco sein Navigationssystem in Betrieb und sucht gleichzeitig über den Datefinder nach möglichen Bekanntschaften. Der Datefinder verfügt dabei allerdings nur über die Information, an welchem Standort sich die Marco und Sven befinden und kann somit auch nur potentielle Kontakte für diesen Standort ermitteln. Allerdings ist für Sven und Marco ihr schon bereits ausgewähltes Ziel von Interesse, da sie sich zu einem späteren Zeitpunkt dort aufhalten werden. Der Datefinder könnte also, wenn er über diese Information verfügen würde, in der Suche auch den zukünftigen Standort der beiden berücksichtigen. Dies kann realisiert werden, indem die Navigationssoftware das Ziel, über das Rahmenwerk, für andere Anwendungen bekannt gibt. Durch diese Funktion wird eine indirekte Kommunikation zwischen den Anwendungen über das Rahmenwerk möglich.

5.2 Non-funktionale Anforderungen

Zugriff mehrerer Anwendungen auf Kontextinformationen – Die Szenarien haben gezeigt, dass mehrere Anwendungen auch zum gleichen Zeitpunkt Kontextinformationen benötigen. Das Rahmenwerk soll so gestaltet sein, dass dies uneingeschränkt möglich ist.

Berücksichtigung der eingeschränkten Stromversorgung – Durch ständiges Abrufen von Sensoren und mehreren gleichzeitig laufenden Anwendungen, kann es zu einem erhöhten Stromverbrauch auf mobilen Geräten kommen. Das ständige Abrufen von Positi-

onsdaten, zum Beispiel per GPS, führt zu einem hohen Stromverbrauch[1] und somit zu einer schnell abnehmenden Akkulaufzeit des Mobiltelefons. Aus diesem Grund sollte das Rahmenwerk möglichst stromsparend konzipiert werden.

Erweitern der Kontextinformationen durch externe Sensoren – Im ersten Szenario muss Thomas erst zu seinem Drucker gehen, um das Problem zu erfassen. Seine Anwendung könnte von externen Sensoren profitieren, welche in diesem Fall der PC darstellen würde, indem er der Anwendung mitteilt, dass Thomas drucken möchte. Sobald die Information angekommen ist, würde die Anwendung die Notiz für Thomas bereit stellen, in der er über den defekten Drucker informiert wird.

Performance – Das Rahmenwerk soll möglichst performant aufgebaut sein, sodass auch beim Zugriff von mehreren Anwendungen zum gleichen Zeitpunkt, eine angemessene Antwortzeit gewährleistet wird.

Sicherheit und Schutz der Privatsphäre – Im ersten Szenario speichert Thomas seine Logindaten über die Document e-Stick-Anwendung ab. Dies wirft Sicherheitsfragen auf. Zum einen erfordern vertrauliche Daten besondere Sicherheitsvorkehrungen, zum anderen kann es bei kontextsensitiven Anwendungen zu einem regelmäßigen Gebrauch von personenbezogenen Daten kommen, welche ebenso vor den Zugriffen von Dritten geschützt werden müssen.

5.3 Vorgegebene Anforderungen des Rahmenwerkes

Die Anforderungen, die innerhalb der Arbeit von Felix Müller [Mül10] festgelegt wurden, werden in diesem Abschnitt zusammen mit einer Kurzbeschreibung aufgeführt, da diese auch für die Konzeption des lokalen Kontextdienstes unter der Berücksichtigung einer Multiprozessumgebung gelten. Es werden nur Anforderungen aufgeführt, die noch nicht durch die Szenarien ermittelt wurden.

- Verwaltung von Kontextinformationen – Umfasst das Abrufen, Erstellen, Ändern und Löschen von Kontexten. Beim Abrufen des Kontextes wird zwischen der Pull- und Push-Methode unterschieden. Bei der Pull-Methode wird explizit nach Daten gefragt, während Benachrichtigungen über die Push-Methode realisiert werden.
- Erweiterbarkeit und Skalierbarkeit – Das System soll um beliebige Kontexte und Kontextquellen erweiterbar sein.

[1] siehe [BA09]

- Trennung von Kontextmanagement und Kontextmodell – Beide Komponenten sollen voneinander getrennt und austauschbar sein.
- Kompaktheit und Portierbarkeit – Das System soll möglichst auf vielen unterschiedlichen Plattformen einsetzbar sein.
- Berücksichtigung der Kontextqualität – Die Daten der Kontextquellen bzw. Sensoren sollen durch Metadaten beschrieben werden können, um deren Qualität sicherzustellen.

5.4 Priorisierung der Anforderungen

Durch die Szenarien wurden Anforderungen ermittelt, welche zum Teil über den Rahmen dieser Arbeit hinausgehen. Eine Priorisierung der einzelnen Anwendungen soll bei der Unterscheidung zwischen relevanten und nicht relevanten Anforderungen helfen. Ihnen wurde jeweils eine Priorität zwischen 0 und 2 zugeordnet, wobei 0 für keine weitere Beachtung, 1 für normale Beachtung und 2 für große Beachtung steht.

- Das automatische Starten und Beenden ist ein zentraler Punkt bei der Verwendung von mehreren kontextsensitiven Anwendungen. Es trägt dazu bei, das System performanter zu gestalten und die Laufzeit des mobilen Gerätes zu verlängern, deswegen wird diesem Punkt eine hohe Priorität zugeordnet.
- Das Aktivieren und Deaktivieren des Wahrnehmens von Kontextinformationen ist ein weiterer Punkt, der zum Schutz der Privatsphäre beitragen kann. Er wird deshalb bei der Konzeption berücksichtigt.
- Das Speichern von vergangenen Informationen, um Rückschlüsse auf das Verhalten der Benutzer führen zu können, erhält für den weiteren Verlauf dieser Arbeit keine Priorität, da es für die eigentliche Betrachtung des Mehrprozessbetriebes des Rahmenwerkes keine Bedeutung hat.
- Die Wiederverwenbarkeit der Daten ist ein wichtiger Punkt, den man in den Bereich “Caching“ einordnen kann.
  “Ein intelligenter Caching-Mechanismus etwa, könnte die Probleme beseitigen, dass der Kontextserver bei schlechten Netzwerkverbindungen nicht erreichbar sein kann und dass ständiges Abfragen des Servers die Batterielaufzeit der mobilen Gerätes reduziert.“ [Mül10]
  Der Bereich des Cachings ist zu umfangreich, als dass man ihn im Rahmen dieser Arbeit behandeln könnte und wurde deswegen mit 0 priorisiert.

- Die Anreicherung der Kontextinformationen durch die Anwendungen und Austausch von Informationen zwischen den Anwendungen birgt ein hohes Potential für das Rahmenwerk und schafft darüber hinaus einen Mehrwert. Diese Funktion wird deshalb mit hoher Priorität behandelt.
- Der Zugriff mehrerer Anwendungen auf Kontextinformationen ist ein Kernpunkt dieser Arbeit und wird deshalb auch mit hoher Priorität behandelt.
- Die Berücksichtigung der stark eingeschränkten Batterielaufzeit soll für die Konzeption des lokalen Kontextdienstes eine Rolle spielen und erhält deswegen normale Beachtung.
- Erweitern der Kontextinformationen durch externe Sensoren wurde schon in der Arbeit von Felix Müller konzeptionell berücksichtigt und wird deshalb nicht mehr genauer untersucht.
- Die Performance spielt beim Mehrprozessbetrieb besonders auf mobilen Geräten eine beachtliche Rolle und soll bei der Konzeptionierung berücksichtigt werden. Dieser Punkt erhält deshalb auch eine Priorisierung von 1.
- Auf den Punkt Sicherheit wird in dieser Arbeit nicht weiter eingegangen, da dieser Bereich zum einen zu umfangreich ist und zum anderen nicht zum Kern der Arbeit gehört.

5.5 Zusammenfassung

In diesem Kapitel wurden die Anforderungen aus den Szenarien extrahiert und vorgestellt. Danach folgte eine Priorisierung der Anforderungen, um deren Relevanz für den weiteren Verlauf der Arbeit festzuhalten. Dabei hat sich herausgestellt, dass einige Anforderungen nicht weiter berücksichtigt werden, da diese vom Umfang her, die Untersuchung durch weitere Arbeiten erfordern, nicht in den Rahmen dieser Arbeit passen oder schon in der Arbeit von Felix Müller behandelt wurden. Die Tabellen 5.1 und 5.2 zeigen noch einmal alle gefunden Anforderungen mit der Priorisierung und ihrem Ursprung aus Kapitel 4 im Überblick.

ANFORDERUNG	URSPRUNG	PRIORITÄT
Automatisches Starten und Beenden von Anwendungen	1a, 6a, 3a	2
Aktivieren und Deaktivieren des Wahrnehmens von Kontextinformationen	siehe Schutz der Privatsphäre	1
Speichern von vergangenen Informationen	2a	0
Wiederverwendbarkeit von Daten	6a, 1b, 2b	0
Anreicherung der Kontextinformationen durch die Anwendungen und Austausch von Informationen zwischen den Anwendungen	2b, 3b	2

Tabelle 5.1: Alle durch die Szenarien gefundenen funktionale Anforderungen im Überblick

ANFORDERUNG	URSPRUNG	PRIORITÄT
Zugriff mehrerer Anwendungen auf Kontextinformationen	6a, 1b, 2b	2
Berücksichtigung der stark eingeschränkten Akkulaufzeit	6a, 1b, 2b	1
Erweitern der Kontextinformationen durch externe Sensoren	4a	0
Performance	6a	1
Sicherheit und Schutz der Privatsphäre	5a	0

Tabelle 5.2: Alle durch die Szenarien gefundenen non-funktionalen Anforderungen im Überblick

6 Konzeption

In diesem Kapitel werden Anforderungen, die unter Berücksichtigung des Multiprozessbetriebes ermittelt wurden, auf eine Umsetzung mit dem bestehenden Rahmenwerks untersucht. Dafür werden alle Anforderungen mit einer technischen Perspektive betrachtet und für die Umsetzung überprüft. Damit zusammenhängend sollen Änderungen, wenn notwendig, vorgenommen werden, um das Rahmenwerk für einen Multiprozessbetrieb anzupassen. Ziel dieses Prozesses ist die Erschließung eines theoretischen Konzeptes, welches den Ansprüchen eines Multiprozessbetriebes für kontextsensitive Anwendungen erfüllt und auf dessen Basis anschließend konkrete Komponenten für eine praktische Umsetzung ausgewählt werden können.

6.1 Zu beachtende Aspekte und Komponenten

Da sich durch die Anforderungen einige Aspekte ergeben haben, welche die Architektur beeinflussen können, sollen diese hier noch einmal erwähnt werden, um einen besseren Überblick zu erhalten.

- **Allgemein** – Alle Punkte die im Weiteren behandelt werden sollen unter Berücksichtigung der non-funktionalen Anforderungen, je nachdem wie sie im Abschnitt 5.4 gewichtet wurden, ausgewählt werden. Des Weiteren soll hierbei darauf geachtet werden, dass alle funktionalen Anforderungen bei der ausgewählten Systemarchitektur umsetzbar sind.

- **Betrachtung von Plattformen und Programmiersprachen** – Für die Umsetzung des Prototyps und bei der Konzeption spielt die Plattform und die Programmiersprache eine wichtige Rolle, da je nach Plattform und Programmiersprache sich unterschiedliche Lösungsansätze ergeben können. Dieser Themenbereich wird in Abschnitt 7.1 behandelt.

- **Aufbau des lokalen Kontextdienstes** – Da bei einer Multiprozessumgebung unterschiedliche, parallel laufende Prozesse vorhanden sind, muss betrachtet werden, wie der lokale Kontextdienst aufgebaut werden soll und wo dieser angesiedelt wird, da sich verschiedene Vor- und Nachteile dabei ergeben.

- **Multitasking** – Das Rahmenwerk muss so konzipiert sein, dass es mehrere Anfragen, von unterschiedlichen Anwendungen, parallel bearbeiten kann.
- **Lebenszyklus und Abfragerhythmus** – Unter dem Aspekt der begrenzten Performance und Akkulaufzeit, muss überlegt werden, wann der lokale Kontextdienst auf dem mobilen Gerät laufen soll und inwiefern es notwendig ist, aktuelle Kontexte oder Sensoren abzufragen
- **Kommunikation und Schnittstellen** – Es muss abgewogen werden, welche Kommunikationsform benutzt wird. Des Weiteren sollen die Schnittstellen für die Kommunikation spezifiziert werden. Diese Thematik wird in Abschnitt 7.3.2 behandelt.
- **Persistente Speicherung** – Daten, die von dem Rahmenwerk verwendet werden, sollen auf eine dauerhafte Speicherung hin untersucht werden. Zusätzlich ist eine Auswahl einer angemessenen Methode zum Speichern der Daten erforderlich.
- **Informationsaustausch zwischen den Anwendungen** – Es soll überprüft werden inwiefern sich ein Informationsaustausch zwischen den Anwendungen realisieren lässt.
- **Systemeinstellungen im Multitaskbetrieb** – Es soll ermittelt werden, ob es notwendig ist, Einstellungen für den lokalen Kontextdienst anzubieten. Dabei sollte berücksichtigt werden, dass unterschiedliche Anwendungen, möglichweise gegensätzliche Einstellungen vornehmen.

6.2 Aufbau des lokalen Kontextdienstes

Für den lokalen Kontextdienst ergeben sich mehrere Möglichkeiten der Umsetzung. Beispielhaft lässt sich die Arbeit von Felix Müller[Mül10] aufführen, in welcher der lokale Kontextdienst konzeptionell als eine eigenständige Komponente und bei der Implementierung als statische Bibliothek umgesetzt wurde. Diese verschiedenen Möglichkeiten sollen in diesem Abschnitt unter der Berücksichtigung mehrerer Anwendungen, die den lokalen Kontextdienst nutzen, erörtert werden. Ziel dieser Erörterung ist es, einen geeigneten Aufbau für den lokalen Kontextdienst, unter Betrachtung des Multiprozessbetriebes, zu erarbeiten.

