Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker(FH)
im Studiengang Allgemeine Informatik

Eine Architektur zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen auf mobilen Geräten

An architecture for deriving and implementing context-dependent security measures on mobile devices

ausgearbeitet von
Rafael Kobinski

Erster Prüfer
Prof. Dr. Stefan Karsch

Zweiter Prüfer
Prof. Dr. Kristian Fischer

vorgelegt an der
Fachhochschule Köln
Fakultät für Informatik und
Ingenieurwissenschaften

Gummersbach, im Januar 2012

Rafael Kobinski
Matrikelnummer: 11054053
Studiengang: Diplom Allgemeine Informatik

Diplomarbeitsthema:
Eine Architektur zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen auf mobilen Geräten

Eingereicht: 11. Januar 2012

Erster Prüfer:
Prof. Dr. Stefan Karsch
Fachgebiet Datensicherheit und Telekommunikationsdienste
Institut für Informatik
Fachhochschule Köln
Steinmüllerallee 1
51643 Gummersbach

Zweiter Prüfer:
Prof. Dr. Kristian Fischer
Fachgebiet Anwendungsentwicklung und Multimediasysteme
Institut für Informatik
Fachhochschule Köln
Steinmüllerallee 1
51643 Gummersbach

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 Prüfungsbehörde vorgelegen.

Gummersbach, den 11. Januar 2012

_____

Rafael Kobinski

Diplomarbeit

Eine Architektur zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen auf mobilen Geräten

von
Rafael Kobinski

Kurzfassung

Aufgrund der steigenden Nutzung mobiler Geräte und der Vielzahl persönlicher bzw. geschäftlicher Daten, die auf mobilen Geräte gespeichert und verarbeitet werden, sind mobile Geräte zu einem attraktiven Angriffsziel geworden. Ungeachtet des Schutzbedarfs hat sich die Sicherheitssoftware, wie man sie von stationären Systemen her kennt, auf mobilen Geräten bisher nicht durchsetzen können. Sicherheitsansätze stationärer Systeme können mobile Geräte zwar vor Bedrohungen schützen, jedoch sind diese Ansätze zu ressourcenintensiv für mobile Geräte, deren Rechenleistung und Akkukapazität relativ begrenzt ist.
Um den Schutzbedarf mobiler Geräte und deren Anwendungen unter Berücksichtigung der begrenzten Ressourcen zu erfüllen, wird in dieser Arbeit ein Architekturmodell konzipiert, das Anwendungen, abhängig von Bedrohungen, Sicherheitsmaßnahmen bereitstellt. Anders als beim Sicherheitsansatz stationärer Systeme werden nur die Sicherheitsmaßnahmen umgesetzt, die aufgrund der aktuellen Bedrohungen notwendig sind. Ermöglicht wird die adaptive Bereitstellung von Sicherheitsmaßnahmen durch einen in dieser Arbeit vorgestellten Ansatz, der die Ermittlung von Bedrohungen und geeigneter Maßnahmen aus Kontextinformationen und Erfahrungswerten zulässt. Zuletzt wird die Realisierbarkeit des Architekturmodells anhand einer prototypischen Implementierung nachgewiesen.

Diploma Thesis

An architecture for deriving and implementing context-dependent security measures on mobile devices

by

Rafael Kobinski

Abstract

Due to the increased use of mobile devices and due to multiplicity of private as well as business data which is stored and processed on mobile devices, the latter are becoming an attractive target for attacks. Regardless of the needs for protection, security software, as known from stationary systems, has not become accepted yet. Security approaches of stationary systems are able to protect mobile devices against threats, but these approaches are too resource-intensive for mobile devices because its computing power and battery capacity is relatively limited.

To fulfil security requirements of mobile devices and their applications, considering their limited resources, this work depicts an architecture model which provides security measures to applications, depending on threats. Unlike the stationary approach of security systems, only those security measures are executed that are necessary due to the currently existing threats. The adaptive deployment of security measures is enabled by an approach introduced in this work, that allows the identification of threats and appropriate actions based on context information and experiences. Finally, the feasibility of the architectural model is demonstrated by a prototype.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Die Nutzung mobiler Geräte hat in den letzten Jahren stetig zugenommen. Smartphones, Tablets und Netbooks erfreuen sich zunehmend wachsender Beliebtheit. Mobile Gerät verwalten Termine, Kontaktdaten, E-Mails und weitere persönliche Daten. Auch Bankgeschäfte werden bereits über mobile Geräte abgewickelt. Abhängig vom jeweiligen Nutzerverhalten werden auf mobilen Geräten gleichermaßen sensible Daten gespeichert und verarbeitet wie auf stationären Geräten. Durch die wachsende Benutzerzahl und die hohe Datendichte persönlicher Informationen, werden mobile Geräte immer attraktivere Angriffsziele, die gegen Bedrohungen geschützt werden müssen.
Durch Änderungen in der Umgebung und der Situation, in der sich ein mobiles Gerät befindet, können sich auch die Bedrohungen ändern. Mobile Geräte und deren Anwendungen haben einen konstanten Schutzbedarf, der aufgrund der wechselnden Bedrohungslage mit bisherigen Ansätzen immobiler Systeme nur unzureichend erfüllt werden kann. Bisherige Sicherheitsansätze immobiler Systeme bieten einen statischen Schutz für alle möglichen Bedrohungen, die auftreten können. Diese Ansätze sind für die Nutzung auf mobilen Geräten nur eingeschränkt geeignet, da sie das mobile Gerät mit Sicherheitsmaßnahmen vor Bedrohungen schützen, die zu diesem Zeitpunkt möglicherweise nicht vorliegen. Aufgrund der beschränkten Ressourcen mobiler Geräte (Akkukapazität, Rechenleistung) wird der Benutzer durch die konstanten Schutzmechanismen in der Regel eingeschränkt. Die Einschränkungen können sich durch eine schlechte Latenz des Geräts und eine verkürzte Akkulaufzeit bemerkbar machen. Die manuelle Festlegung auszuführender Sicherheitsmaßnahmen wäre zwar eine Lösung, jedoch ist dies aufgrund des mangelnden Wissens vieler Benutzer, eines zu großen Aufwands und üblicherweise zu großer Reaktionszeit von seiten des Benutzers, nicht praktikabel. Weder der immobile Ansatz noch der manuelle Ansatz stellen eine befriedigende Lösung für mobile Geräte dar, so dass viele Benutzer vermutlich auf Sicherheitsmaßnahmen verzichten würden, wenn sie dadurch zu große Einschränkungen hinnehmen müssten.

1.1 Zielsetzung

Angelehnt an das Konzept kontextsensitiver Anwendungen, die ihr Verhalten der Umgebung und der Situation anpassen können, sollen sich Sicherheitsmaßnahmen dynamisch an den Kontext und die daraus hervorgehenden Bedrohungen anpassen, um das Risiko der Bedrohungen zu verringern. Eine dynamische Anpassung ermöglicht es, nur die Sicherheitsmaßnahmen bereitzustellen, die aufgrund der aktuellen Bedrohungslage benötigt werden. Eine übermäßige Ressourcenbelastung aufgrund unnötiger Sicherheitsmaßnahmen kann dadurch vermieden werden. Durch einen möglichst hohen Automatisierungsgrad kann außerdem gewährleistet werden, dass der Benutzer nicht durch Abfragen behindert wird.
Das Ziel dieser Diplomarbeit besteht in der Konzeption einer Architektur, die Anwendungen abhängig von den Bedrohungen, Sicherheitsmaßnahmen bereitstellt. Anwendungen erhalten so die Möglichkeit, selbst auf Bedrohungen zu reagieren und ihre Daten entsprechend den bereitgestellten Sicherheitsmaßnahmen zu schützen, ohne sich um die Erkennung von Bedrohungen und die Ableitung benötigter Sicherheitsmaßnahmen kümmern zu müssen. Die Anbindung der Anwendungen an die Architektur soll durch ein Rahmenwerk ermöglicht werden. Anhand einer prototypischen Implementierung soll die Realisierbarkeit der Architektur überprüft werden.

1.2 Vorgehensweise

Die Auseinandersetzung mit dem Thema erfordert vorab die Klärung relevanter Grundlagen im Hinblick auf die Zielsetzung. Dazu gehören die Grundlagen kontextsensitiver System sowie relevante Begriffe der IT-Sicherheit. Aufbauend auf den Grundlagen wird ein Ansatz entwickelt, der die Ableitung von Bedrohungen aus relevanten Kontextinformationen ermöglicht und für die ermittelten Bedrohungen eine automatisierte Ableitung benötigter Sicherheitsmaßnahmen realisiert. Aufbauend auf dem Ansatz zur Ableitung kontextabhängiger Sicherheitsmaßnahmen wird ein Architekturmodell konzipiert, welches Anwendungen Sicherheitsmaßnahmen zur Verfügung stellt, die das Risiko von Bedrohungen verringern sollen. Zuletzt wird die Realisierbarkeit der Architektur anhand einer prototypischen Implementierung nachgewiesen.

1.3 Aufbau der Arbeit

In Kapitel 2 werden die Grundlagen für diese Diplomarbeit geschaffen, indem die für das Verständnis dieser Diplomarbeit wichtigen Begriffe der IT-Sicherheit sowie

Begriffe kontextsensitiver Systeme betrachtet werden. In Kapitel 3 wird die Ableitung kontextabhängiger Sicherheitsmaßnahmen thematisiert. Nachdem relevante Arten von Kontextinformationen definiert wurden, wird ein Ansatz zur Ermittlung von Bedrohungen vorgestellt. Ein weiterer Ansatz beschreibt die Ableitung notwendiger Sicherheitsmaßnahmen mithilfe von Erfahrungswerten, aus den ermittelten Bedrohungen. Nach der Betrachtung der Möglichkeiten zur Umsetzung abgeleiteter Sicherheitsmaßnahmen, endet das Kapitel mit der Analyse einer Race-Condition Problematik. In Kapitel 4 werden zunächst die Anforderungen an eine Architektur zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen zusammengetragen. Darauf aufbauend werden zwei Architekturmodelle konzipiert, die sich in der Art der Umsetzung von Sicherheitsmaßnahmen unterscheiden. Für ein ausgewähltes Architekturmodell werden die Architekturkomponenten modelliert und die Schnittstellen spezifiziert. In Kapitel 5 wird die prototypische Implementierung des Architekturmodells dokumentiert. Kapitel 6 fasst die Ergebnisse dieser Diplomarbeit zusammen und gibt einen Ausblick auf mögliche weiterführende Forschungsinteressen.

2 Grundlagen

In diesem Kapitel werden Grundlagen betrachtet, die zu einem besseren Verständnis dieser Arbeit beitragen. Die Betrachtung umfasst die in dieser Arbeit verwendeten Begriffe der IT-Sicherheit sowie Begriffe aus dem Bereich der kontextverarbeitenden Systeme.

2.1 Kontext

Kontext ermöglicht eine automatische Anpassung von Systemen an die Situationen und Bedürfnisse der Benutzer. Das Thema Kontext im Zusammenhang mit kontextverarbeitenden Systemen wurde bereits von vielen Autoren behandelt. Die daraus entstandenen, teilweise unterschiedlichen Definitionen, werden in diesem Abschnitt betrachtet.

2.1.1 Kontextbegriff

Eine allgemeine Definition des Begriffs Kontext liefert der Brockhaus [Bro92]. Er beschreibt Kontext als "Situation, in der ein Text geäußert und verstanden wird (situativer K.)". Diese allgemeine Definition ist sprachwissenschaftlicher Herkunft und eignet sich daher nicht als informatik-spezifische Definition des Begriffs Kontext. Eine der ersten informatik-spezifischen Definitionen des Kontextbegriffs, wurde von Spreitzer und Theimer [ST93] wie folgt verfasst:

> "If information is available about who and what is in the vicinity of a person, then that person's computing environment and applications can behave in a contextsensitive manner."

Die beiden Autoren definieren in ihrer Arbeit die Voraussetzungen für kontextsensitives Verhalten und damit den Begriff Kontext, als Informationen darüber, wer und was sich in der Umgebung einer Person befindet.
Schilit et al. [SAW94] definieren Kontext anhand von drei Aspekten, die abgesehen vom Standort ("where you are") den Aspekten aus [ST93] entsprechen:

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

Eine allgemeinere Definition des Kontextbegriffs, die im weiteren Verlauf dieser Arbeit verwendet wird, wurde von Dey [Dey00] verfasst:

> "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 application themselves."

Die Definition von Dey grenzt den Kontextbegriff auf Informationen ein, die die Situation einer Entität beschreiben und für die Interaktion zwischen dem Benutzer und einer Anwendung relevant sind. Auch der Benutzer und die Anwendung gehören nach der Definition von Dey zum Kontext. Relevante Informationen können nach Meinung des Autors nicht aufzählend definiert werden, da diese Definition niemals vollständig wäre.
Dey beschreibt Kontext als jegliche Art von Informationen, die die Situation einer Entität charakterisieren. Die Gesamtmenge dieser Informationen, die eine eindeutige Beschreibung der Situation ermöglichen, wird demnach Kontext genannt. Einzelne Informationen aus dieser Menge werden Kontextinformationen genannt (vgl. [Tur06]).
Die Definition von Dey wurde von Dey und Abowd [DA00] etwas kompakter verfasst:

> "If a piece of information can be used to characterize the situation of a participant in an interaction, then that information is context. "

Demnach handelt es sich bei einer Information um Kontext, wenn die Information dazu verwendet werden kann, die Situation eines an einer Interaktion beteiligten Teilnehmers zu charakterisieren.

2.1.2 Kontextadaptivität

Der im Englischen verwendete Begriff *Context-Awareness* kann mit Kontextsensitivität oder Kontextadaptivität übersetzt werden (vgl. [Sit09]) und bezeichnet die Anpassung von Anwendungen an den Kontext. Eine der ersten Definitionen des Begriffs *Context-Aware* wurde von Schilit und Theimer [ST94] verfasst und lautet wie folgt:

> "Context-aware computing is the ability of a mobile user's application to discover and react to changes in the environement they are situated in."