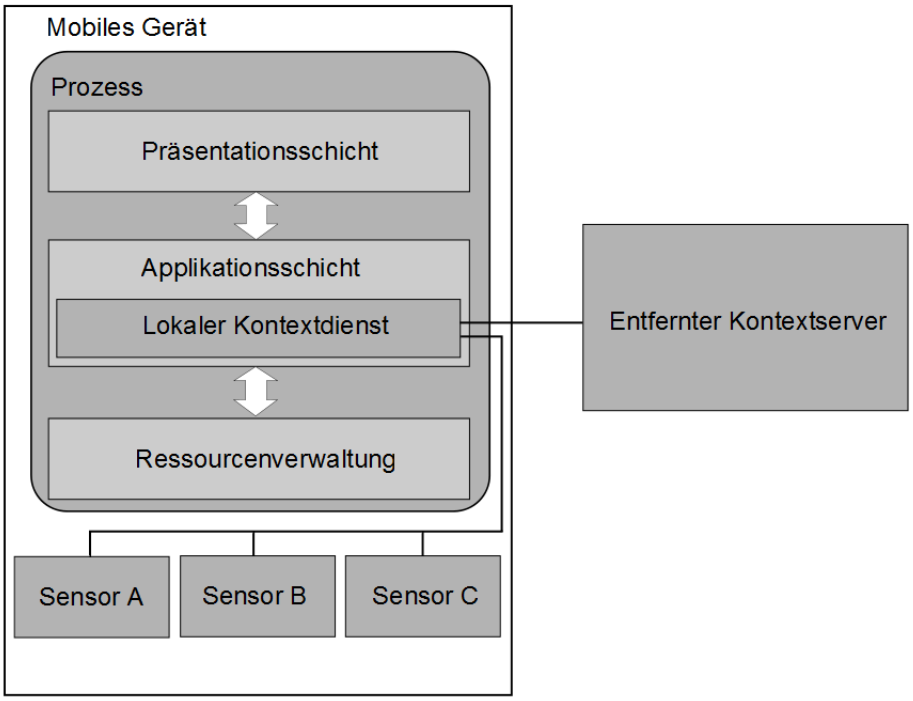


Abbildung 6.1: Abbildung des lokalen Kontextdienstes innerhalb einer Anwendung (modifiziert aus [Alo03])

6.2.1 Problematik mehrerer lokaler Kontextdienste

Grafik 6.1 zeigt eine Umsetzung des lokalen Kontextdienste innerhalb der Applikationsschicht einer Anwendung[1]. Die Anwendung verfügt neben der Applikationsschicht, über eine Präsentationsschicht zur Interaktion mit dem Benutzer und einer Ressourcenverwaltung zum Speichern von Daten.
Die Ansiedlung des lokalen Kontextdienstes innerhalb einer Anwendung bedeutet, dass er fest in der Anwendung integriert ist, die den lokalen Kontextdienst benutzt und somit ein Teil des Prozesses der Anwendung ist. Dies hat zur Folge, dass bei Änderungen an dem lokalen Kontextdienst, beispielsweise durch das Hinzufügen neuer Sensoren oder Funktionen, die Anwendung mit der aktuellen Version des Kontextdienstes neu übersetzt werden muss, um von diesen Änderungen profitieren zu können.
Unter Betrachtung einer Multiprozessumgebung, führt die feste Integration des Kontextdienstes in die Anwendung dazu, dass jede Anwendung ihren eigenen lokalen Kontextdienst besitzt, welcher wiederum mit dem externen Kontextserver und den Sensoren kommuniziert. Das Schaubild 6.2 zeigt einen Aufbau von mehreren Anwendungen mit einem

[1] Es sei angemerkt, dass Abbildung 6.1 nur einen möglichen Aufbau der Anwendung zeigt und viele weitere Variationen möglich sind.

integrierten Kontextdienst[2]. Jede Anwendung läuft dabei als ein einzelner Prozess und hat jeweils seine eigenen separaten Verbindungen zu den Sensoren und dem entfernten Kontextserver. Die Problematik besteht darin, dass sobald zwei Anwendungen zum gleichen Zeitpunkt auf eine Kontextquelle zugreifen oder Informationen vom Kontextserver abfragen, es zum redundanten Informationsaustausch kommt. Da das Abrufen von Informationen aus dem Internet und den Kontextquellen mit einem erhöhten Stromverbrauch verbunden ist, wirkt sich dies auch negativ auf die Akkulaufzeit des Gerätes aus.

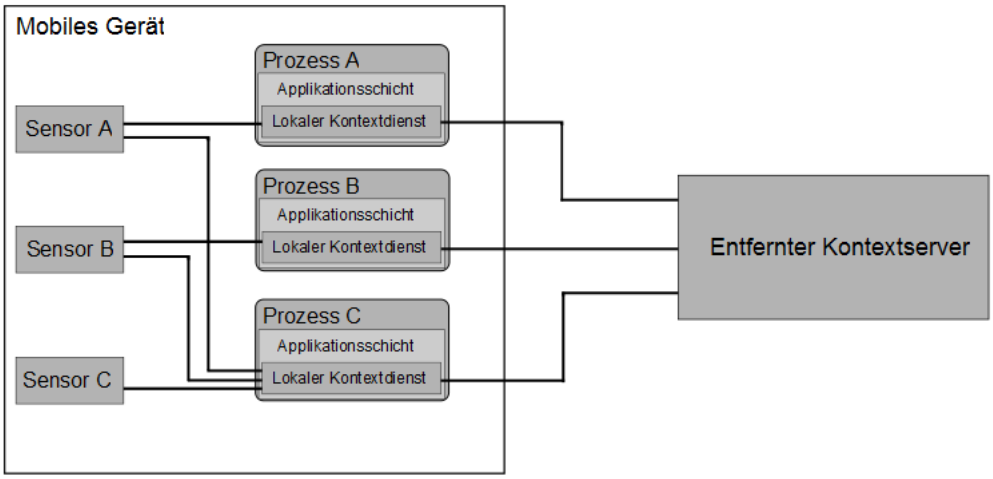


Abbildung 6.2: Abbildung mehrerer Anwendungen, die den lokalen Kontextdienstbeinhalten (modifiziert aus [Alo03])

6.2.2 Vernetzung der lokalen Kontextdienste

Um diese Probleme zu vermeiden, ist es notwendig, dass die einzelnen lokalen Kontextdienste miteinander kommunizieren. Allerdings birgt der Ansatz, alle Anwendungen miteinander zu verbinden, einige neue Probleme. Bei einer Vernetzung der einzelnen Kontextdienste sind unterschiedliche Topologien denkbar, wie es bei Netzwerken bekannt ist. Jede Architektur bietet dabei ihre eigenen Vor- und Nachteile unter der Betrachtung von Ausfällen, Fehleranfälligkeit und Performance, welche hier nicht näher erläutert werden sollen. Diese unterschiedlichen Aufbauten haben zur Folge, dass die Teilnehmer untereinander verbunden sein müssen, was einen administrativen Aufwand für den Endanwender oder Entwickler bedeutet. Hinzu kommt, dass weiterhin redundante Verbindungen zum externen Kontextdienst und zu den Kontextquellen bestehen. Weitere Probleme kön-

[2]Auf die Präsentationsschicht und Ressourcenverwaltung wurde in dieser Grafik für eine bessere Übersicht verzichtet

nen durch unterschiedliche Versionen des lokalen Kontextdienstes entstehen, denn für die Kommunikation der lokalen Kontextdienste, müssen diese untereinander kompatibel sein. Zusätzlich kann es dazu kommen, dass die lokalen Kontextdienste nicht alle die gleichen Sensoren verwenden und somit bei der Ermittlung von Kontexten unterschiedliche Ergebnisse erzielen. Ebenso stellt die Verwaltung ein Problem dar. Möchte der Benutzer beispielsweise eine Kontextquelle bei den Kontextdiensten deaktivieren oder eine andere Einstellung ändern, so muss er das in diesem Fall für jeden Kontextdienst einzeln durchführen.

6.2.3 Der lokale Kontextdienste als zentrale Komponente

Zur Lösung der eben aufgeführten Probleme, ist die Einführung einer zentralen Komponente angebracht. Um redundante Aufrufe des Kontextservers und der Kontextquellen zu vermeiden, ist ein Ansatz, die Kommunikation zwischen den Anwendungen über eine gemeinsame Komponente zu regeln. Dabei ist beispielsweise eine Datenbank denkbar, über welche die verschiedenen lokalen Kontextdienste kommunizieren können, indem sie vorher einsehen können, welche Aufrufe gerade getätigt werden und selber vermerken, welche Aufrufe sie durchführen. Allerdings bleibt so immer noch die Problematik der redundanten Verbindungen. Des Weiteren kann es sein, dass Anwendungen mit unterschiedlichen Versionen des lokalen Kontextdienstes weiterhin veraltete Datenstrukturen oder Algorithmen verwenden, da sie mit einer älteren Version des lokalen Kontextdienstes arbeiten. Damit diese Probleme gelöst werden können, muss die Applikationslogik des lokalen Kontextdienstes auf die zentrale Komponente ausgelagert werden. So ist es für die Applikationen möglich, immer mit der gleichen Version zu arbeiten. Bei einer Aktualisierung des lokalen Kontextdienstes wird sichergestellt, dass alle Anwendungen, die auf dem Gerät vorhandene, effektivste Implementierung des Dienstes verwenden. Zusätzlich wird es so ermöglicht redundante Aufrufe und Verbindungen zu vermeiden, da Ergebnisse von Abfragen der Kontextquellen und von Kontexten für spätere Zwecke zwischengespeichert werden können[3]. Des Weiteren werden die Ergebnisse für Abfragen des aktuellen Kontextes nicht verfälscht, da allen Anwendungen die gleiche Anzahl an Kontextquellen zur Verfügung steht. Ein weiterer positiver Aspekt ist eine erhöhte Wartbarkeit, da Fehler bzw. so genannte Bugs nur noch an einer Stelle auftreten und nicht an mehreren Stellen gleichzeitig. Ein Nachteil dieser Architektur bleibt allerdings. Dadurch dass der lokale Kontextdienst als eine eigene unabhängige Anwendung existiert, wird solange der Kontextdienst aktiv ist, mehr Performance benötigt. Gleiches gilt für prozessübergreifende Kommunikation, die durch die Auslagerung des lokalen Kontextdienstes erforderlich wird. Abbildung 6.3 zeigt den lokalen Kontextdienst als eigene Komponente in dem Gesamtsystem. Der lo-

[3] siehe Abschnitt 5.4

kale Kontextdienst läuft dabei, wie eine Anwendung, in einem eigenen Prozess. Bei einem Zugriff der Anwendungen fungiert er als Hintergrundprozess, der nicht sichtbar für den Anwender wird. Eine eigene Präsentationslogik, sowie Ressourcenverwaltung sei hier nur angedeutet, da zu diesem Zeitpunkt noch nicht festgelegt werden kann, ob diese benötigt werden.

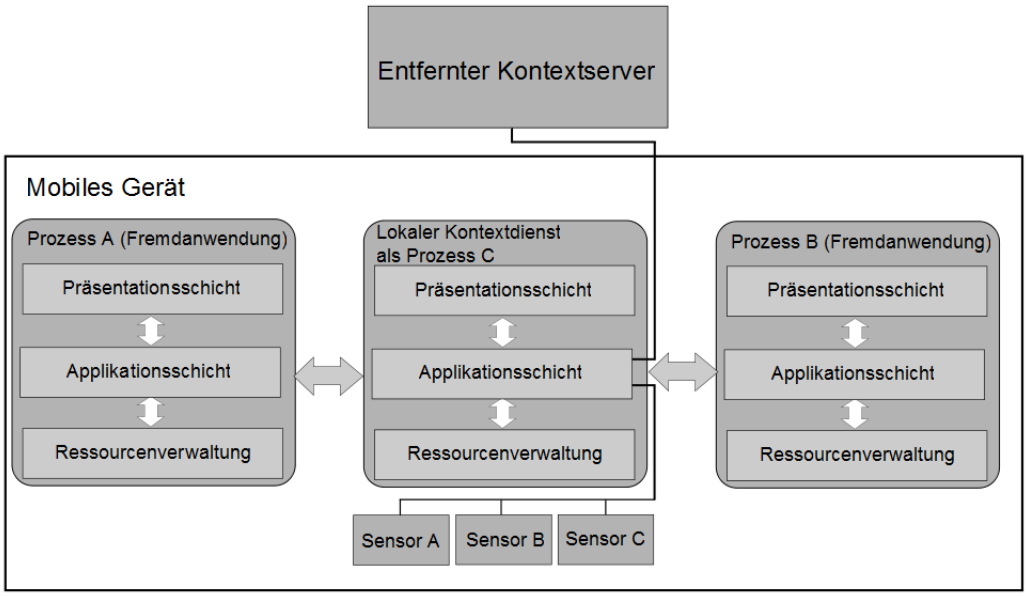


Abbildung 6.3: Abbildung des lokalen Kontextdienstes als unabhängige Komponente (modifiziert aus [Alo03])

6.3 Abrufen der Daten

Das Abrufen der Daten hat, wie schon in Abschnitt 6.2 beschrieben, den Nachteil, dass ein erhöhter Performance- und Energiebedarf besteht. Es bleibt also durchaus fragwürdig, ununterbrochen Daten abzurufen. Unter Betrachtung der Anforderungen, dass der Nutzer benachrichtigt werden kann[4], wenn sich Kontextänderungen ergeben oder eine Anwendung gestartet werden soll, sobald ein bestimmt Kontext eintrifft, ist es allerdings notwendig, diese in regelmäßigen Abständen abzufragen. Da letztendlich der Benutzer des mobilen Gerätes, selber am besten in der Lage ist, zu entscheiden, welche Genauigkeit er benötigt, um Kontextänderungen zu erfahren oder ob er diese Funktion überhaupt verwenden möchte, soll es für ihn möglich sein, diese zu aktivieren bzw. zu deaktivieren. Des Weiteren sollte der Benutzer darüber entscheiden können, in welchen Abständen das

[4]vgl. [Mül10]

Abfragen nach neuen Kontexten stattfinden soll. Hierfür ist es also notwendig, dass das Rahmenwerk nicht nur im Hintergrund läuft und die Anwendungen mit Daten versorgt, sondern auch eine Oberfläche anbietet, damit Einstellungen durch den Benutzer geändert werden können.

6.4 Lebenszyklus des lokalen Kontextdienstes

Da der lokale Kontextdienst als eine eigene Anwendung umgesetzt wird, muss diese zu der Zeit, in der sie von Anwendungen zur Ermittlung von Kontextinformationen genutzt wird, auch als eigener Prozess laufen. Um nicht unnötige Performance zu beanspruchen, ist es von Vorteil, wenn der lokale Kontextdienst nur läuft, solange er benötigt wird, bzw. eine Anwendung auf ihn zugreift. Bei regelmäßigen Abfragen des Kontextes ist es allerdings erforderlich, dass der lokale Kontextdienst auch laufen kann, ohne dass eine Anwendung ihn verwendet. Da dies den Anforderungen der Performance und Energiesparsamkeit widerspricht, soll es auch hier, wie in Abschnitt 6.3 beschrieben, für den Nutzer möglich sein, selbst zu entscheiden, ob der Kontextdienst aktiv sein soll oder nicht. Des Weiteren muss der lokale Kontextdienst in der Lage sein, entweder nur aktiv zu sein, wenn Anwendungen auf ihn zugreifen oder dauerhaft aktiv zu sein, wenn er in regelmäßigen Abständen Kontextinformationen sammeln soll.