Nach den Autoren ist Kontextsensitivität die Fähigkeit einer mobilen Anwendung, Veränderungen in der Umgebung des Benutzer und der Anwendung zu erkennen und

darauf zu reagieren. Schilit und Theimer beschränken den Begriff *Context-Aware* auf mobile Anwendungen, jedoch entspricht diese Beschränkung nicht der Realität, da heutzutage auch stationäre kontextsensitive Anwendungen existieren (z.B. Pennyworth[1]).
Nach Brown et al. [BBC97] sind Anwendungen kontextsensitiv, wenn sie ihr Verhalten abhängig vom Kontext verändern. Pascoe [Pas98] definiert *Context-Awareness* als die Fähigkeit einer Anwendung oder eines Gerätes, unterschiedliche Zustände der Umgebung und ihrer/seiner Selbst wahrzunehmen. Eine auf Anwendungen bezogene Definition des Begriffs liefert Rothermel [Rot08]:

> "Eine Anwendung ist kontextbezogen, wenn ihr Verhalten durch Kontextinformation beeinflusst wird."

Dey [Dey00] definiert den Begriff *Context-Aware* folgendermaßen:

> "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."

Demnach ist ein System kontextsensitiv, wenn es Kontext verwendet, um dem Benutzer relevante Informationen und/oder Dienste anzubieten, wobei die Relevanz der Informationen von den Aufgaben des Benutzers abhängig ist. Anders als viele andere Definitionen beschränkt Dey Kontextsensitivität nicht auf bestimmte Entitäten, wie z.B. Anwendungen. Daher können nach der Definition von Dey auch Systemdienste kontextsensitiv sein, wenn diese Kontext verwenden, um dem Benutzer relevante Informationen und/oder Dienste anzubieten. Das Anbieten von Diensten kann auch für den Benutzer unsichtbar im Hintergrund erfolgen, wenn der Dienst für den Benutzer relevant ist (z.B. Sicherheitsdienst).

2.1.3 Kontextkategorien

Kontext kann nach unterschiedlichen Ansätzen kategorisiert werden. Eine der ersten Klassifikationen von Kontext wurde von Schilit et al. [SAW94] durchgeführt. Die Autoren unterscheiden Kontext nach Standort ("where you are"), Personen in der Nähe des Standorts ("who you are with") und Umfeld ("what resources are nearby"). Springer [Spr04] teilt Kontext in seiner Dissertationsschrift in die Kategorien physikalischer Kontext, technischer Kontext, persönlicher Kontext und situationsbezogener Kontext ein, die in Tabelle 2.1 aufgeführt sind.

[1] http://www.pennyworthproject.org/pennyworth/ (Abgerufen am 18.12.2011)

Kontextklasse	Beispiele
physikalischer Kontext	Ort, Zeit, Temperatur, Lichtintensität
technischer Kontext	Display- und Speichergröße des Endgerätes, Datenrate und Verzögerung des genutzten Kommunikationskanals, verfügbare Energie
persönlicher Kontext	Adresse, Telefonnummer, Termine, bekannte Personen, Voreinstellungen für Anwendungen
situationsbezogener Kontext	Aktivität, Rolle und Aufgabe des Benutzers

Tabelle 2.1: Kontextkategorien (Quelle [Spr04])

Dey und Abowd [DA00] teilen Kontext in die Kategorien Standort, Identität, Aktivität und Zeit ein, die ihrer Ansicht nach wichtiger als andere Kategorien sind. Die Autoren bezeichnen die fünf genannten Kategorien als Primärkontexte und alle anderen Kontextkategorien als Sekundärkontexte. Primärkontexte können nicht nur zur Charakterisierung der Situation einer Entität herangezogen werden, aus ihnen können auch Sekundärkontexte (z.B. Wohnort) abgeleitet werden.
Chen et al. [CK00] unterteilen Kontext in *Low-Level* und *High-Level* Kontext. Dem *Low-Level* Kontext werden die Kontexte Ort, Zeit, Objekte in der Umgebung, Netzwerkbandbreite, Ausrichtung des Gerätes und weitere Kontexte, die durch physikalische Sensoren oder Systemabfragen ermittelt werden können, zugeschrieben. *High-Level* Kontext kann nicht durch Sensoren oder Systemabfragen ermittelt werden, sondern muss durch geeignete Methoden abgeleitet werden. Als geeignet betrachten Chen et al. unter anderem die Verwendung der von Schmidt et al. [SBG98] vorgestellten Methode zur Ableitung von komplexem Kontext aus Sensordaten. Dargie [Dar06] verwendet in seiner Dissertation die Begriffe *Lower-Level* und *Higher-Level* Kontext, die gleichbedeutend mit den Termini *Low-Level* bzw. *High-Level* Kontext sind. Die Kategorisierung von Kontext erfolgt in dieser Arbeit nach dem von Chen et al. vorgestellten Ansatz.

2.1.4 Kontextmodelle

Die maschinelle Verarbeitung von Kontext setzt die Verwendung eines Kontextmodells voraus, welches Kontextinformationen in einer Art und Weise darstellt, die von einem System interpretiert werden kann. Bettini et al. [BBH+10] beschreiben *Object-Role* (ORM) basierte Modelle, räumliche Modelle und ontologiebasierte Modelle als die wichtigsten Ansätze der Kontextmodellierung. Ein weiterer Ansatz der Kontextmodellierung, die hybrid-basierte Modellierung, wird von Topcu [Top] betrachtet. Die hybrid-basierte Kontextmodellierung kombiniert die räumliche und die ontologiebasierte Modellierung.
Chen et al. [CK00] differenzieren zwischen einem Standortmodell für geografische

Positionsdaten und Datenstrukturen für eine generelle Darstellung von Kontextinformationen. Die Datenstrukturen werden von den Autoren in die Kategorien Schlüssel-Wert Paare, Markup-Schemata, objektorientierte Modelle, logikbasierte Modelle und weitere eingeteilt. Der von Chen et al. verwendete Begriff der Datenstrukturen kann in Bezug auf die Kontextmodellierung, mit dem Begriff Kontextmodell gleichgesetzt werden. Strang et al. [SLP04] haben sechs Ansätze der Kontextmodellierung evaluiert, die sie als relevant eingestuft haben. Dazu zählen Schlüssel-Wert Modelle, Markup-Schema Modelle, grafische Modelle, objektorientierte Modelle, logikbasierte Modelle und ontologiebasierte Modelle. Im Folgenden werden aufgrund ihrer Relevanz in dieser Arbeit nur die Schlüssel-Wert Modelle und ontologiebasierte Modelle betrachtet.

2.1.4.1 Schlüssel-Wert Modelle

Schlüssel-Wert Modelle zählen zu den einfachsten Kontextmodellen, da ihre Struktur nur aus einem Schlüssel und einem Wert besteht. Durch den Schlüssel wird die Kontextinformation, die durch den Wert repräsentiert wird, eindeutig identifiziert.
Der Ansatz der Schlüssel-Wert Modellierung ist nicht neu, auch Umgebungsvariablen unter Unix werden durch ein Schlüssel-Wert Paar dargestellt. Umgebungsvariablen wurden bereits von Schilit et al. [SAW94] und Voelker et al. [VB94] verwendet, um Kontextinformationen zu modellieren und an eine Anwendung zu übermitteln.