6.5 Systemeinstellungen im Multiprozessbetrieb

Bei einem Multiprozessbetrieb kann es dazu kommen, dass Anwendungen Einstellungen ändern, die für andere Anwendungen relevant sind. Dabei kann es zu Problemen kommen, wenn diese Anwendungen von den Einstellungen abhängig sind. Um dies zu vermeiden, muss die konkrete Einstellung und damit in Verbindung stehende Probleme betrachtet werden. Im Fall des Rahmenwerkes gibt es drei Einstellungen:

- Kontextquellen können aktiviert oder deaktiviert werden.
- Das automatische Starten und Beenden von Anwendungen kann aktiviert oder deaktiviert werden
- Die Abstände in denen Kontexte ermittelt werden, können variabel eingestellt werden

Die zwei letzteren Einstellungen sind kaum eine Gefahr, da sie hauptsächlich für den Anwender relevant sind. Bei der ersten Einstellung kann es allerdings zu Problemen kommen, wenn eine Anwendung die Attributwerte einer Kontextquelle aufrufen möchte, die

von einer anderen Anwendung deaktiviert wurde. Es muss also in diesem Fall entweder darauf hingewiesen werden, dass eine Kontextquelle möglicherweise deaktiviert ist, damit der Anwendungsentwickler bei der Entwicklung darauf reagieren kann oder eine Kontextquelle kann trotz Deaktivierung abgefragt werden, wird aber bei anderen Aktionen[5] nicht verwendet. Beide Ansätze besitzen Vor- und Nachteile. Im ersten Ansatz könnte es möglich sein, dass eine Anwendung eine Kontextquelle aktiviert, wenn diese deaktiviert ist und eine andere Anwendung das genaue Gegenteil tut. Eine der beiden Anwendungen erreicht dabei nicht das gewünschte Ziel. Im zweiten Fall könnte es sein, dass eine Anwendung eine Kontextquelle verwendet, bei der der Benutzer aber explizit nicht möchte, dass diese verwendet wird. Um dieses Problem zu vermeiden, wäre es möglich Einstellungen, die der Anwender vornimmt, eine höhere Priorität zu zuordnen. Bei neu hinzukommenden Einstellungen sollte erneut abgewogen werden, wie im einzelnen Fall verfahren wird.

6.6 Multitasking

Das Abfragen von Kontexten oder Kontextquellen kann zu Verzögerungen führen, da beispielsweise zu dem Zeitpunkt einer Abfrage, nur eine Internetverbindung mit unzureichender Bandbreite auf dem mobilen Gerät verfügbar ist. In diesen Fällen muss der lokale Kontextdienst trotzdem in der Lage sein, weitere Anfragen entgegen zu nehmen und zu verarbeiten. Ist der lokale Kontextdienst dazu nicht in der Lage, können sehr starke Verzögerungen entstehen, wobei alle weiteren anfragenden Anwendungen ausgebremst werden, wenn sie auf die Antwort des lokalen Kontextdienstes warten müssen. Um dies zu vermeiden, muss der lokale Kontextdienst Anfragen parallel bearbeiten können. Dies kann mittels Threads oder mehrerer Instanzen des lokalen Kontextdienstes ermöglicht werden. Bei einer Lösung mit mehreren Instanzen ergeben sich allerdings ähnliche Probleme, wie schon im Abschnitt 6.2 beschrieben wurde. Threads bieten die Möglichkeit einen gemeinsamen Adressraum zu benutzen, so dass jeder Thread auf gemeinsame Daten zugreifen kann. Des Weiteren sind Threads leichtgewichtiger als Prozesse und können schneller gestartet, sowie beendet werden [Tan09]. Dies bringt einen weiteren Performanzgewinn im Vergleich zu mehreren Instanzen des lokalen Kontextdienstes, die jeweils in einem eigenen Prozess laufen müssten. Ohne die Verwendung von Threads, besteht die Gefahr, bei gleich eintreffenden Anfragen oder Anfragen mit einer längeren Bearbeitungsdauer, das Rahmenwerk stark zu verlangsamen. Aus diesem Grund sollte für jeden Aufruf ein eigener Thread gestartet und die Aufgabe innerhalb des Threads bearbeitet werden.

[5] wie Beispielsweise der Ermittlung des Kontextes

6.7 Persistente Speicherung

Da der lokale Kontextdienst im Regelfall nicht dauerhaft läuft, wird er, nachdem ihn keine Anwendung mehr benötigt, beendet. Dies hat zur Folge, dass alle Daten, die er zur Laufzeit im Speicher hält, nach seiner Beendigung verloren gehen, insofern keine persistente Speicherung der Daten stattfindet. Sobald es allerdings Daten gibt, die auch beim nächsten Start des lokalen Kontextdienstes wieder vorhanden sein müssen, wird eine persistente Speicherung benötigt. Bei Betrachtung des lokalen Kontextdienstes bieten sich zum jetzigen Zeitpunkt, die Anwendungseinstellungen und die registrierten Anwendungen für den automatischen Start beim Eintreten von Kontexten als Kandidaten für die persistente Speicherung an, da alle anderen Daten nur kurzweilig benötigt werden. Somit wird gewährleistet, dass der Benutzer die Einstellung nicht bei jedem Start neu eingeben muss und auch die Anwendungen nicht jedes mal neu registriert werden müssen.

6.8 Austausch von Informationen zwischen den Anwendungen

Damit Informationen über das Rahmenwerk ausgetauscht werden können, ist es erforderlich, Informationen zu einem Kontext hinterlegen zu können. Des Weiteren muss es möglich sein, diese Informationen abzurufen und auch zu entfernen, wenn die Informationen gebraucht bzw. nicht mehr gebraucht werden. Eine entsprechende Anpassung der Schnittstelle zwischen dem lokalen Kontextdienst und den Anwendungen ist deshalb erforderlich, da dies noch nicht im Rahmenwerk vorgesehen war. Diese Thematik wird im nächsten Abschnitt 6.9 behandelt.

6.9 Schnittstellen

In diesem Abschnitt werden die Schnittstellen der einzelnen Module des gesamten Rahmenwerkes aufgeführt. Dabei wurde der Großteil aus dem bestehendem Rahmenwerk[6] wiederverwendet. Alle Änderungen werden im Folgenden aufgeführt.

6.9.1 Schnittstelle zwischen den Anwendungen und dem lokalen Kontextdienst

Die Schnittstelle bietet für die Anwendungen die Möglichkeit mit dem lokalen Kontextdienst zu kommunizieren. Dabei ist von dem Rahmenwerk schon vorgegeben, dass Kontexte abgefragt, bearbeitet und hinzugefügt werden können. Zusätzlich ist es möglich,

[6] (vgl. Felix Müller [Mül10])

Kontextquellen zu steuern und hinzuzufügen. Um den Anforderungen des Multiprozessbetriebes gerecht zu werden, wird diese Schnittstelle um die folgenden Funktionen ergänzt:

- Automatisches Starten und Beenden von Programmen
  - Programmstart für Kontext registrieren
  - Programmstart für Kontext entfernen
- Automatisches Aktualisieren von Kontextinformationen
  - Automatische Aktualisieren aktivieren
  - Automatisches Aktualisieren deaktivieren
  - Intervall für das Aktualisieren der Informationen setzen
- Austausch von Informationen zwischen den Anwendungen
  - Hinzufügen von Informationen zu einem Kontext
  - Abfragen von Informationen zu einem Kontext
  - Entfernen von Informationen zu einem Kontext

6.9.2 Schnittstelle zwischen dem lokalen Kontextdienst und den Kontextquellen

Die Schnittstelle zwischen Kontextdienst und den Kontextquellen beschreibt welche Funktionen die Kontextquellen bereitstellen müssen, um von dem Kontextdienst verwendet werden zu können. Hierbei liegt der Fokus auf dem Abfragen von Attributwerten und Attributnamen. Diese Schnittstelle wurde unverändert aus dem bestehenden Rahmenwerk verwendet, da sich unter Betrachtung einer Multiprozessumgebung keine neuen Anforderungen für diese Schnittstelle ergeben.

6.9.3 Schnittstelle zwischen dem entfernten Dienstanbieter und lokalem Kontextdienst

Über diese Schnittstelle wird die Kommunikation des lokalen Dienstanbieter mit dem entfernten Kontextdienst geregelt. Zu den Kernpunkten gehört das Bearbeiten des Kontextmodells, Bearbeiten von Kontexten und Abfragen von Informationen des Modells.[7]. Dieses Schnittstelle kann auch unter Betrachtung des Multiprozessbetriebes unverändert übernommen werden.

[7] vgl. [Mül10]

6.10 Zusammenfassung

In diesem Kapitel wurde unter Berücksichtigung aller Anforderungen, die unter der Betrachtung eines Multiprozessbetriebs ermittelt wurden, das bestehende Rahmenwerk untersucht. Im ersten Abschnitt wurden mögliche Umsetzungen des lokalen Kontextdienstes durchgespielt und eine geeignete ausgewählt. Diese unterscheidet sich nicht von dem ursprünglichen Konzept von Felix Müller[8], so dass viele Grundstrukturen, wie beispielsweise die vorhandenen Schnittstellen mit ein paar Erweiterungen, übernommen werden konnten. Danach wurden die Anforderungen auf ihre technische Realisierung, innerhalb des Rahmenwerks, geprüft. Als Ergebnisse dieses Prozesses, lassen sich nun Komponenten anhand konkreter technischer Vorgaben gezielt auswählen. Um das, in diesem Kapitel vorgestellte, theoretische Konzept zu evaluieren, ist nun die Durchführung eines Testlaufs mit multiplen Anwendungen, die das Rahmenwerk verwenden, notwendig. Um eine Testumgebung für den Testlauf schaffen zu können, bedarf es einer prototypischen Implementierung, welche in den nächsten Kapiteln besprochen wird.

[8] siehe konzeptioneller Aufbau in Abschnitt 3.2

7 Implementierung und Durchführung eines Testszenarios

In diesem Kapitel werden die Schritte zur Realisierung eines Prototyps des theoretischen Konzeptes aus Kapitel 6 durchlaufen. Hierfür wird zuerst eine Plattform und Programmiersprache für die Umsetzung ausgewählt. Danach werden einzelne Komponenten auf Basis der Plattform ausgewählt, mit denen die Umsetzung des Prototyps erfolgen soll. Darauf hin wird die Implementierung und einiger Kernfunktionen detailliert beschrieben. Das Kapitel schließt mit der Erläuterung des hier durchgeführten Testszenarios und den Erkenntnissen hieraus ab.

7.1 Plattform und Programmiersprache

Während der Konzeption haben sich bei der Betrachtung des Multiprozessbetriebes hauptsächlich Aspekte ergeben, die den lokalen Kontextdienst betreffen, weshalb dieser vorrangig in diesem Kapitel behandelt wird. Um konkrete Komponenten für den lokalen Kontextdienst auswählen zu können, ist es notwendig eine Plattform und eine Programmiersprache festzulegen. Im folgenden Abschnitt erfolgt zuerst eine Auseinandersetzung mit der Problematik der Heterogenität bei der Entwicklung auf mobilen Systemen. Danach wird eine multiprozessfähige Plattform mit einer Programmiersprache ausgewählt, auf welcher der lokale Kontextdienst entwickelt wird.

7.1.1 Problematik der unterschiedlichen Plattformen und Programmiersprachen bei mobilen Geräten

Damit der lokale Kontextdienst die zentralen Aufgaben übernehmen kann, wie die Kommunikation mit dem externen Kontextserver, Bereitstellung der Kontexte für die Anwendungen etc., ist er auf den mobilen Geräten angesiedelt. Allerdings gibt es für mobile Geräte viele voneinander unabhängige Hersteller, welche unterschiedliche Betriebssysteme verwenden, auf welchen Anwendungen laufen, geschrieben in verschiedenen Programmiersprachen. Der lokale Kontextdienst muss also auf heterogenen Plattformen lauffähig sein, damit eine Vielzahl an mobilen Geräten das Rahmenwerk nutzen kann. Dies wird

durch den modularen Aufbau des Rahmenwerkes ermöglicht. Der lokale Kontextdienst stellt dabei eine frei austauschbare Komponente in dem Gesamtsystems dar. Dies bedeutet, für jede Plattform, die mit den verfügbaren lokalen Kontextdiensten nicht kompatibel ist, muss eine neue Implementierung oder Anpassung erfolgen. Hierbei ist es von Vorteil, den Aufwand, für mögliche Anpassungen an die jeweilige Plattform, gering zu halten. Die Entwicklung von mobilen Anwendungen für mehrere Plattformen, stellt oftmals eine große Hürde dar[Peu10]. Um das theoretische Konzept zu evaluieren, genügt es die Funktionalität des lokalen Kontextdienst auf einer Plattform zu demonstrieren. Aus diesem Grund wird bei der abschließenden Auswahl von Komponenten nur eine Plattform betrachtet. Es sei hier allerdings angemerkt, dass der Bedarf nach einer Implementierung besteht, die eine hohe Portabilität bietet.

7.1.2 Abwägung von Programmiersprache und Plattform

Für die Konzeption und den resultierenden Prototypen wurde Java zusammen mit der Plattform Android ausgewählt. Für Java würde sprechen, dass kein großer Aufwand für die Einarbeitung benötigt wird, da schon Erfahrung in anderen Projekten mit Java gesammelt werden konnte. Hinzu kommt, dass Java auf einer Vielzahl an Plattformen verfügbar ist[Peu10]. Dies bringt zwar noch nicht die gewünschte Portabilität, da die verschiedenen Plattformen, die Java als Programmiersprache für Anwendungen gemein haben, untereinander nicht unbedingt kompatibel sind[Peu10]. Dafür besteht allerdings die Möglichkeit durch die Kapselung von Anwendungsmodulen, die Unterschiede der Plattformen zu verbergen. In dem Artikel von Jan Peuker[Peu10] wird diese Thematik behandelt und dies bezüglich eine Architektur beschrieben.

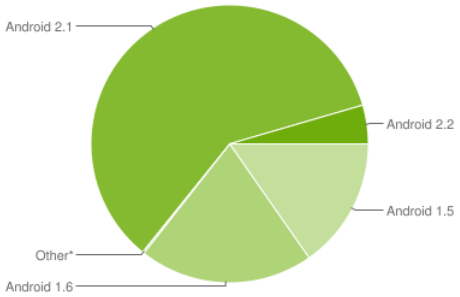
Android wurde aufgrund der Verbreitung bei vielen unterschiedlichen Herstellern ausgewählt. So benutzen Unternehmen wie Samsung, LG, Sony Erricsson, Google, Acer, HTC, General Mobile [1] Android für ihre Geräte. Im Vergleich dazu gibt es für das iPhone mit Apple nur einen einzigen Hersteller. Des Weiteren kann durch die Verwendung einer anderen Plattform und Programmiersprache als bei dem Prototyp von Felix Müller, evaluiert werden, ob das Rahmenwerk für unterschiedliche mobile Plattformen funktioniert. Allerdings ist das ausschlaggebende Kriterium das iPhone nicht zu verwenden, dass es nur einen eingeschränkte Multiprozessumgebung[2] anbietet und somit bei einer Umsetzung für den Multiprozessbetrieb ungeeignet ist. Android bietet nicht nur eine uneingeschränkte Multiprozessumgebung, sondern durch Google's Android Market auch eine Vielzahl an Anwendungen, was den potentiellen Bedarf des Rahmenwerkes steigert. Bei ande-

[1] vgl. [Gla10]
[2] Siehe hierzu [Woo10], [ho10], [der10]

ren Plattformen, wie beispielsweise JavaME fehlt ein einheitlicher *App-Store*, was dazu führt, dass Benutzer relativ selten Fremdanwendungen auf ihr mobiles Gerät runterladen [Gla10]. Android schafft es zwar nicht so viele unterschiedliche Plattformen wie JavaME zu adressieren, dafür gibt es von Android allerdings nur eine Implementierung und damit verhält es sich auf allen Geräten gleich [Gla10]. Somit ist es nicht notwendig, tiefer auf die verschiedenen Geräte einzugehen. Abschließend kann noch der für diese Arbeit positive Aspekt festgehalten werden, dass die Android-Plattform starke Ähnlichkeit zu JavaSE[3] [Gla10] hat und somit weiterer Aufwand für Einarbeitung entfällt.

7.2 Versionsunterschiede

Ein Problem bei der Entwicklung mit Android ist, dass es verschiedene Versionen gibt, wobei momentan Version 2.1 am stärksten vertreten ist, wie in Abbildung 7.1 dargestellt wird.[4]. Anwendungen die in neueren Versionen geschrieben sind, können nicht auf älteren Versionen der Plattform laufen. Die Plattformen unterscheiden sich dabei meist in Funktionalität für den Endanwender und in einigen Änderungen bzw. Optimierung der API[5]. Der Prototyp für das Rahmenwerk wurde in Version 1.5 entwickelt, somit können zum jetzigen Zeitpunkt fast alle der Geräte mit Android das Rahmenwerk verwenden. Bei den neueren Versionen der Android-Plattform wurden Optimierungen im Bereich der Performance vorgenommen, deswegen sollte bei einer Weiterentwicklung des Rahmenwerkes auf der Android-Plattform, ein Sprung auf Version 2.1 betrachtet werden. Dies gilt insbesonders, wenn der Marktanteil der Geräte mit der Android-Version 1.5 und 1.6 bis dahin weiterhin zurückgeht.



Platform	API Level	Distribution
Android 1.5	3	15.3%
Android 1.6	4	20.3%
Android 2.1	7	59.7%
Android 2.2	8	4.5%

Abbildung 7.1: Übersicht der verschiedenen Versionen der Android Plattform von [Dev10d]

[3] Java Standard Edition
[4] Der Stand dieser Information bezieht sich auf den 10.08.2010
[5] Die einzelnen Änderungen können auf der Developer-Webseite [Dev10d] eingesehen werden

7.3 Auswahl der Komponenten auf Basis eines Betriebssystems

Nachdem nun eine Plattform ausgewählt wurde, soll die Architektur des Konzeptes auf der ausgewählten Plattform umgesetzt werden. Hierfür ist es erforderlich, die vorhandenen Möglichkeiten der Android-Plattform zu betrachten. In dem folgenden Abschnitt werden deshalb Komponenten der Android-Plattform begutachtet und anschließend ausgewählt. Bei der Auswahl der Komponenten sollen die Erkenntnisse aus Kapitel 6 berücksichtigt werden.

7.3.1 Komponenten des lokalen Kontextdienstes

Der lokale Kontextdienst soll als eigenständige Komponente laufen, die zum einen im Hintergrund läuft und zum anderen auch eine Oberfläche bietet, um Einstellungen vornehmen zu können. Da es sich bei Android um ein komponentenbasiertes System handelt[6], ist es möglich, hierfür vorgefertigte Klassen zu verwenden. Diese geben einen Rahmen zur Erzeugung von Oberflächen, Hintergrundprozessen und anderen Komponenten. Zur Umsetzung des lokalen Kontextdienstes als eine eigene Anwendung, sind die Komponenten *Service* und *Activitiy* relevant. Activities besitzen eine Oberfläche mit welcher der Benutzer interagieren kann. Services hingegen sind Hintergrundanwendungen, die für den Anwendbar nicht sichtbar sind. Man unterscheidet bei den Services zwischen lokalen und entfernten Services[7]. Ein lokaler Service läuft in dem gleichen Prozess wie die Anwendung, die ihn startet[BA09]. Ein entfernter Service dagegen wird in einem eigenen Prozess ausgeführt und hat dadurch die Möglichkeit, unabhängig von der aufrufenden Anwendung bzw. Prozesses zu laufen, das heißt wird der Prozess, der den Service aufgerufen hat beendet, kann der entfernte Service trotzdem aktiv bleiben. Services und Activities erlauben es beide Threads zu verwenden. Abbildung 7.2 zeigt einen Überlbick von Activities und Services.

Im Hinblick auf die Umsetzung des Konzeptes wird eine Komponente benötigt, die von mehreren anderen Prozessen verwendet werden kann. Eine Activity bietet sich in diesem Fall nicht an, da sie bei jedem Aufruf auf der Oberfläche des mobilen Gerätes erscheint. Deswegen kommt nur die Verwendung eines Services in Frage. Hierbei ist ein lokaler Service zwar für eine einzelne aufrufende Anwendung geeignet, sobald allerdings weitere Anwendungen diesen nutzen möchten, ist der lokale Service nicht mehr ausreichend. Das Problem ist zum einen, dass der Service an die Anwendung und deren Lebensdauer ge-

[6] vgl. [BA09]
[7] In der Literatur auch oft als remote und local Service bezeichnet

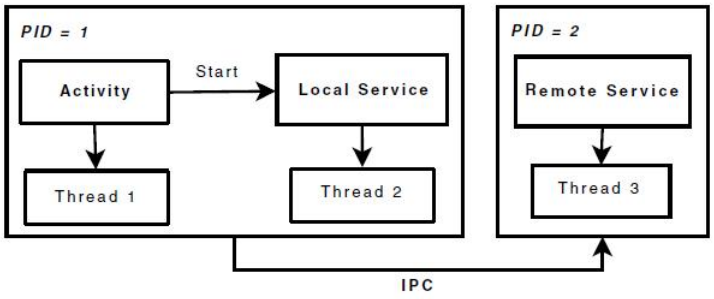


Abbildung 7.2: Übersicht von Activities und Services aus [BA09]

bunden ist und zum anderen Anwendungen als ein entfernter Prozess auf diesen Service zugreifen können müssen. Ein entfernter Service kann von mehreren Anwendungen verwendet werden und muss dabei nicht zwangsweise abhängig von der Lebensdauer anderer Anwendungen sein [BA09]. Fraglich ist nun, wie die Anwendungen mit dem entfernten Service kommunizieren können und welche Kommunikationsart sich am besten für die Umsetzung eignet. In der Abbildung 7.2 ist eine Verbindung zwischen den zwei Prozessen per IPC dargestellt, was für Inter Process Communication steht. Die Realisierung der Kommunikation zu dem entfernten Service wird im nächsten Abschnitt behandelt.

7.3.2 Kommunikation

Jede Anwendung läuft in einem eigenen Prozess, deshalb muss eine prozessübergreifende Kommunikation stattfinden, bei der ein Prozess mit dem lokalen Kontextdienst kommuniziert. Hierfür können herkömmliche Techniken wie Sockets, Shared Memory, das Linux-Dateisystem etc.[8] verwendet werden. Eine Implementierung, bei der diese Techniken verwendet werden, birgt allerdings viel Aufwand und gilt als fehleranfällig[RR09]. Beispielsweise müssten bei einer Kommunikation mehrere Schritte durchlaufen werden, wie das Einrichten des Kanals, Abfangen von Fehlern, Erstellen eines Protokolls zur Kommunikation [Alo03] u.v.m. Um sich diese Arbeit zu ersparen unterstützen Services in Android auch die Möglichkeit RPC-Aufrufe zu verwenden. RPC steht für Remote Procedure Call[9] eine Middleware, die bei Android für die Interprozesskommunikation verwendet wird[BA09] und das Aufrufen von Methoden über die Prozessgrenzen hinweg erlaubt . Dabei wird die entfernte Methode aufgerufen, als würde sie lokal verfügbar sein, der Aufruf wird aber in dem entfernten Prozess ausgeführt.
Zu den bisher hier aufgeführten Aspekten bietet RPC noch einen weiteren entscheiden Vorteil, denn RPC-Aufrufe werden über einen Thread-Pool ausgeführt [Dev10b][Dev10c],

[8] vgl. [RR09]
[9] Es wird hier das Wort Prozedur verwendet, in Java spricht man von Methoden. Im weiteren Verlauf der Arbeit wird deshalb auch das Wort Methode verwendet.

was dazu führt, dass der lokale Kontextdienst automatisch jede Anfrage von anderen Anwendungen direkt in einem eigenen Thread ausführt. Damit werden die Anforderungen aus Abschnitt 6.6 von der Middleware erfüllt und es ist nicht mehr erforderlich, diese selbst zu implementieren. Interprozesskommunikation birgt allerdings den Nachteil, dass sie aus Systemsicht sehr aufwändig ist. Android führt jeden Prozess in einer eigenen Virtual Machine[10] aus. Das führt allerdings dazu, dass Parameter serialisiert und deserialisiert werden müssen, da die Kommunikation zwischen den Prozessen außerhalb der DVM auf Betriebssystemebene abläuft[BA09]. Nichts desto trotz bietet RPC genügend Vorteile, so dass eine Verwendung gerechtfertigt ist. Wie nun die Kommunikation bei diesem Rahmenwerk im Detail abläuft, wird im nächsten Abschnitt behandelt.

7.3.3 Datenspeicherung

Für die Speicherung von persistenten Daten bieten sich auf dem Androidsystem viele Möglichkeiten an. Mit SQLite ist eine Datenbank bei Android schon von vornherein vorhanden[11]. Allerdings erscheint eine Datenbank für diese geringe Menge von Daten überdimensioniert. Bei jedem Abruf der Datenbank wäre es erforderlich, mittels SQL-Befehlen die Daten aus der Datenbank abzurufen und dann in Objekte oder Variablen zu speichern. Der umgekehrte Prozess muss vorgenommen werden, um Daten in die Datenbank zu schreiben. Zusätzlich müsste ein Datenbankschema erstellt werden. Eine weniger aufwändige Variante zur Speicherung, ist das Speichern in Dateien. Diese Variante benötigt keine SQL-Befehle, sondern ermöglicht es Daten, wie beispielsweise den Inhalt eines Arrays, direkt in eine Datei zuschreiben. Allerdings erwies sich das Lesen einer Datei bei ersten Tests als relativ aufwendig, da hier die komplette Datei, Zeichen für Zeichen, ausgelesen werden muss, um die Daten wieder in eine Java-Datenstruktur zubringen. Android bietet mit der Schnittstelle *Shared Preference* die Möglichkeit Daten als sogenanntes *Schlüssel-Wert-Paar* zu speichern. Diese werden dann in einer Datei auf dem Gerät abgelegt. Dabei ist es möglich, einen Schlüssel und den dazugehörigen Datentypen anzugeben . Dies macht die Handhabung sehr einfach, da Daten sehr schnell, ohne Umwandlungen direkt geladen und abgefragt werden können. Aus diesem Grund sollen Shared Preferences in dieser Arbeit für das persistente Speichern von Daten verwendet werden.

7.4 Prototypische Implementierung

Nachdem die Plattform und die Komponenten abgewogen und ausgewählt wurden, kann nun die Beschreibung der Implementierung durchgeführt werden. Ziel dieses Abschnitts

[10] In Android wird dies Dalvik Virtual Machine genannt, abgekürzt DVM
[11] vgl. [BA09]

ist es, die Umsetzung des lokalen Kontextdienstes vorzustellen und zu zeigen, wie die neu hinzugekommen funktionalen Anforderungen umgesetzt wurden. Darauf aufbauend kann dann der Testlauf mit mehreren Anwendungen durchgeführt werden.

7.4.1 Aufbau des lokalen Kontextdienstes auf der Android-Plattform

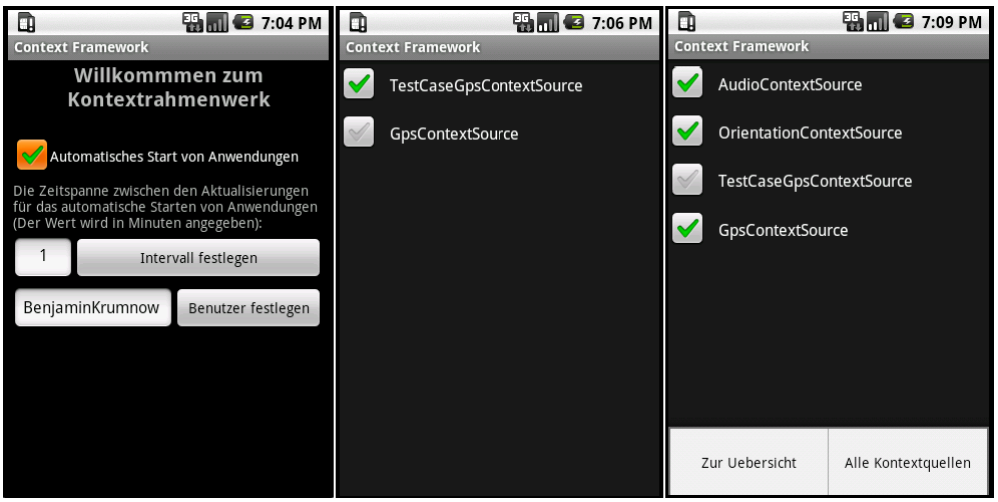


Abbildung 7.3: Oberfläche des lokalen Kontextdienstes

Für die Umsetzung des lokalen Kontextdienstes auf der Android-Plattform wurden ein entfernter Service und zwei Activities verwendet. Beide Activities dienen dem Ändern von Einstellungen. Abbildung 7.3 zeigt die Oberfläche der *ContextFrameworkActivity* (links), welche Einstellungen für das automatische Starten von Anwendungen anbietet und der *ContextSourceActivity*(mittig und rechts), welche das aktivieren und deaktivieren von Kontextquellen ermöglicht. Um zwischen den beiden Anwendungen zu navigieren, wird das Menü verwendet, welches auf der rechten Seite in Abbildung 7.3 dargestellt wurde. Des Weiteren zeigt die Abbildung die dynamische Erweiterung der Oberfläche. Werden neue Kontextquellen hinzugefügt oder entfernt, wird die Oberfläche dementsprechend erweitert.

Die Activities greifen auf den lokalen Kontextdienst, genauso wie alle anderen Anwendungen, über Schnittstellen zu und laufen in einem eigenen Prozess, damit der Service weiterhin unabhängig gestartet und beendet werden kann. Allerdings haben sie auch die Möglichkeit über die SharedPreferences des Services zuzugreifen. Bei der Implementierung wurde nur lesender Zugriff verwendet, wenn die Activities auf SharedPreferences zugreifen, die der Service benutzt. Bei Änderungen geben die Activities diese an den Service weiter, der die Änderungen vornimmt.

Der Service implementiert die eigentliche Funktionalität des Rahmenwerkes und kann

über eine Schnittstelle von Anwendungen bzw. Prozessen erreicht werden. Dabei spielt es keine Rolle, ob es sich um eine Activity oder einen Service handelt, der mit dem lokalen Kontextdienst kommuniziert. Abbildung 7.4 zeigt den Aufbau des lokalen Kontextdienstes mit den Android Komponenten.

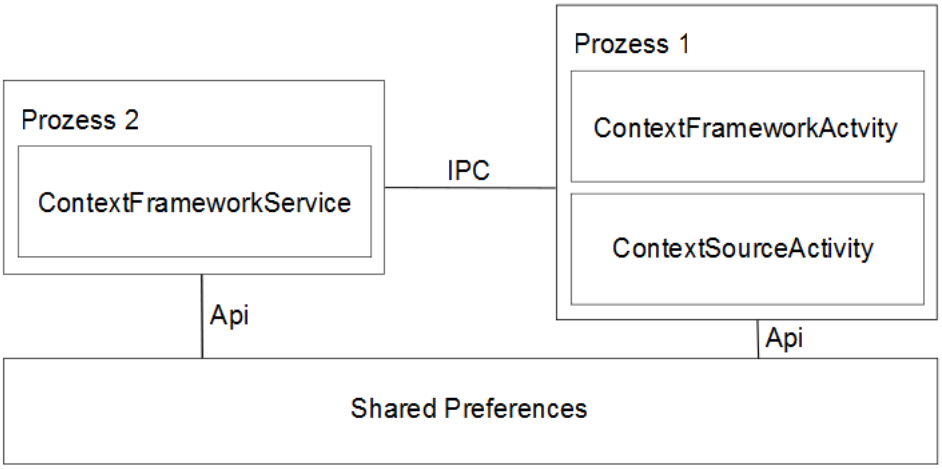


Abbildung 7.4: Übersicht der Komponenten des lokalen Kontextdienstes

7.4.2 Schnittstellen

Als nächstes werden die Schnittstellen betrachtet, die das Rahmenwerk anbietet. Da die Schnittstellen zwischen dem lokalen Kontextdienst und dem entfernten Kontextserver, sowie zwischen den Kontextquellen und den Sensoren nicht verändert wurden, sind diese hier nicht noch einmal explizit aufgeführt. Bis auf die Schnittstelle zwischen dem lokalen Kontextdienst und dem entfernten Kontextserver, wurden die Schnittstellen so angepasst, dass nun Java-Datentypen verwendet werden.