2.1.4.2 Ontologiebasierte Modelle

Ontologien sind formale Systeme der Informatik, die zur Repräsentation von Wissen in maschinenlesbarer Form verwendet werden. Sie ermöglichen durch die Darstellung von Wissen die Modellierung eines Ausschnitts aus der realen Welt. Durch die standardisierte Form der Repräsentation gewährleisten Ontologien, in Bezug auf eine gemeinsame Kontextnutzung, die Interoperabilität unterschiedlicher Anwendungen und Systeme.
Der Begriff Ontologie entspringt ursprünglich der Philosophie und bezieht sich auf die Lehre des Seins. Definitionen aus dem Bereich der Informatik wurden von Chandrasekaran et al. [CJB99] und Voß [Voß03] verfasst. Voß definiert den Begriff Ontologie als "formal definiertes System von Konzepten und Relationen zwischen ihnen. Zusätzlich enthalten Ontologien (zumindest implizit) Regeln". Anhand von Regeln können neue Informationen aus vorhandenen Kontextinformationen abgeleitet werden. Der Ableitungsprozess wird auch *Context-Reasoning* genannt. Aus der Definition von [Voß03] leitet Ay [Ay07] in Bezug auf Kontextmodellierung und *Context-Reasoning* die folgenden Punkte ab:

- "A context model is also a system of concepts (entities) and relations,

which makes an ontology a possible mean for context modelling.“

- “An ontology is 'formally defined', which is a precondition for a computer to interpret it, e.g. for reasoning purposes.“
- “Rules can be used to implement context reasoning.“

Ontologien können mit formalen Sprachen wie z.B. DAML+OIL[2], RDF-Schema[3] oder OWL[4] erstellt werden. Die Sprache OWL basiert auf den Sprachen RDF und RDFS. Sie wurde in den Untersprachen OWL Lite, OWL DL, und OWL Full definiert, die sich durch die zur Verfügung stehenden Sprachkonstrukte unterscheiden. Abbildung 2.1 zeigt einen Ausschnitt aus einer in OWL modellierten Wildtier-Ontologie.

Listing 2.1: Ausschnitt einer in OWL modellierten Ontologie (nach [AvH09])

```
1  <owl:Class rdf:ID="Löwe">
2    <rdfs:comment>Löwen sind Tiere die nur Pflanzenfresser essen</rdfs:comment>
3    <rdfs:subClassOf rdf:type="#Fleischfresser"/>
4    <rdfs:subClassOf>
5      <owl:Restriction>
6        <owl:onProperty rdf:resource="#isst"/>
7        <owl:allValuesFrom rdf:resource="#Pflanzenfresser"/>
8      </owl:Restriction>
9    </rdfs:subClassOf>
10 </owl:Class>
```

2.1.5 Context-Reasoning

Nach Auffassung der Autoren Dargie und Hamann [DH06] bedeutet Kontextbewusstsein, den erfassten Sensordaten eine sinnvolle Bedeutung zuzuordnen. Aus dieser Aussage leiten Dargie und Hamann die Forderung nach der Verarbeitung von Sensordaten (*Low-Level* Kontext) ab, um eine sinnvolle Abstraktion der realen Welt (*High-Level* Kontext) zu erreichen.
High-Level Kontext, der für kontextsensitive Anwendungen von Interesse ist, kann nicht direkt erfasst werden (z.B. durch Sensoren), sondern muss abgeleitet werden (vgl. [vSvHW+06]). Der Begriff *Context-Reasoning* beschreibt in diesem Zusammenhang die Ableitung von impliziten Fakten (*High-Level* Kontext) aus expliziten Kontextinformationen (*Low-Level* Kontext) (vgl. [Ay07]). Nurmi und Floréen [NF11] definieren *Context-Reasoning* wie folgt:

[2] DARPA Agent Markup Language (`http://www.daml.org/2001/03/daml+oil-index.html`)
[3] Resource Description Framework Schema (RDFS) (`http://www.w3.org/TR/rdf-schema/`)
[4] Web Ontology Language (`http://www.w3.org/TR/owl-features/`)

> "A more precise definition for context-reasoning is deducing new and relevant information to the use of application(s) and user(s) from the various sources of context-data."

Die Ableitung von *High-Level* Kontext aus *Low-Level* Kontext wird von Nurmi und Floréen [NF11] auch als Kontextinferenz bezeichnet. Ein weiterer Anwendungsfall des *Context-Reasonings*, neben der Ableitung von *High-Level* Kontext, ist die Ermittlung und Auflösung von Inkonsistenzen in Kontextinformationen (vgl. [Ay07], [WGZP04]).
Die Methoden des *Context-Reasonings* wurden von Wang et al. [WGZP04] in die Kategorien *Ontology-Reasoning*, durch Anwendung einer Beschreibungslogik und benutzerdefiniertes *Reasoning* aufgeteilt. Nurmi und Floréen [NF11] führen als geeignete *Reasoning*-Mechanismen probabilistisches *Reasoning* und die Verwendung von Ontologien, in Kombination mit logischer Schlussfolgerung, auf. Van Sinderen et al. [vSvHW+06] verwenden neben *Ontology-Reasoning* auch die Methode des maschinellen Lernens. Ein Beispiel des *Ontology-Reasoning* durch Anwendung einer Beschreibungslogik zeigt Tabelle 2.2.

Eingabe	Reasoning Regeln (Beschreibungslogik)	(?P rdf:type owl:TransitiveProperty)∧ (?A ?P ?B) ∧ (?B ?P ?C) ⇒ (?A ?P ?C)
		(?P owl:inverseOf ?Q) ∧ (?X ?P ?Y) ⇒ (?Y ?Q ?X)
	Expliziter Kontext	< owl:ObjectProperty rdf:ID="locatedIn" > < rdf:type="owlTransitiveProperty"/ > < owl:inverseOf rdf:resource="#contains"/ > < /owl:ObjectProperty > < Person rdf:ID="User" > < locatedIn rdf:resource="#Bedroom"/ > < /Person > < Room rdf:ID="Bedroom" > < locatedIn rdf:resource="#Home"/ > < /Room >
Ausgabe	Impliziter Kontext	< Person rdf:ID="User" > < locatedIn rdf:resource="#Home"/ > < /Person > < Building rdf:ID="Home" > < contains rdf:resource="#Bedroom"/ > < contains rdf:resource="#User"/ > < /Building > < Room rdf:ID="Bedroom" > < contains rdf:resource="#User"/ > < /Room >

Tabelle 2.2: Ontology-Reasoning mit Beschreibungslogik (nach [WGZP04])

Auf die expliziten Kontextinformationen wurden die Regeln *TransitiveProperty* und *inverseOf* angewandt, die den Eigenschaften Transitivität und inverses Element

der Prädikatenlogik erster Stufe - einer Beschreibungslogik - entsprechen. Durch Anwendung der transitiven Regel (*owl:TransitiveProperty*) auf die expliziten Kontextinformationen folgt:

$$\begin{aligned}&(?\text{locatedIn rdf:type owl:TransitiveProperty}) \wedge (?User\ ?locatedIn\ ?Bedroom)\\ &\wedge(?Bedroom\ ?locatedIn\ ?Home) \Rightarrow (?User\ ?locatedIn\ ?Home)\end{aligned} \tag{2.1}$$

Durch Anwendung der Umkehrfunktion (*owl:inverseOf*) folgt:

$$\begin{aligned}&(?\text{locatedIn owl:inverseOf ?contains}) \wedge (?Bedroom\ ?locatedIn\ ?Home)\\ &\Rightarrow (?Home\ ?contains\ ?Bedroom)\end{aligned} \tag{2.2}$$

$$\begin{aligned}&(?\text{locatedIn owl:inverseOf ?contains}) \wedge (?User\ ?locatedIn\ ?Home)\\ &\Rightarrow (?Home\ ?contains\ ?User)\end{aligned} \tag{2.3}$$

$$\begin{aligned}&(?\text{locatedIn owl:inverseOf ?contains}) \wedge (?User\ ?locatedIn\ ?Bedroom)\\ &\Rightarrow (?Bedroom\ ?contains\ ?User)\end{aligned} \tag{2.4}$$

Ein nach Wang et al. [WGZP04] flexiblerer Mechanismus des *Context-Reasonings* ist das benutzerdefinierte *Reasoning*. Benutzerdefiniertes *Context-Reasoning* verwendet Prädikatenlogik erster Ordnung zur Formulierung benutzerdefinierter Regeln. Tabelle 2.3 stellt benutzerdefinierte *Reasoning*-Regeln dar, die die Ableitung einiger Benutzersituationen ermöglichen.

Situation	Reasoning-Regeln
Schlafen	$(?u\ locatedIn\ Bedroom) \wedge (Bedroom\ lightLevel\ LOW)$ $\wedge(Bedroom\ drapeStatus\ CLOSED)$ $\Rightarrow (?u\ situation\ SLEEPING)$
Duschen	$(?u\ locatedIn\ Bathroom) \wedge (WaterHeater\ locatedIn\ Bathroom)$ $\wedge(Bathroom\ doorStatus\ CLOSED) \wedge (WaterHeater\ status\ ON)$ $\Rightarrow (?u\ situation\ SHOWERING)$
Kochen	$(?u\ locatedIn\ Kitchen) \wedge (ElectricOven\ locatedIn\ Kitchen)$ $\wedge(ElectricOven\ status\ ON)$ $\Rightarrow (?usituationCOOKING)$

Tabelle 2.3: Benutzerdefinierte Reasoning-Regeln (nach [WGZP04])

2.2 Begriffe der IT-Sicherheit

In diesem Abschnitt werden grundlegende Begriffe der IT-Sicherheit, die im weiteren Verlauf dieser Arbeit verwendet werden, beschrieben. Die Begriffsdefinitionen ermöglichen die Einordnung der einzelnen Begriffe in den Gesamtzusammenhang

der Thematik. Aus dem Zusammenhang werden im weiteren Verlauf dieser Arbeit Schritte extrahiert, die für die Ableitung von Sicherheitsmaßnahmen aus vorliegenden Kontextinformationen notwendig sind.
Alle an einem IT-Szenario beteiligten Entitäten werden als Objekte bezeichnet und nach passiven und aktiven Objekten differenziert. Passive Objekte sind nur in der Lage Informationen zu speichern (z.B. Dateien). Aktive Objekte können Informationen verarbeiten und werden auch als Subjekte bezeichnet (z.B. Prozesse oder Personen). Die von informationstechnischen Systemen verarbeiteten Daten haben an sich keinen Wert (z.B. leere Word-Datei), erst durch das Hinzufügen wertvoller Informationen werden die Daten schützenswert (vgl. [Kar10]). Der Schutz schützenswerter Informationen ist ein Anliegen der IT-Sicherheit, das durch die Schutzziele genauer spezifiziert wird.

2.2.1 Schutzziele

Schutzziele leiten sich aus Unternehmenszielen, Verordnungen und Gesetzen ab (vgl. [Kar10]) und verfolgen das Ziel, Informationen eines IT-Szenarios zu schützen. Wird ein Schutzziel verletzt, kann dadurch ein Schaden entstehen. Zu den klassischen Schutzzielen der IT-Sicherheit gehören Authentizität, Datenintegrität, Informationsvertraulichkeit, Verfügbarkeit, Verbindlichkeit und Anonymität.

Authentizität

Authentizität beschreibt die Sicherstellung der Echtheit eines Objekts bzw. Subjekts (vgl. [Eck09]). Die Authentizität einer Person kann durch biometrische Merkmale oder durch die Überprüfung der Identität (Benutzerkennung) in Kombination mit einem Identitätsnachweis (geheimes Passwort) sichergestellt werden. Bei Programmen oder Daten beschränkt sich die Sicherstellung der Authentizität auf die Urheberschaft. Die Überprüfung der spezifizierten Funktionalität bleibt in diesem Fall aus (vgl. [Eck09]).

Datenintegrität

Das Schutzziel Datenintegrität soll die unberechtigte Veränderung oder Manipulation der zu schützenden Daten durch Subjekte unterbinden (vgl. [Kar10], [Eck09]). Eine Verletzung der Datenintegrität kann nicht nur durch Angriffe ausgelöst werden, sondern auch durch Fehlfunktionen von Prozessen. Die Beschränkung des Zugriffs kann durch Mechanismen der Zugriffskontrolle erfolgen, während Manipulationen durch Prüfsummen erkannt werden können.

Informationsvertraulichkeit

Die Informationsvertraulichkeit ist gewährleistet, wenn ein System keine unautorisierte Informationsgewinnung ermöglicht (vgl. [Eck09]). Mechanismen der Zugriffskontrolle bieten keinen ausreichenden Schutz vor unautorisierter Informationsgewinnung, wenn der Zugriff auf die Daten von außerhalb des Systems erfolgt (z.B. Diebstahl des Datenträgers). Die Verschlüsselung der zu schützenden Daten, kann, unabhängig vom Aufenthaltsort der Daten, Schutz vor unautorisierter Informationsgewinnung bieten.

Verfügbarkeit

Die Verfügbarkeit beschreibt das Ziel, dass ein System oder Daten für einen vorgesehenen Zweck tatsächlich genutzt werden können (vgl. [Kar10]). Die Verfügbarkeit von Systemen oder Daten wird nicht nur durch Angriffe gefährdet, sondern auch durch Fehlfunktionen der Prozesse, die Daten verarbeiten bzw. Dienste bereitstellen.

Verbindlichkeit

Ein System gewährleistet die Verbindlichkeit einer Aktion, wenn ein Subjekt die Durchführung der Aktion im Nachhinein nicht abstreiten kann (vgl. [Eck09]). So kann z.B. eine Person das Absenden einer E-Mail nicht abstreiten, wenn die E-Mail mit ihrer digitalen Signatur unterschrieben wurde.

Anonymität

Anonymität wird gewährleistet, wenn Subjekte nicht identifiziert und deren Aktionen diesen nicht zugeordnet werden können (vgl. [Kar10]). Die anonyme Nutzung von Systemen widerstrebt dem Schutzziel Authentizität, so dass das Schutzziel Anonymität nicht zusammen mit dem Schutzziel Authentizität erfüllt werden kann. Eine schwächere Form der Anonymisierung stellt die Pseudonymisierung dar, die nicht im Konflikt mit dem Schutzziel Authentizität steht. Die Pseudonymisierung verändert Daten durch eine Zuordnungsvorschrift, so dass diese einem Subjekt nur durch Kenntnis dieser Zuordnungsvorschrift zugeordnet werden können (vgl. [Eck09]).