7.4.2.1 Schnittstelle zwischen dem lokalen Kontextdienst und den Anwendungen

Die Schnittstelle für die Interprozesskommunikation wurde als AIDL[12]-Datei erstellt. Die AIDL-Datei beinhaltet alle Methoden, die der lokale Kontextdienst für andere Prozesse bzw. Anwendungen anbietet. Allerdings reicht eine Datei mit der Aufführung aller Methoden nicht aus, weshalb noch drei weitere AIDL-Dateien eingeführt wurden. Um zu begründen, weshalb dieser Schritt notwendig ist, werden nun einige Android spezifische

[12]AIDL steht für Android Interface Definition Language und ist eine Form von IDL, der Interface Definition Language

Aspekte betrachtet.

RPC-Aufrufe sind von der Grundeinstellung synchron, d.h. Prozesse, die einen Aufruf starten, unterbrechen ihren weiteren Ablauf, bis die Antwort des lokalen Kontextdienstes angekommen ist. Bei Android wird ein sogenannter UI-Thread[13] verwendet, der für die Oberfläche und die Interaktion mit dem Benutzer zuständig ist. Wenn eine Aktion im UI-Thread ausgeführt wird und diese eine Berechnung ausführt die länger als vier Sekunden dauert, wird ein *ANR*[14] an den Benutzer gesendet, was dazu führt, dass die Anwendung beendet wird[BA09]. Dies kommt daher, dass Berechnungen die im UI-Thread durchgeführt werden, die Oberfläche blockieren und der Anwender in dieser Zeit somit keine Interaktionselemente der Oberfläche verwenden kann. Bei Synchronen RPC aufrufen kann es auch zu Verzögerungen kommen, beispielsweise, wenn der lokale Kontextdienst GPS-Daten ermitteln muss. Damit bei solchen Verzögerungen ein ANR vermieden werden kann, gibt es zwei Möglichkeiten. Entweder die Anwendung führt Abfragen an den lokalen Kontextdienst außerhalb des UI-Threads aus, dies kann Beispielsweise mittels eines Threads oder Service passieren, oder der lokale Kontextdienst bietet eine asynchrone Übertragung an. Hierfür ist allerdings ebenso viel Aufwand für den Entwickler der zugreifenden Anwendung notwendig, wie bei der Verwendung eines Threads. Des Weiteren werden bei asynchronen Aufrufen *Callback*-Objekte[15] benötigt, was das Interface unübersichtlich macht. Aus diesem Grund wird weiterhin eine synchrone Übertragung mit einer Ausnahme verwendet. Bei dem Push-Mode, durch den sich Prozesse benachrichtigen lassen können, ob der Kontext sich geändert hat, wurde eine asynchrone Übertragung vorgesehen, da hier der aufrufende Prozess, erst eine Rückmeldung erhalten soll, wenn es Änderungen gab und keine direkte Antwort benötigt. Durch die asynchrone Übertragung wird eine eigene AIDL-Datei benötigt.

Eine weitere Eigenschaft von Android, die für die Schnittstellen des lokalen Kontextdienst relevant ist, ist das Übertragen von komplexen Datentypen. Simple Datentypen wie Strings, Integers, Longs etc. lassen sich ohne Weiteres übertragen. Bei komplexen Datentypen wie Objekten, müssen diese Objekte das Interface *Parcelable* implementieren. Der lokale Kontextdienst überträgt Objekte der Klasse *Attribute* und *Context*, die beide das Parcelable-Interface implementieren. Dies muss auch in jeweils einer weiteren AIDL-Datei gekennzeichnet werden.
Aus diesem Grund wurden insgesamt vier AIDL-Dateien erstellt. Die Schnittstelle *IContext-Framwork* beinhaltet alle Methoden, die der lokale Kontextdienst anbietet. Die Schnitt-

[13]UI steht für User Interface
[14]ANR steht für Application Not Responding
[15]Hiermit sind Objekte gemeint, die die Rückgabe von Werten bei der asynchronen Übertragung ermöglichen

stelle *ICallBackContextFramework* beinhaltet die Schnittstelle für das Callback-Objekt. Als letztes wurde noch jeweils eine Schnittstelle für die Objekte Context und Attribute eingefügt, wobei alle hier aufgeführten Schnittstellen mit der Endung .aidl gekennzeichnet sind.
Mittels dieser AIDL-Dateien kann nun das AIDL-Tool, einen Stub jeweils für den lokalen Kontextdienst und für jede Anwendung, die mit dem Rahmenwerk kommunizieren möchte, erzeugen. Dabei müssen die AIDL-Dateien für jede Anwendung und den lokalen Kontextdienst verfügbar sein, damit ein Stub erzeugt werden kann. Die Kommunikation zwischen den Anwendungen und dem lokalen Kontextdienst, sowie der Verbindungsaufbau, werden im nächsten Abschnitt behandelt. Im Quellcodeauszug 8.1 wird ein Ausschnitt der IContextFramwork-Schnittstelle gezeigt, bei der die neu hinzugekommenen Funktionen aufgeführt sind.

7.4.3 Interprozesskommunikation

Im Folgenden wird der Kommunikationsaufbau von Anwendungen zum Framework besprochen. Dies ist notwendig, da es nur so möglich ist, die einzelnen Schritte zu begründen, die ein Anwendungsentwickler vornehmen muss, damit seine Anwendungen mit dem Rahmenwerk kommunizieren können.
Damit ein Prozess mit dem Rahmenwerk kommunizieren kann, muss der Service des lokalen Kontextdienstes gestartet werden. Dies wird über einen Intent erreicht. Intents sind Systemnachrichten, die es Komponenten in Android erlauben miteinander zu kommunizieren [BA09]. Der Intent muss dabei auf den Namen des Services und des Paketnamen verweisen, denn Android benötigt Intentfilter um einen Intent einer Komponente zuweisen zu können. Der Intentfilter wurde in der AndroidManifest-Datei des lokalen Kontextdiensts definiert. Des Weiteren ist es notwendig, dass der Service an die Applikationen gebunden wird. Dies erfolgt über den bindService-Befehl.
Der Auszug aus dem Quellcode 7.2 zeigt das AndroidManifest in dem Intentfilter definiert wird. Ein Aufruf zum Verbinden mit dem ContextFrameworkService ist in Quellcodeauszug 7.1 dargestellt.

Listing 7.1: Quellcode Auszug für das Verbinden mit den ContextFramework

```
1 contextFrameworkConnection = new FrameworkConnection();
2             Intent intent = new Intent();
3             intent.setClassName( "de.contextframework", "
                 de.contextframework.
                 ContextFrameworkService" );
```

```
4            bindService( intent,
                 contextFrameworkConnection, Context.
                 BIND_AUTO_CREATE);
```

Listing 7.2: Quellcode Auszug des Manifest des ContextFramework

```
1 <service android:name="de.contextframework.
     ContextFrameworkService"
2                  class="de.contextframework.
                       ContextFrameworkService"
3                  android:exported="true"
4                  android:process=":remote">
5                  <intent-filter>
6                         <action android:name="android.
                             intent.action.Main" />
7                         <action android:name="de.
                             contextframework.
                             ContextFrameworkService" />
8                  </intent-filter>
9            </service>
```

In dem Sequenzdiagramm 7.5 ist der Ablauf dargestellt, wie es zur Bindung und Datenaustausch mit dem ContextFrameworkService kommt. Die Kommunikation mittels Intent erfolgt nach dem Publish-Subscribe-Entwurfsmuster[Kro09]. Damit Komponenten eine Nachricht über das Android-Framework erhalten können, müssen sie registriert werden. Dies wird durch das Android-Framework beim Einlesen der AndroidManifest-Datei[Kro09] erledigt (1). Zur Laufzeit sendet eine Anwendung mittels dem *bindService()*-Befehl einen Intent an das Android-Framework (2). Die dabei mitgegebene Flag gibt Operationen für das Binden an. Zusätzlich wird eine Instanz der Klasse FrameworkConnection die das Interface ServiceConnection implementiert mitgegeben. Das Android-Framework ermittelt, über die im Intent angegebenen Merkmale, welche Komponente den Intent erhält. Das Android-Framework ermittelt in diesem Fall den ContextFrameworkService als Empfänger des Intents, was zur Folge hat, das der Service gestartet wird(3). Der Service gibt als Rückgabe, auf den eingehenden Intent, eine Instanz des IBinders zurück, die eine Anwendung verwenden kann, um mit dem Service zu kommunizieren [Dev10a]. Das Android-Framework startet darauf hin die onServiceConnect-Methode der ServiceConnection der aufrufenden Anwendung (4). Dabei erhält die Anwendung die Instanz des IBinders, mittels dessen entfernte Methodenaufrufe durchgeführt werden können (5), bis die unBind()-Methode aufgerufen wird (6). Je nach dem, ob noch andere Anwendungen mit dem Ser-

vice verbunden sind, läuft der Service weiter oder wird beendet.

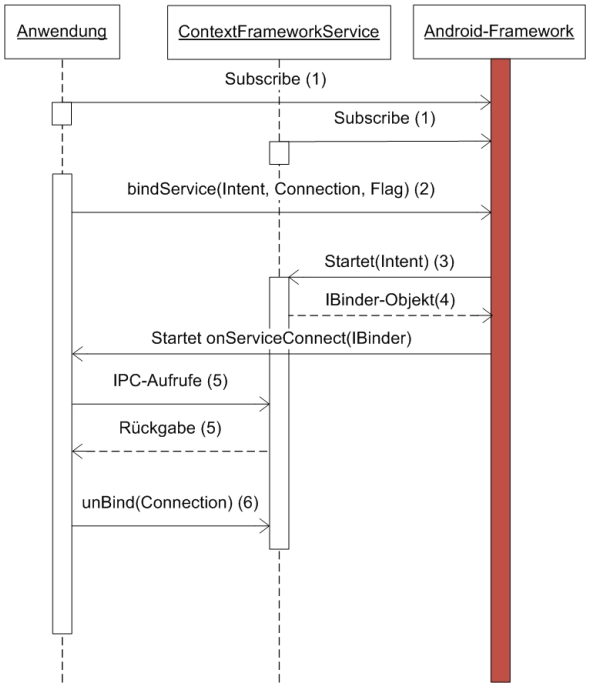


Abbildung 7.5: Verbindungsaufbau zum lokalen Kontextdienst (modifiziert aus [Kro09])

Abbildung 7.6 zeigt die Schritte eines gewöhnlichen RPC-Aufrufs. Der Client-Prozess stellt in dieser Grafik die Anwendungen und der Server-Prozess den ContextFrameworkService dar. Nachdem der ContextFrameworkService und die Anwendung miteinander verbunden sind, kommunizieren sie über die Stubs. Dabei werden die Parameter serialisiert und deserialisiert(siehe Abschnitt 7.3.2). Der Dispatcher kümmert sich hierbei um die eingehenden Verbindungen unterschiedlicher Clients.

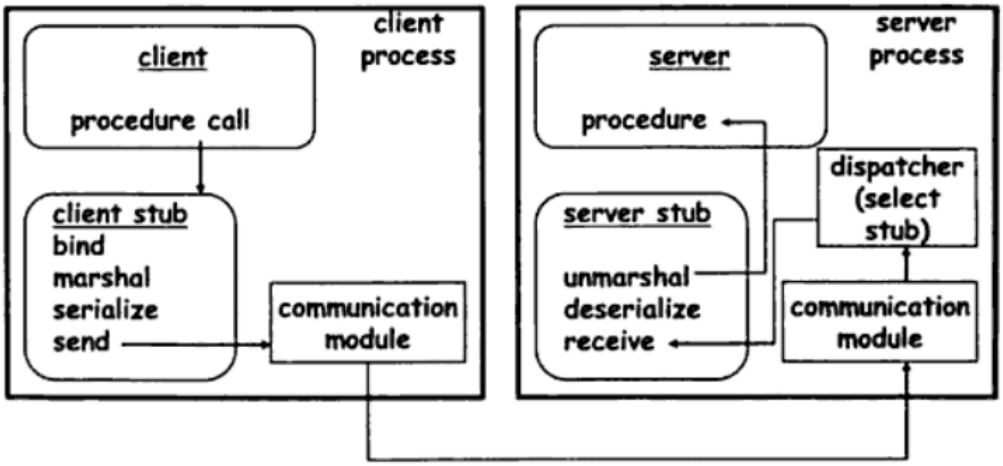


Abbildung 7.6: Grafische Darstellung eines RPC-Aufrufs

7.4.4 Klassendiagramm

In der Abbildung 7.7 ist das Klassendiagramm des lokalen Kontextdienstes dargestellt. In dem Paket *de.contextframwork* befindet sich der Service, die Activities und die Schnittstellen zur Kommunikation. Im Paket *de.contextframework.data* wurden die Datenstrukturen und die zwei Schnittstellen zur Übertragung von Objekten, die das Interface Parcelable implementieren, angelegt. Das Paket *de.contextframework.helpers* beinhaltet Hilfsklassen für den ContextFrameworkService. Die Kontextquellen wurde im Paket *de.contextframework.contextsoures* modelliert, wobei das Prinzip von [Mül10] übernommen wurde.

7.5 Beschreibung der neuen Funktionalitäten

Nachdem der Aufbau und die Kommunikation des lokalen Kontextdienstes vorgestellt wurden, folgt nun eine Beschreibung einiger neu hinzugefügten Funktionen.