2.2.2 Schwachstellen

Eine Schwachstelle ist eine “technische, physische oder organisatorische Eigenschaft, die es erlaubt, dass mindestens ein Schutzziel verletzt wird“ [Kar10]. Die vorhandenen Schwachstellen eines Systems sind für gewöhnlich statisch und ändern sich ohne präventive Maßnahmen nicht. Eine Ausnahme bildet hier die Neuinstallation bzw.

Aktualisierung von Software, durch die neue Schwachstellen auftreten können.

2.2.3 Bedrohungen

Eine Bedrohung ist ein "potentiell mögliches Ereignis, das zu einer Verletzung eines Schutzziels führt" [Kar10]. Die Existenz einer Bedrohung setzt voraus, dass eine Schwachstelle vorhanden ist und ein Ereignis, mit der der Schwachstelle zugrunde liegenden Eigenschaft des Systems, zu einer Verletzung der Schutzziele führt (vgl. [Kar10]). Bedrohungen beruhen nicht nur auf vorsätzlichen Handlungen, sondern auch auf höherer Gewalt, Fahrlässigkeit, organisatorischen Mängeln und technischem Versagen (vgl. [Eck09]).

2.2.4 Sicherheitsmaßnahmen

Bei Sicherheitsmaßnahmen handelt es sich um "Maßnahmen die dazu beitragen, dass ein Sicherheitsziel erreicht wird" [Kar10]. Abhängig von den Bedrohungen kann ein Sicherheitsziel teilweise nur durch die Kombination mehrerer Maßnahmen erreicht werden, andererseits kann sich eine Sicherheitsmaßnahme auch positiv auf das Erreichen mehrerer Sicherheitsziele auswirken (vgl. [Kar10]). Sicherheitsmaßnahmen sollen das Risiko einer Bedrohung reduzieren, indem sie entweder die Eintrittswahrscheinlichkeit einer Bedrohung, oder den möglichen Schaden, der durch die Bedrohung verursacht wird, verringern. Eine hundertprozentige Sicherheit kann jedoch nicht erreicht werden, da die Eintrittswahrscheinlichkeit nicht auf null reduziert werden kann.

2.3 Zusammenfassung

In diesem Kapitel wurden grundlegende Begriffe aus den Bereichen der kontextverarbeitenden Systeme und der Informationssicherheit definiert, welche in dieser Arbeit verwendet werden.
Nach der Darstellung gängiger Definitionen des Kontextbegriffs, wurden die Eigenschaften kontextadaptiver Systeme betrachtet. Die Verwendung ontologiebasierter Kontextmodelle in kontextadaptiven Systemen ermöglicht, neben der gemeinsamen Kontextnutzung unterschiedlicher Anwendungen, auch *Context-Reasoning*. Durch eine Einteilung von Kontext in *High-Level* und *Low-Level* Kontext, können implizite und explizite Kontextinformationen unterschieden werden.
Im Rahmen der Betrachtung grundlegender Begriffe der IT-Sicherheit wurde der Zusammenhang zwischen Schwachstellen, Bedrohungen und Sicherheitsmaßnahmen

dargelegt. Eine Bedrohung entsteht durch die Präsenz einer Schwachstelle und eines potentiellen Ereignisses, durch welches die Schwachstelle ausgenutzt werden kann. Sicherheitsmaßnahmen reduzieren das Risiko einer Bedrohung, indem sie die ihr zugrunde liegende Schwachstelle beseitigen oder verhindern, dass diese ausgenutzt werden kann.

3 Kontextabhängige Sicherheitsmaßnahmen

Die Sicherheit eines mobilen Geräts ist abhängig von der Situation, in der sich das mobile Gerät befindet (vgl. [Joh09], [JAB10]). Um den wechselnden Bedrohungen mobiler Szenarien zu begegnen, empfehlen Fischer und Karsch [FK11] einen flexiblen Einsatz von Sicherheitsmaßnahmen, abhängig von den Bedrohungen.
In diesem Kapitel wird ein Ansatz zur Ableitung von Sicherheitsmaßnahmen aus relevanten Kontextinformationen beschrieben. Aus Kontextinformationen abgeleitete Sicherheitsmaßnahmen, sind dem Kontext aus dem sie abgeleitet wurden, angepasst und werden im Folgenden als kontextabhängige oder kontextadaptive Sicherheitsmaßnahmen bezeichnet. Des Weiteren werden in diesem Kapitel Möglichkeiten der Umsetzung kontextabhängiger Sicherheitsmaßnahmen aufgezeigt.
Sicherheitsmaßnahmen mobiler Szenarien sollen sich dem aktuellen Kontext und den daraus resultierenden Bedrohungen anpassen, um das Risiko von Bedrohungen zu verringern und dabei die Ressourcen mobiler Geräte, im Gegensatz zu stationären Ansätzen, zu schonen. Die Ableitung kontextabhängiger Sicherheitsmaßnahmen setzt vorab die Erkennung von Bedrohungen voraus. Bedrohungen können aus relevanten Kontextinformationen abgeleitet werden, wenn die Umstände, unter denen die Bedrohungen auftreten, bekannt sind. Eine Kategorisierung der relevanten Kontextinformationen wird in Abschnitt 3.1 vorgenommen. Die Betrachtung der relevanten Kontextinformationen, die eine Ableitung von Bedrohungen ermöglichen, erfolgt in Abschnitt 3.2. Im darauf folgenden Abschnitt wird die Ableitung von Bedrohungen betrachtet.
Das Risiko ermittelter Bedrohungen soll durch geeignete Sicherheitsmaßnahmen verringert werden. Welche Sicherheitsmaßnahmen dazu geeignet sind lässt sich nicht allgemein definieren, sondern ist in der Regel vom Einzelfall abhängig. Daher ist auch keine vollständig automatisierte Ableitung von Sicherheitsmaßnahmen auf Basis der ermittelten Bedrohungen möglich. Stattdessen werden Erfahrungswerte benötigt, wie sie z.B. durch Signaturen zur Erkennung von Malware verwendet werden. Die Ableitung von Sicherheitsmaßnahmen wird in Abschnitt 3.4 beschrieben. In Abschnitt 3.5 werden Möglichkeiten der Umsetzung von Sicherheitsmaßnahmen betrachtet sowie in Abschnitt 3.6 mögliche Probleme aufgezeigt, die vermieden werden sollten.

3.1 Kontextkategorien

Die für die Ableitung von Bedrohungen relevanten Kontextinformationen können im Allgemeinen der Anwendungs-, Benutzer- oder Geräteumgebung zugeschrieben werden. Die Einteilung der Kontextinformationen in diese Kategorien ermöglicht eine bessere Trennung von Kontextinformationen, wodurch Sicherheitsmaßnahmen nur an die Anwendungen übermittelt werden können, die auch einer Bedrohung ausgesetzt sind.

3.1.1 Anwendungskontext

Aufbauend auf der Kontextdefinition von Dey [Dey00] wird Anwendungskontext wie folgt definiert:

Definition

> Anwendungskontext ist jegliche Menge von Informationen aus der Anwendungsumgebung, die dazu verwendet werden kann, den Zustand einer Anwendung zu beschreiben. Die Anwendungsumgebung entspricht dem Prozessadressraum der Anwendung.

Aus der Definition folgt, dass der Anwendungskontext einer Anwendung für gewöhnlich nur durch Systemkomponenten oder die Anwendung selbst ermittelt werden kann, da andere Anwendungen keinen Zugriff auf die Informationen haben. Im Gegensatz zur Anwendung können Systemkomponenten den Zustand der Anwendung nicht vollständig erfassen, weil ihnen die benötigte Interpretationslogik fehlt. Der Kontext *Anwendung versendet E-Mail* kann üblicherweise nur von der Anwendung selbst erfasst werden. Eine Systemkomponente kann hingegen nur wahrnehmen, dass Daten versendet werden.

Anwendungen sind aufgrund der Schnittstellen, durch die sie Daten ausgeben und entgegennehmen, Bedrohungen ausgesetzt. Der Zugriff auf eine Schnittstelle erfolgt durch den Aufruf einer Bibliotheksfunktion, die einen Systemaufruf auslöst. Die Benutzung dieser Schnittstellen kann zu Bedrohungen führen, da Dritte die Daten abgreifen, manipulieren oder Malware einschleusen können. Infolgedessen sollte vor der Ausführung von Funktionen, die diese Schnittstellen nutzen, erst der Anwendungskontext übermittelt werden und die Umsetzung der Sicherheitsmaßnahmen abgewartet werden. Aus dem übermittelten Anwendungskontext können Bedrohungen ermittelt werden, die im Zusammenhang mit der Schnittstelle stehen. Aus den ermittelten Bedrohungen können dann Sicherheitsmaßnahmen abgeleitet werden, durch deren Umsetzung das Risiko der Bedrohungen verringert werden kann. Er-

folgt dieser Prozess vor der Nutzung der Schnittstelle, treten bei der Nutzung der Schnittstelle in der Regel keine Bedrohungen auf.
Soll eine Anwendung kontextabhängige Sicherheitsmaßnahmen verwenden, dann besteht eine zusätzliche Aufgabe der Anwendungsentwickler darin, den Anwendungskontext bei relevanten Aufrufen den Anforderungen der zuständigen Architekturkomponente entsprechend zu übergeben. Zuvor müssen relevante Aufrufe durch die Anwendungsentwickler identifiziert werden. Für eine möglichst umfassende Ermittlung von Bedrohungen sollte der Anwendungskontext möglichst vollständig vorliegen.

3.1.2 Gerätekontext

Ebenfalls aufbauend auf der Kontextdefinition von Dey [Dey00] wird Gerätekontext wie folgt definiert:

Definition

> Gerätekontext ist jegliche Menge von Informationen, die dazu verwendet werden kann, den Zustand eines mobilen Geräts, ausgenommen seiner Anwendungen, zu beschreiben.

Als Gerätekontext werden Informationen beschrieben, die aus der Umgebung des mobilen Geräts ermittelt werden können (z.B. die Position des Geräts, die Helligkeit, oder die Umgebungslautstärke), aber auch Informationen, die den internen Zustand des Gerätes beschreiben (z.B. die Systemzeit, das Systemdatum, Informationen zu Kommunikationsverbindungen), werden als Gerätekontext bezeichnet. Anwendungsspezifische Informationen sind hingegen dem Anwendungskontext zuzuschreiben.
Kontextinformationen, die den internen Zustand des Geräts beschreiben, können durch Systemfunktionen abgefragt werden soweit das Betriebssystem Funktionen zur Abfrage der gewünschten Informationen anbietet. Ausgewählte Kontextinformationen aus der Umgebung des mobilen Gerätes können mittelbar bzw. unmittelbar durch die Sensoren des Gerätes ermittelt werden.

3.1.3 Benutzerkontext

Vorhandene Definitionen beschreiben Benutzerkontext in aufzählender Form ohne den Begriff allgemein zu definieren; so auch Chen et al. [CK00], die Benutzerkontext wie folgt beschreiben:

> "the user's profile, location, people nearby, even the current social situation."

Aufbauend auf den Definitionen von [CK00] und [Dey00] wird Benutzerkontext folgendermaßen definiert:

Definition

> Benutzerkontext ist jegliche Menge von Informationen, die dazu verwendet werden kann, den Benutzer selbst, die Situation in der es sich befindet, seine Handlungen und seine Absichten zu beschreiben.

Zu den Informationen, die dem Benutzerkontext zugeordnet werden können, gehören jene, welche die Situation des Benutzers beschreiben (z.B. die Position des Benutzers, die lokale Uhrzeit des Ortes an dem sich der Benutzer aufhält oder das aktuelle Datum), aber auch Informationen über die Person selbst (z.B. die Identität) oder seine Absichten (z.B. die Absicht ins Bett zu gehen). Trotz einiger Gemeinsamkeiten ist der Gerätekontext in vielen Fällen nicht dem Benutzerkontext gleichzusetzen, da sowohl das mobile Gerät als auch der Benutzer sich in unterschiedlichen Situationen und Umgebungen befinden können. So muss die Position oder die Uhrzeit des Benutzers nicht mit der Position oder der Systemzeit des mobilen Gerätes übereinstimmen.
Die direkte Ermittlung des Benutzerkontexts ist nur eingeschränkt möglich, da der Benutzer seinen Benutzerkontext zwar selbst kennt (teilweise nur unterbewusst), ihn für gewöhnlich aber weder selbst weitergibt, noch diesen explizit durch Sensoren bestimmen lässt. So kann zum Beispiel die Position des Benutzers nur anhand der Geräteposition bestimmt werden, die aber nicht der Benutzerposition entsprechen muss. Auch die Interaktion des Benutzers mit dem mobilen Gerät beziehungsweise mit den installierten Anwendungen kann nicht direkt aus der Benutzerumgebung ermittelt werden. Jedoch kann der Benutzerkontext üblicherweise, wenn auch nicht immer vollständig, aus dem Geräte- und Anwendungskontext abgeleitet werden.

3.2 Sicherheitsspezifischer Kontext

Mit der Betrachtung des Kontextbegriffs im Zusammenhang mit IT-Sicherheit haben sich An et al. [ABKS09] und Mostéfaoui et al. [MB03] beschäftigt. An et al. [ABKS09] definieren sicherheitsspezifischen Kontext als jede Information, die dazu verwendet werden kann die Bedrohungen eines mobilen Geräts zu beurteilen. Eine weitere Definition wurde von Mostéfaoui et al. [MB03] verfasst:

> "A security context is a set of information collected from the user's environment and the application environment and that is relevant to the security infrastructure of both the user and the application."

Die Autoren definieren sicherheitsspezifischen Kontext als Menge von Informationen aus der Anwendungs- und Benutzerumgebung, die direkt oder indirekt Einfluss auf die Sicherheit haben. Informationen aus der Geräteumgebung werden von Mostéfaoui et al. [MB03] nicht dem sicherheitsspezifischen Kontext zugeschrieben, obwohl auch sie Einfluss auf die Sicherheit haben können. So ist die Position des mobilen Geräts eine Information aus der Geräteumgebung, welche nicht immer aus der Anwendungsumgebung oder Benutzerumgebung hergeleitet werden kann.
Mithilfe von relevantem Kontext soll versucht werden, Bedrohungen zu ermitteln und ihnen möglichst mit geeigneten Sicherheitsmaßnahmen zu begegnen. Aufbauend auf den Definitionen von [Dey00], [ABKS09] und [MB03] wird relevanter Kontext in Bezug auf die Ableitung von Sicherheitsmaßnahmen daher wie folgt definiert:

Definition

> Sicherheitsspezifischer Kontext ist jegliche Menge von Kontextinformationen des Anwendungs-, Benutzer- oder Gerätekontexts, die dazu verwendet werden kann, benötigte Sicherheitsmaßnahmen eines mobilen Geräts zu ermitteln.