7.5.1 Vermeidung von redundanten Abrufen

Jeder Aufruf des ContextFrameworkService hat zur Folge, dass ein neuer Thread gestartet wird, in der Aufruf durchgeführt wird. Damit zwei Threads nicht gleichzeitig versuchen Kontexte oder Kontextquellen abzufragen, wurde eine entsprechende Funktion eingebaut. Abbildung 7.8 zeigt einen typischen Ablauf, bei zwei gleichzeitig abrufenden Threads, dabei wurde das Entwurfsmuster des Observer verwendet[16]. Bei diesem Ablauf fragt Thread 1 an der *RequestBoardCollection*, ob die bevorstehende Anfrage schon existiert. Bei die-

[16] siehe [Ull09]

ser Abfrage befindet sich der Thread in einer kritischen Region, da alle Threads die eine Abfrage durchführen, die gemeinsame Ressource, die RequestBoardCollection nutzen. Deshalb ist es notwendig dass sich Thread 1 zu diesem Zeitpunkt in einem *Synchronized-Block* befindet, um zu verhindern, dass andere Threads gleichzeitig auf die gemeinsame Ressource zugreifen. Die RequestBoardCollection speichert alle Anfragen, die zum aktuellen Zeitpunkt durchgeführt werden, als Objekte vom Typ *RequestBoard*[17]. Da die Anfrage noch nicht existiert, erhält der Thread ein *False* als Rückgabe. Thread 1 erstellt daraufhin ein neues RequestBoard und fügt es der RequestBoardCollection hinzu. Danach verlässt Thread 1 den Synchronized-Block und fragt alle Attribute einer Kontextquelle ab. Kurz darauf wird Thread 2 gestartet, der ebenfalls nach der gleichen Anfrage bei der RequestBoardCollection fragt und sich dabei in einem Synchronized-Block befindet. Der Thread erhält ein *True* als Antwort, woraufhin er das RequestBoard bei der RequestBoardCollection anfordert. An dem RequestBoard registriert der Thread ein neues Objekt vom Typen RequestObserver, welches für ihn ab sofort das RequestBoard beobachtet und den Thread benachrichtigt, wenn es Änderungen an dem RequestBoard gibt. Thread 2 verlässt danach seinen Synchronized-Block und geht in den wartenden Zustand. Sobald Thread 1 alle Attribute der Kontextquelle erhalten hat, gibt dieser seine Ergebnisse an das RequestBoard weiter. Das RequestBoard benachrichtigt alle RequestObserver, die sich bis zu diesem Zeitpunkt beim Requestboard registriert haben. Die RequestObserver wecken wiederum den entsprechenden Thread auf. In diesem Fall wird nur Thread 2 benachrichtigt. Thread 1 entfernt das RequestBoard von der RequestBoardCollection und verlässt danach seinen Synchronized-Block. Thread 1 arbeitet den Rest seiner Aufgaben ab, gibt seine Daten an den aufrufenden Prozess zurück und wird beendet. Thread 2 ruft derweil alle Attribute über den RequestObserver ab und erledigt danach die noch verbleibenden Arbeitsschritte.

[17] RequestBoards erben von der Klasse *Observable* und können somit beobachtenden Objekten mitteilen, wenn Änderungen an ihnen durchgeführt wurden

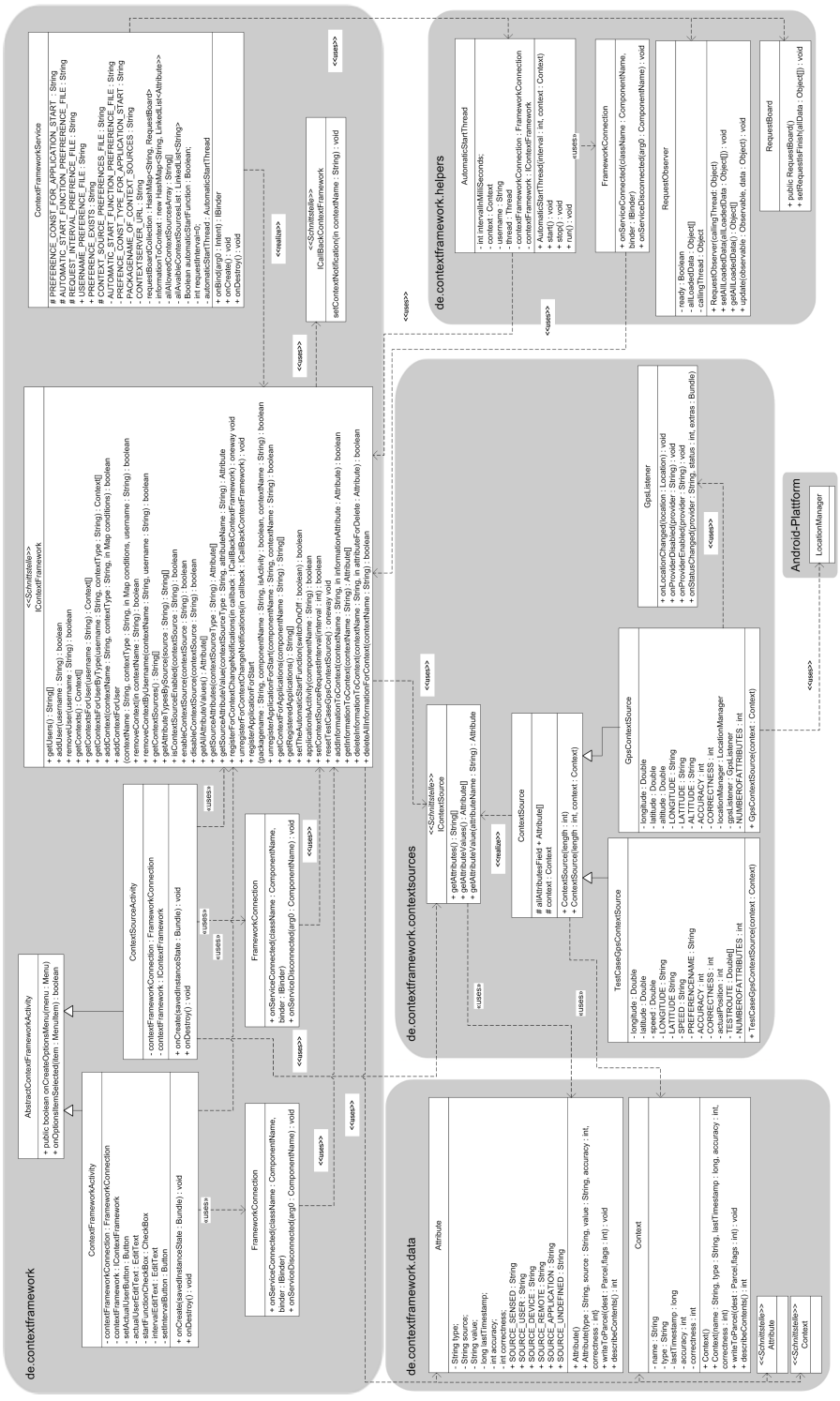


Abbildung 7.7: Klassendiagramm des lokalen Kontextdiensts

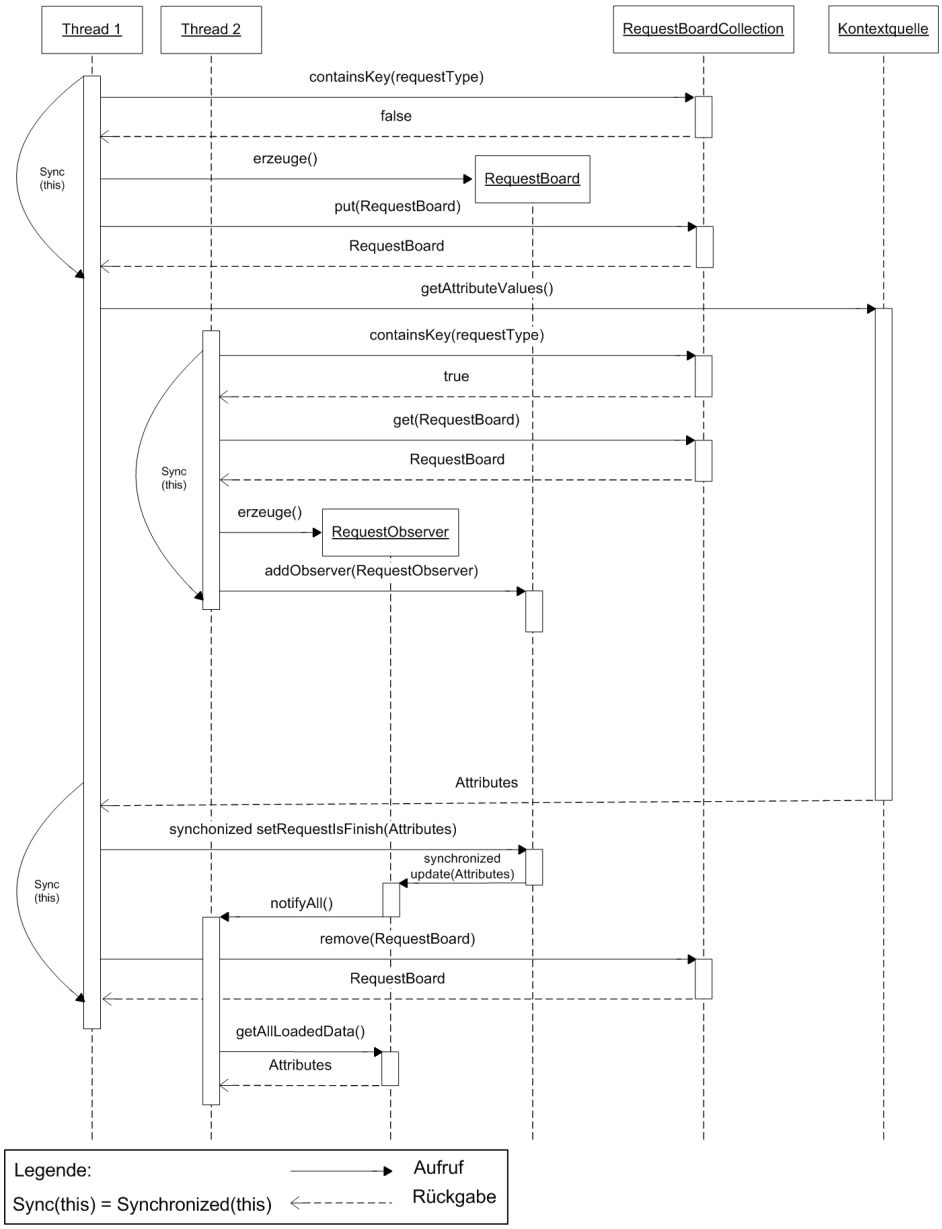


Abbildung 7.8: Grafische Darstellung für die Vermeidung von redundanten Abfragen

7.5.2 Automatischer Start von Anwendungen

Das Registrieren von Anwendungen wurde über die hinzugefügten Schnittstellen realisiert. Problematisch ist hierbei, dass der lokale Kontextdienst nur läuft, wenn Anwendungen eine bestehende Verbindung zu ihm haben. Deshalb wurde die Funktionalität für das automatische Starten in einem eigenen Thread umgesetzt. Dies hat den Vorteil, dass der lokale Kontextdienst aktiv bleibt, obwohl keine Anwendungen mehr an ihn gebunden sind, bis der Thread beendet wird. Der Ablauf innerhalb des Threads ist dabei immer gleich. Zuerst werden alle aktuellen Kontexte eines Benutzers ermittelt. Danach werden alle Anwendungen, die für einen der bestehenden Kontexte registriert sind, ermittelt und gestartet.

7.6 Testlauf

Nachdem die technische Umsetzung einiger Komponenten besprochen wurde, kann nun der konkrete Testlauf erfolgen. Ziel dieses Testlaufes ist es zu zeigen, dass mehrere Anwendungen auf einem Gerät, zum selben Zeitpunkt, den lokalen Kontextdienst verwenden und dabei von den Vorteilen der neu hinzugefügten Funktionen profitieren können. Im nächsten Abschnitt wird zuerst auf das Szenario eingegangen, welches für den Testlauf verwendet wird. Danach werden die Anwendungen, die auf das Rahmenwerk zugreifen und die Umsetzung des entfernten Kontextserver beschrieben. Im Anschluss daran wird der Ablauf des Szenarios zusammen mit den Anwendungen auf einer technischen Ebene besprochen. Die gewonnen Erkenntnisse des Testlaufs werden im Fazit erläutert, bevor das Kapitel mit der Zusammenfassung abschließt.

7.6.1 Szenario für den Testlauf

In Abschnitt 4.3 wurde das Szenario von Marco und Sven vorgestellt. In diesem Szenario benutzen sie eine Navigationssoftware auf einem mobilen Gerät, während sie zum gleichen Zeitpunkt die Anwendung Datefinder benutzen, um neue Kontakte zu erhalten. In der Analyse des Szenarios, stellte sich heraus, dass anwendungsübergreifende Informationen der Datefinder-Anwendung zu einem umfangreicheren Suchergebnis verholfen hätten. Dies soll als der Ausgangspunkt für den Testlauf verwendet werden. Zur Simulation dieses Testlaufes werden also die folgenden Anwendungen benötigt:

- Ein Anwendung, die Umgebungssuchen durchführt
  - Die Anwendung muss in der Lage sein, den Suchradius durch andere Koordinaten als den Standort zu erweitern.
- Eine Anwendung, die ein Navigationssystem simuliert

  - Die Anwendung muss im Hintergrund laufen können
  - Die Anwendung muss über das Rahmwerk den Zielort kenntlich machen

- Das Kontextrahmenwerk, zum Ermitteln der Informationen und zum Austausch anwendungsübergreifender Informationen

7.6.2 Die Anwendungen

7.6.2.1 ContextDatefinder

Der ContextDatefinder sucht anhand von GPS-Daten nach in der Umgebung liegenden Personen. Dabei ist es möglich, eine Suche durchzuführen, die sich auf den aktuellen Standort bezieht. Zusätzlich können bei der Suche Kontextinformationen berücksichtigt werden. Dadurch sucht die Anwendung mittels Kontextinformationen auch an anderen Standorten, wie in diesem Beispiel an dem Zielort, sollte der Benutzer sich unterwegs befinden. Da es nur eingeschränkt möglich ist, Daten von Personen für eine größere Fläche zuerstellen, beschränkt sich dieses Beispiel auf eine festgelegte Route. Zum simulieren der Route wurde eine weitere Kontextquelle hinzugefügt und auf dieser Strecke weitere Kontexte angelegt, die in Abschnitt 7.6.4 beschrieben sind.

7.6.2.2 ContextNavi

Die Navigationssoftware wurde unter dem Namen ContextNavi mittels einer Activity und eines entfernten Service umgesetzt. Die Activity[18] bietet dabei die Möglichkeit der Angabe eines Start- und Zielortes. Des Weiteren kann die Navigation gestartet, gestoppt und die Kontextquelle zurückgesetzt werden. Beim Starten der Navigation wird der entfernte Service verwendet, der Positionsdaten des ContextFrameworkService abruft. Somit ist es möglich die Anwendung zu schließen und weiterhin Informationen über das Rahmenwerk abzurufen. Zusätzlich erweitert der Service einen Kontext um die Start- und Zielkoordinaten.

7.6.3 Entfernter Kontextserver

Der entfernte Kontextserver wurde bis zum jetzigen Zeitpunkt bei der Konzeption nicht verändert, da sich die Aspekte des Multiprozessbetriebes auf den lokalen Kontextdienst beziehen. Allerdings wird er für den Testfall benötigt, da er Kontexte an den lokalen Kontextdienst weitergeben muss. Aus diesem Grund wurde die Implementierung des entfernten Kontextserver von Felix Müller[Mül10] übernommen. Der entfernte Kontextdienst wurde unter der URL `http://contextserverbkrumnow.heroku.com` veröffentlicht.

[18] siehe Grafik

Die Schnittstellenbeschreibung und Beispielaufrufe können aus der Arbeit von [Mül10] entnommen werden.

7.6.4 Kontextquelle und Ontologie

Um die vorgebene Route zu simulieren wurde die Kontextquelle TestCaseGpsContextSource implementiert. Sie gibt Koordinaten für die bestehende Route an. Die Route umfasst den Weg von der Fachhochschule Köln bis zum Rudolfplatz(siehe Abbildung 7.10). Des Weiteren gibt die Kontextquelle eine konstante Geschwindigkeit von 5 Km/h an.
Zusätzlich wurde die Ontologie für diesen Testfall angepasst. Es wurde der Benutzer *BenjaminKrumnow* und die Kontexte *Unterwegs*, *Fachhochschule Köln* und *Arbeit* für diesen Nutzer hinzugefügt. Der Kontext *Unterwegs* wird aktiv, sobald eine Geschwindigkeit über 1 Km/H erreicht wird. Die Kontexte Arbeit und Fachhochschule werden beim erreichen der Standorte B und A aktiv. Eine Übersicht der hinzugefügten Kontexte ist in Abschnitt 8.3 hinterlegt.
Für das beschriebene Szenario ist zwar nur der Kontext Unterwegs interessant, allerdings wurde noch eine weitere Applikation entwickelt, die es erlaubt die automatische Startfunktion von Anwendungen zu testen. Hierfür werden die zwei anderen Kontexte benötigt. Mit der Anwendung AutomaticStartApp kann eine Activity und ein Service für den Kontext Arbeit registriert werden, so dass diese beim Erreichen des Zielortes gestartet werden.

7.6.5 Ablauf des konkreten Testlaufs

Der konkrete Ablauf des Testfalls wird im Folgenden beschrieben. Die Activity des ContextNavi wird gestartet und ruft den entfernten Service auf. Dieser greift auf den lokalen Kontextdienst zu. Dabei erweitert er den Kontext Unterwegs um die Attribute *departureLongitude*, *departureLatitude*, *destinationLongitude* und *destinationLatitude*. Danach fragt er regelmäßig Ortsinformationen ab. Danach wird die Activity des ContextNavi geschlossen und der ContextDatefinder geöffnet. Dieser ruft über das Rahmenwerk Ortsinformationen ab und ermittelt entsprechende Kontakte in der Umgebung. Beim Suchen mittels des Buttons für die Kontextsuche, ermittelt der ContextDatefinder alle aktuellen Kontexte. In den Kontexten sucht der ContextDatefinder nach dem Kontext Unterwegs. Wird der Kontext gefunden, versucht der ContextDatefinder weitere Informationen zu dem Kontext abzurufen. Da diese von dem ContextNavi hinterlegt wurden, erhält er die Attribute für die Start- und Zielposition. Der ContextDatefinder kann nun mit den Koordinaten an der aktuellen Position und der Zielposition suchen. Das Ergebnis der Suche ist in Abbildung 7.11 dargestellt.

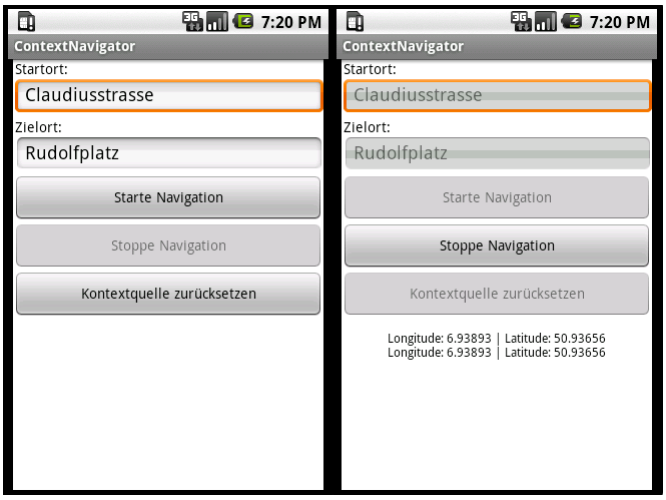


Abbildung 7.9: Oberfläche des ContextNavi

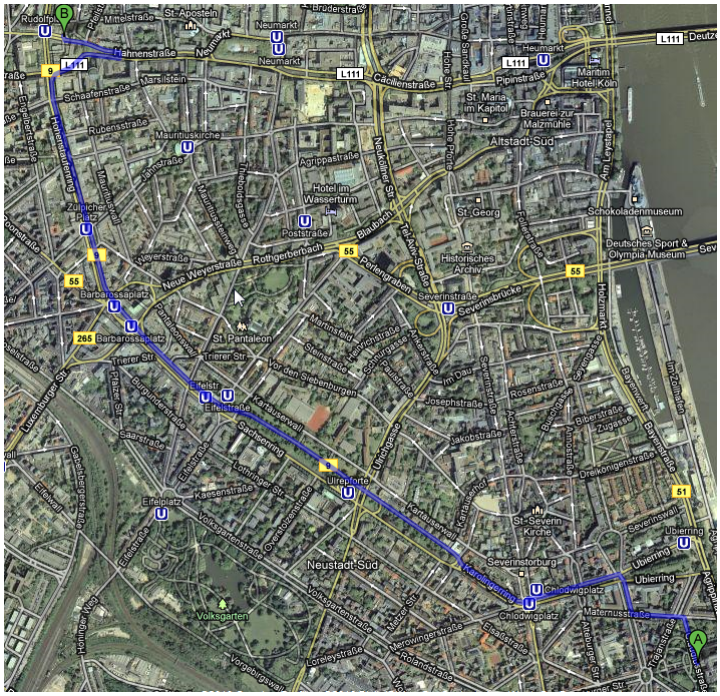


Abbildung 7.10: Die verwendete Route für das Testszenario von [Map10]

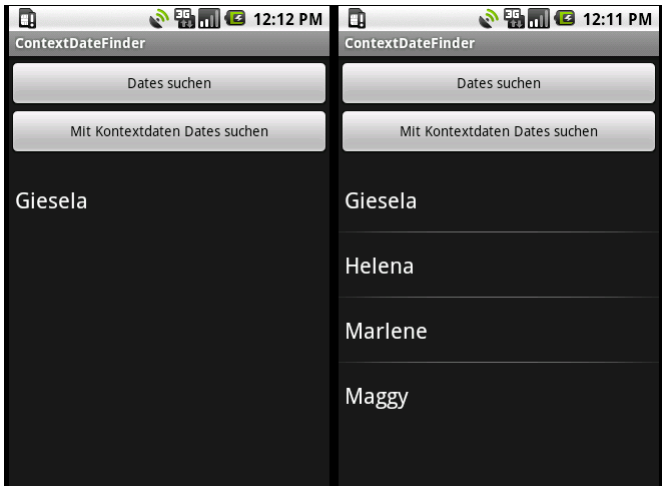


Abbildung 7.11: Oberfläche des Datefinders

7.6.6 Fazit

Erste Tests mit der automatischen Startfunktion lassen erahnen, welche Möglichkeiten sich durch diese Funktion erschließen. Es zeigt sich allerdings auch, dass bei übermäßigen Gebrauch und falscher Anwendung, diese Funktion von dem Benutzer als störend empfunden werden kann. Aus diesem Grund sollte die Funktion mit Bedacht verwendet werden.
Die Funktion zum Austauschen von Daten wurde hier anhand eines Beispieles demonstriert. Die Anwendungen, die diese Funktion verwenden wurden allerdings von demselben Entwickler entworfen. Im Realfall, sind die Anwendungen von vielen unterschiedlichen Entwicklern, welche nicht genau wissen können, wie ein Entwickler Attribute benennt, die er verwendet, um einen Kontext anzureichern. Hier für ist es notwendig ein Konzept zu entwerfen, damit Entwickler unabhängig von einander, Anwendungen entwickeln können, die Daten austauschen. Hinzu kommt, dass nicht immer die Motivation für einen Entwickler vorhanden ist, Informationen zu Kontexten bereitzustellen.

7.7 Zusammenfassung

In diesem Kapitel wurde im ersten Schritt eine Plattform für die Implementierung des lokalen Kontextdienstes ausgewählt. Danach erfolgte eine Abwägung wichtiger Bestandsteile des Rahmenwerkes. Dabei wurden spezifische Komponenten der Android-Plattform verwendet. Im Anschluss dessen wurden die technischen Umsetzungen der neuen Funktionen näher erläutert. Abschließend wurde die Implementierung durch einen Testlauf demonstriert. Dadurch wurde das theoretische Konzept evaluiert und es konnten Erkenntnisse zu den neuen Funktionen gesammelt werden. Im folgenden Kapitel können nun die Erkenntnisse dieser Arbeit zusammengefasst werden und ein Ausblick auf weitere, noch umzusetzende Aspekte geben werden.

8 Zusammenfassung, Ergebnisse und Ausblick

In dieser Arbeit wurde die Fragestellung behandelt, wie sich das kontextsensitive Rahmenwerk von [Mül10] für gleichzeitig laufende Anwendungen, auf mobilen Geräten mit multitaskfähigen Betriebssystemen umsetzen lässt. Zum Abschluss dieser Bachelorarbeit werden der Arbeitsprozess und dessen Ergebnisse zusammengefasst. Danach folgt ein Ausblick auf weiterführende Schritte und mögliche aufbauende Arbeiten.

8.1 Zusammenfassung

Um sich der Thematik dieser Arbeit zu näheren, wurden im ersten Schritt einige Grundlagen besprochen und relevante Begrifflichkeiten der Domäne eingegrenzt. Danach folgte eine Betrachtung des bestehenden Rahmenwerks, dessen Aufbau und Ansatzpunkte für diese Arbeit. Nach der Auseinandersetzung mit den einleitenden Themen, wurden über Szenarien Aspekte des Multiprozessbetriebes im Zusammenhang mit kontextsensitiven, mobilen Anwendungen gesammelt. Aus diesen Aspekten wurden die Anforderungen extrahiert, welche das Fundament für die darauffolgende Konzeption stellte. Bei der Konzeption wurden die Anforderungen und unterschiedliche Umsetzungen auf eine technische Realisierung hingehend überprüft. Für das daraus entstandene, präskriptive, konzeptionelle Modell, wurden eine multiprozessfähige Plattform und damit verbundene Komponenten zur Umsetzung ausgewählt. Anhand der ausgewählten Komponenten wurde der lokale Kontextdienst prototypisch implementiert. Ziel der Implementierung war es, das in dieser Arbeit entstandene Modell, auf die Zielerreichung hin zu evaluieren. Dabei wurde ein konkreter Testlauf simuliert, für den drei weitere Anwendungen entwickelt wurden. Abschließend wurden das implementierte System, sowie der Testlauf und die daraus gewonnen Erkenntnisse in dieser Arbeit dokumentiert.

8.2 Ergebnisse

In dem folgenden Abschnitt werden die Ergebnisse, die während der einzelnen Arbeitsschritte erzielt werden konnten, zusammengefasst.

Die Auseinandersetzung mit der in dieser Arbeit behandelten Thematik, führte zu der Konfrontation mit der Frage, ob ein Multiprozessbetrieb für kontextsensitive Anwendungen im mobilen Kontext benötigt wird. Des Weiteren waren mögliche Probleme, die die Einführung des Multiprozessbetriebes auf dem mobilen Gerät, in das bestehende System, noch ungeklärt. Um diese offenen Punkte zu klären, wurden mittels der, in den Grundlagen vorgestellten Kategorien, kontextsensitive Anwendungen ausgewählt, mit denen Szenarien erstellt wurden. Dabei ergaben sich in den Szenarien Situationen, in denen ein Bedarf für die Benutzung von mehreren kontextsensitiven Anwendungen, zum gleichen Zeitpunkt, existiert. Zusätzlich wurden Probleme aufgezeigt, etwa fehlende Automatismen und Informationsaustausch zwischen den Anwendungen. Dadurch wurde klargestellt, dass eine weiterführende Behandlung dieses Themas erforderlich ist, da noch einige ausstehende Probleme zu lösen sind.

Die Analyse der aus den Szenarien gewonnen Aspekte, brachte viele unterschiedliche Anforderungen hervor, wovon nur die Teilmenge, die für die Thematik dieser Arbeit als relevant erschien, behandelt wurde. Dabei zeigte sich, dass einige Bereiche weiterführend noch behandelt werden sollten, damit das Rahmenwerk noch mehr an Attraktivität und Effizienz gewinnt.
Bei der Konzeption konnten durch die Betrachtung der Anforderungen auf technischer Ebene, weitere tiefergehende Aspekte zur Konzeption gesammelt werden. Dazu gehört der Bedarf einer Oberfläche, zum Ändern der Einstellungen für den Benutzer. Zusätzlich wurde im Zuge der Suche eines geeigneten Aufbaus des lokalen Kontextdienstes, das grundlegende Konzept des Rahmenwerkes, nämlich die Umsetzung des lokalen Kontextdienstes als eigenständige Komponente, als die passende Variante, für die Erweiterung des Rahmenwerkes um den Multiprozessbetrieb, ausgewählt.

Anschließend konnten durch Abwägungen Komponenten für die Umsetzung des Prototyps auf der Android-Plattform ausgewählt werden. Hierbei wurde ein entfernter Service als Basiskomponente für den lokalen Kontextdienst, RPC für die Interprozesskommunikation und SharedPreferences für die Speicherung von Einstellungen, als passende Lösung für die Umsetzung auf der Android-Plattform gewählt. Durch den abschließenden Testlauf mit den Komponenten, wurden neue Funktionen demonstriert und das theoretische Konzept, sowie die Wahl der Komponenten, evaluiert. Das Ergebnis dieses Prozesses zeigt, dass es mit dem jetzigen Stand des Rahmenwerkes durchaus möglich ist, mehrere Anwendungen, die alle zeitgleich das Rahmenwerk benutzen, auf einem Gerät zu betreiben. Dabei ist es möglich, dass Anwendungen von den neuen Möglichkeiten des Rahmenwerkes durch den Multiprozessbetrieb profitieren und somit durch die neuen Funktionalitäten auch ein Mehrwert für das Rahmenwerk entstanden ist. Des Weiteren konnte festgestellt

werden, dass auch unterschiedliche mobile Geräte, die nicht die selbe Plattform verwenden, in das Rahmenwerk integriert werden können, ohne dass Änderungen an dem entfernten Server vorgenommen werden müssen. Allerdings wurde auch aufgezeigt, dass die neuen Funktionalitäten noch ausgebaut werden können, um die Entwicklung für voneinander unabhängige Entwickler zu vereinfachen. Das Ziel dieser Arbeit, die Klärung der hier behandelten Fragestellung durch eine Umsetzung des Multiprozessbetriebes für das Rahmenwerk von Felix Müller, konnte erreicht werden.

8.3 Ausblick

In diesem Abschnitt folgt ein Ausblick auf noch nicht behandelte Themenbereiche, die sich in dem Kontext dieser Arbeit anbieten. Des Weiteren werden nicht umgesetzte Funktionen angeben.
Bei der Implementierung handelt es sich um einen Prototypen, weshalb das Rahmenwerk nicht in seiner Gesamtheit umgesetzt wurde. Bei diesem Prozess wurde der Kontextserver von [Mül10] verwendet, was zur Folge hatte, dass die dort nicht implementierten Funktionaltiäten des Kontextservers, auch in dieser Arbeit nicht umgesetzt wurden. Hierzu gehört der schreibende Zugriff auf den Kontextserver und die Verwaltung von Benutzern. Des Weiteren wurde kein Gebrauch des *Reasonings* oder von Qualitätsangaben hinsichtlich der Kontextinformationen gemacht. Auf dem mobilen Kontextdienst wurde die Funktion zur Benachrichtigung von Anwendungen bei Kontextänderungen nicht umgesetzt. Um das Rahmenwerk zu vervollständigen, ist es notwendig, diese Funktionen umzusetzen.
Der durchgeführte Testlauf, führte zu der Erkenntnis, dass sich bei voneinander unabhängigen Entwicklern, ein Austausch von Informationen zu einem Kontext als schwierig erweist. Hierfür sollten geeignete Konzepte entworfen werden, um den Zugriff auf Informationen, die einem Kontext hinzugefügt werden können, zu vereinfachen. Das Konzept sollte zudem beinhalten, inwiefern ein geräteübergreifender Informationsaustausch sinnvoll ist und wie sich Entwickler dazu motivieren lassen können, Kontexte mit den Informationen ihrer Anwendung zu erweitern. Die Verwendung der automatischen Startfunktion für Anwendungen kann bei unbedachter Verwendung ein Problem sein, da es zur Ablehnung von Seiten des Benutzers führen kann. Weitere Konzepte zur genaueren Steuerung dieser Startfunktion, können hier Abhilfe verschaffen.
Weiterhin fehlen noch die Behandlung der Themenbereiche Sicherheit und Datenschutz, sowie Caching-Verfahren [Mül10]. Bei der Implementierung wurden redundante Zugriffe auf die Kontextquellen und den entfernten Kontextserver schon reduziert. Dies wurde allerdings nur für gleichzeitige Zugriffe vorgenommen und könnte mittels der Caching-Verfahren auch für aufeinander folgende Zugriffe eingeführt werden.

Durch den hier und aus der Arbeit von [Mül10] entstandenen Prototypen ist es möglich mit dem Rahmenwerk die Plattformen Android und iPhone zu erreichen. Allerdings stellen diese beiden Plattformen nur einen Teil des Marktes, weshalb weitere Plattformen erschlossen werden sollten, um die Verbreitung des Rahmenwerkes zu steigern. Eine Möglichkeit dabei ist es, die in dieser Arbeit entstandene Implementierung dahingehend zu modifizieren, dass sie auf anderen Plattformen mit Java lauffähig ist.
Das Ableiten von neuen Informationen, aus bereits vergangenen Informationen, wurde in dieser Arbeit nicht behandelt und bietet Raum für eine Annährung an die Thematik durch weitere Untersuchungen.
Die Analyse der Szenarien zeigte, dass einige Anforderungen noch unentdeckt waren. Allerdings wurde mit den Szenarien nur eine Methodik verwendet, um Aspekte des Multiprozessbetriebes zu ermitteln. Weitere Untersuchungen durch die Anwendung von Modellen der Mensch-Computer-Interaktion können zur Aufdeckung neuer Aspekte und Anforderungen führen.

Literaturverzeichnis

[Alo03] ALONSO, GUSTAVO ET AL.: *Web Services - Concepts, Architectures and Applications*. Springer Verlag, Berlin, 1. Auflage, 2003. ISBN 3-540-44008-9.

[BA09] BECKER A., PANT M.: *Android - Grundlagen und Programmierung*. dpunkt.verlag, Heidelberg, 1. Auflage, 2009. ISBN 978-3-89864-574-4.

[BH05] BAUER H., REICHARDT T., SCHÜLE A.: *User requirements for location based services*. 2005.

[Bit09] BITCOM: *Mehr als vier Milliarden Handy-Nutzer weltweit*, 2009. `http://www.bitkom.org/de/markt_statistik/64046_60608.aspx`. Sichtung: 29.08.2010.

[BJ09] BENRA J., HALANG W.: *Software-Entwicklung für Echtzeitsysteme*. Springer Verlag, Berlin Heidelberg New York, 1. Auflage, 2009. ISBN 978-3-642-01595-3.

[Bro96] BROWN, P. J.: *The stick-e document: a framework for creating context-aware applications*. 9, 1996.

[Car99] CARROLL, J. M.: *Five Reasons for Scenario-Based Design*. 1999.

[DAK00] DEY A. K., ABOWD G. D.: *Towards a Better Understanding of Context and Context-Awareness*. 2000.

[der10] DERSTANDARD.AT: *iPhone 4G Besserer Grafikprozessor, Multitasking und mehr*, 2010. `http://derstandard.at/1262209197482/Ueberblick-iPhone-4G-Besserer-Grafikprozessor-Multitasking-und-mehr`. Sichtung: 29.08.2010.

[Dev10a] DEVELOPER, ANDROID: *Application Fundamentals*, 2010. `http://developer.android.com/guide/topics/fundamentals.html`. Sichtung: 29.08.2010.

[Dev10b] DEVELOPER, ANDROID: *Designing a Remote Interface Using AIDL*, 2010. `http://developer.android.com/guide/developing/tools/aidl.html`. Sichtung: 29.08.2010.

[Dev10c] DEVELOPER, ANDROID: *IBinder*, 2010. `http://developer.android.com/reference/android/os/IBinder.html`. Sichtung: 29.08.2010.

[Dev10d] DEVELOPER, ANDROID: *Platform Versions*, 2010. `http://developer.android.com/resources/dashboard/platform-versions.html`. Sichtung: 29.08.2010.

[DF98] DAVID F., FLACHSBART J.: *All gadget and no representation makes jack a dull environment*. 7, 1998.

[Dud07] DUDEN: *Deutsches Universalwörterbuch*. Dudenverlag, Mannheim, 6. Auflage, 2007. ISBN 3-411-05506-7.

[Dud10] DUDEN: *Sensitiv*, 2010. `http://www.duden.de/definition/sensitiv`. Sichtung: 29.08.2010.

[Gla10] GLAHN, K.: *Java Magazin - Das Jahr des Androiden*. Juli 2010.

[ho10] ONLINE HEISE: *Apple beginnt iPhone-4-Verkauf in Deutschland*, 2010. `http://www.heise.de/newsticker/meldung/Apple-beginnt-iPhone-4-Verkauf-in-Deutschland-990767.html`. Sichtung: 29.08.2010.

[HR97] HULL R., NEAVES P., BEDFORD-ROBERTS J.: *Towards Situated Computing*. 1997.

[Kro09] KROOP, S.: *Evaluation der Android anhand einer Referenzeplattform*. 2009.

[Map10] MAPS, GOOGLE, 2010. `http://maps.google.de/`. Sichtung: 29.08.2010.

[Mül10] MÜLLER, F.: *Modellierung und Repräsentation des Kontextes von mobilen Nutzungsszenarien – ein Rahmenwerk für mobile kontextsensitive Applikationen*. 2010.

[Pas98] PASCOE, J.: *Adding Generic Contextual Capabilities to Wearable Computers*. 1998.

[Peu10] PEUKER, J.: *Java Magazin - Echt Mobil*. Juli 2010.

[Pla05] PLASSMANN, G.: *Draft zum kleinen Handbuch der Mensch-Computer Interaktion*. 2005.

[RN97] RYAN N., PASCOE J., MORSE D.: *Enhanced Reality Fieldwork: the Context Aware Archaeological Assistant*. 1997.

[Rot] ROTHERMEL, K.: *Kontextbezogene Systeme - Die Welt im Computer modelliert*.

[RR09] ROGERS R., LOMBARDO J., MEDNIEKS Z. MEIKE B.: *Android - Application Development*. OReilly Media, Köln, 1. Auflage, 2009. ISBN 978-0-596-52147-9.

[SBN94a] SCHILIT B. N., BILL N., ADAMS N. WANT R.: *Context-Aware Computing Applications*. 7, 1994.

[SBN94b] SCHILIT B. N., THEIMER M.: *Dissemating active map information to mobile hosts*. 1994.

[SD98] SALBER D., DEY A. K., ABOWD G. D.: *Ubiquitous Computing: Defining an HCI Research Agenda for an Emerging Interaction Paradigm*. 1998.

[Tan09] TANENBAUM, A.: *Moderne Betriebssysteme*. Pearson Studium, München, 3. Auflage, 2009. ISBN 978-3-8273-7342-7.

[Ull09] ULLENBOOM, C.: *Java ist auch eine Insel*. Galileo Computing, Bonn, 8. Auflage, 2009. ISBN 978-3-8362-1371-4.

[Woo10] WOODS, P: *So funktioniert Multitasking auf dem iPhone*, 2010. http://www.macwelt.de/artikel/_News/372056/so_funktioniert_multitasking_auf_dem_iphone/1. Sichtung: 29.08.2010.

Anhang

Ontologie Erweiterung

KONTEXTNAME	REGEL FÜR AKTIVITÄT
Arbeit Label: Arbeit Type: #LocationContext	(belongsToUser:#BenjaminKrumnow) (#hasLongitudeMax:6.9391) (#hasLongitudeMin:6.9387) (#hasLatitudeMax:50.9367) (#hasLatitudeMin:50.9363)
Fachhochschulekoeln Label:Fachhochschule Koeln Type: #LocationContext	(belongsToUser:#BenjaminKrumnow) (#hasLongitudeMax:6.9391) (#hasLongitudeMin:6.967) (#hasLatitudeMax:50.9196) (#hasLatitudeMin:50.9192)
Unterwegs Label:Unterwegs Type: #LocationContext	(belongsToUser:#BenjaminKrumnow) (hasSpeedMin:1)

Tabelle 8.2: Hinzugefügte Kontexte

Quellcode der Schnittstelle IContextFramekwork

Listing 8.1: Quellcode Auszug der IContextFramework-Schnittstelle

```
1  /**
2     * Registriert Anwendungen für den automatischen Start
3     * beim eintreffen eines bestimmten Kontextes.
4     * Androidanwendungen, die diesen Service nutzen
5     * wollen, müssen einen Intent-Filter im Manifest.xml
6     * deklarieren und den Klassen-, sowie Paketnamen
7     * angeben.
8     * @param packagename Ein String mit dem Packetnamen
9     * mit dem Muster "paket.unterpaket.unterunterpaket"
10    * @param applicationname Ein String mit dem Namen
11    * der Komponente
12    * @param isActivity Ein Boolean mit TRUE wenn es sich
13    * um eine Activity handelt oder FALSE wenn es sich um
14    * einen Service
15    * @param contextname Ein String mit dem Namen des
16    * Kontextes, für den die Anwendung registriert werden
17    * soll
18    * @return TRUE wenn die Registierung erfolgreich war
19    * FALSE wenn die Registrierung nicht erfolgreich war,
20    * weil ein Fehler aufgetreten ist oder die Anwendung
21    * schon für den übergebenen Kontext registriert wurde
22    */
23   boolean registerApplicationForStart(String packagename
       , String componentName, boolean isActivity, String
       contextname);
24
25   /**
26    * Deregistriert eine Anwendung für den automatischen
```

```
27      * Start eines Kontextes.
28      * @param componentName Ein String, der den Namen der
29      * Anwendung enthält (ohne Paketnamen!)
30      * @param contextName Ein String, der den Namen des
31      * Kontextes entält, der für diese Anwendung entfernt
32      * werden soll
33      * @return TRUE Wenn die Anwendung erfolgreich für den
34      * Kontext deregistriert wurde.
35      * FALSE Wenn die Anwendung oder der Kontext nicht
36      * gefunden wurde
37      */
38     boolean unregisterApplicationForStart(String
          componentName, String contextName);
39
40     /**
41      * Gibt alle Kontextbezeichnungen zurück,
42      * die für eine Anwendung registiert wurden
43      * @param componentName Ein String, der den Namen der
44      * Anwendung enthält (ohne Paketnamen!)
45      * @return Alle Kontexte für diese Anwendung oder Null
           ,
46      * wenn die Anwedung nicht gefunden wurde
47      */
48     String[] getContextForApplications(String
          componentName);
49
50     /**
51       * Gibt alle Registrieren Anwendungen zurück
52       * @return Ein Stringarry mit dem Namen der Anwendung
53       * und dem Paketnamen nach dem Schema
54       * PAKETNAME.ANWENDUNGSNAME
55       * Null sollten keine Anwendungen registriert sein
56       */
57     String[] getRegisteredApplications();
58
59    /**
60      * Aktiviert oder deaktiviert das automatische Starten
61      * von Anwendungen
```

```
62      * @param switchOnOff Ein Boolean, die mit TRUE das
63      * automatische Starten aktiviert
64      * und mit FALSE deaktiviert
65      * @return TRUE, wenn der Modus erfolgreich geändert
66      * wurde. FALSE, wenn die Änderung fehlgeschlagen ist
67      */
68     boolean setTheAutomaticStartFunction(boolean
          switchOnOff);
69
70    /**
71      * Gibt an, ob es sich bei der angebenen Anwendung um
72      * eine Activity oder einen Service handelt
73      * @param componentName Ein String, der den Namen der
74      * Anwendung enthält (ohne Paketnamen!)
75      */
76     boolean applicationIsActitvity(String componentName);
77
78     /**
79       * Ändert das Intervall
80       * @param interval Intervall, in dem die Abfrage des
81       * Kontextes stattfinden soll
82       * @return TRUE if the mode change succesfully
83       * FALSE if the mode can not change or an error
84       * did orrcur
85       */
86     boolean setContextSourceRequestInterval(int interval);
87
88     /**
89       * Setz die Kontextquelle zurück, damit der Testfall
90       * von vorne beginnen kann !! ACHTUNG: Dies ist keine
91       * reguläre Funktion des lokalen Kontextdienstes, da
92       *es sich hier nur um eine Testquelle handelt.
93       */
94     oneway void resetTestCaseGpsContextSource();
95
96     /**
97       * Fügt ein Attribute zu einer Kontextquelle hinzu
98       * @param contextName Ein String, den Kontextnamen
```

```
99      * beinhaltet, für den Attribute hinzugefügt werden
100     * sollen
101     * @param informationAttribute Ein String mit dem
102     * Namen des Attributes, das hinzugefügt werden soll
103     * @return TRUE, wenn das Hinzufügen erfolgreich war
104     * FALSE, wenn das Hinzufügen nicht erfolgreich war
105     */
106   boolean addInformationToContext(String contextName, in
        Attribute informationAttribute);
107
108   /**
109     * Gibt alle Attribute zu einem angegebenen Kontext
110     * zurück
111     * @param contextName Ein String, den Kontextnamen
112     * beinhaltet, für den Attribute abgefragt werden
113     * sollen
114     * @return Ein Feld mit Attributen, die diesem
115     * Kontext zugeordnet sind
116     */
117   Attribute[] getInformationToContext(String contextName
        );
118
119   /**
120     * Löscht ein Attribute zu einem Kontext
121     * @param contextName Ein String, den Kontextnamen
122     * beinhaltet, für den Attribute gelöscht werden
123     * sollen
124     * @param attributeForDelete Ein String mit dem Namen
125     * des Attributes, das gelöscht werden soll
126     * @return TRUE, wenn das Attribute gelöscht wurde
127     * FALSE, wenn das Attribute nicht gelöscht wurde
128     */
129   boolean deleteInformationToContext(String contextName,
        in Attribute attributeForDelete);
130
131   /**
132     * Löscht alle Informationen, die zu einem Kontext
133     * angegeben wurden
```

```
134     * @param contextName Ein String, den Kontextnamen
135     * beinhaltet, für den alle informationen  gelöscht
136     * werden sollen
137     * @return TRUE, wenn alles gelöscht wurde
138     * FALSE, wenn das Löschen fehlgeschlagen ist
139     */
140   boolean deleteAllInformationForContext(String
        contextName);
```

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsgbehörde vorgelegen.

Köln, den 30. August 2010

Benjamin Krumnow