Da die Bedrohungen eines mobilen Geräts bekannt sein müssen, bevor benötigte Sicherheitsmaßnahmen abgeleitet werden können, dürfen Kontextinformationen auch als sicherheitsspezifisch angesehen werden, wenn daraus Bedrohungen ermittelt werden können.

3.3 Ableitung von Bedrohungen

Die Bedrohungen eines mobilen Geräts können aus dessen sicherheitsspezifischem Kontext abgeleitet werden, wenn dieser die zur Ableitung benötigten Informationen enthält und Regeln definiert wurden, die eine Ableitung von Bedrohungen ermöglichen. Voraussetzung für die Ableitung von Bedrohungen ist ein Bedrohungsmodell, welches die Bedrohungen formal beschreibt. Eine formale Beschreibung von Bedrohungen ist nötig, da die Ableitung von Bedrohungen anhand logischer Regeln erfolgen soll. Des Weiteren müssen die sicherheitsspezifischen Kontextinformationen durch ein Kontextmodell dargestellt werden, das die Ableitung von High-Level Kontext ermöglicht.

3.3.1 Schwachstellen- und Bedrohungsmodell

Die Existenz einer Bedrohung setzt eine vorliegende Schwachstelle und ein potentielles Ereignis voraus, welches durch Ausnutzen der Schwachstelle ein Schutzziel verletzen kann (vgl. [Kar10]). Um Bedrohungen zu erkennen, müssen diese zuvor

formal anhand ihrer Merkmale beschrieben werden. Zu den Merkmalen einer Bedrohung zählen die der Bedrohung zugrunde liegenden Schwachstellen und potentiellen Ereignisse sowie gegebenenfalls weitere sicherheitsspezifische Kontextinformationen, die eine eindeutige Beschreibung der Bedrohung ermöglichen. Merkmale von Bedrohungen basieren auf Erfahrungswerten, daher gibt es für die Modellierung einer bestimmten Bedrohung kein Patentrezept. Da bestimmte Schwachstellen und Ereignisse auch mehreren Bedrohungen angehören können, sollen diese zur Vermeidung von Redundanzen als eigenständige Elemente modelliert und mit den modellierten Bedrohungen verknüpft werden.
Schwachstellen können als statisch betrachtet werden, weil sie sich in der Regel nicht wie Bedrohungen mit dem Kontext verändern. Da sich die vorhandenen Schwachstellen von Gerät zu Gerät unterscheiden können, ist es jedoch sinnvoll, die vorliegenden Schwachstellen für jedes mobile Gerät zu bestimmen und nicht statisch festzulegen. Eine Schwachstelle kann durch Attribute und Bedingungen beschrieben werden, die eine Bestimmung der Schwachstelle auf Basis von sicherheitsspezifischem Kontext zulassen. Eine Schwachstelle ist auf dem mobilen Gerät vorhanden, wenn die Attribute die durch Bedingungen spezifizierten Werte annehmen.

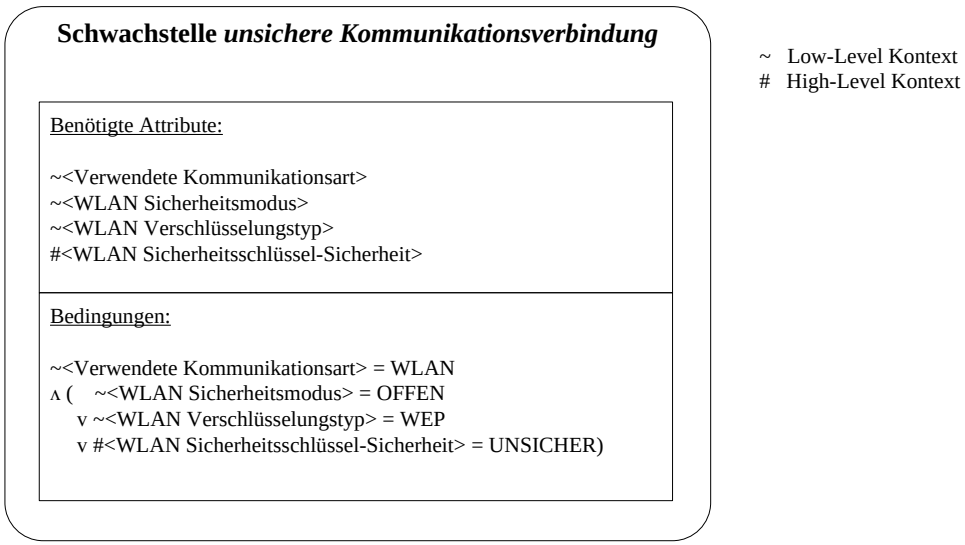

Abbildung 3.1: Schwachstellenmodell einer unsicheren Kommunikationsverbindung

Um die Definition einer Schwachstelle zu veranschaulichen, wurde ein Schwachstellenmodell für die Schwachstelle *unsichere Kommunikationsverbindung* definiert, das in Abbildung 3.1 dargestellt ist. Das Schwachstellenmodell beschreibt zuerst die Attribute, durch die sich die Existenz der Schwachstelle nachweisen lässt. Dazu gehören für die Schwachstelle *unsichere Kommunikationsverbindung* der *verwendete Kommunikationskanal*, der *WLAN Sicherheitsmodus*, der *WLAN Verschlüsselungstyp* und die *WLAN Sicherheitsschlüssel-Sicherheit*. Die Bedingung $verwendete\ Kommunikationsart = WLAN$ prüft, ob es sich bei der verwendeten

Kommunikationsart um WLAN handelt. Andere Kommunikationsarten werden von diesem Schwachstellenmodell nicht abgedeckt, können aber durch Hinzufügen weiterer Attribute und Bedingungen integriert werden.
Um korrekte Aussagen über den Zustand einer Bedingung zu machen, dürfen nicht beliebige Attributsnamen verwendet werden, sondern diese müssen normiert sein. Dadurch können Fehler in der Auswertung der Bedingungen verhindert werden, die z.B. durch die Übermittlung des Attributs *W-Lan* anstatt *WLAN* entstehen würden. Im vorliegenden Beispiel liegt nur dann die Schwachstelle *unsichere Kommunikationsverbindung* vor, wenn das WLAN entweder offen ist, WEP als Verschlüsselungstyp eingesetzt wird oder der Sicherheitsschlüssel unsicher ist. Bei dem Attribut *WLAN Sicherheitsschlüssel-Sicherheit* handelt es sich anders als bei den anderen Attributen dieses Beispiels um High-Level Kontext, der nicht direkt im sicherheitsspezifischen Gerätekontext enthalten ist, sondern erst ermittelt werden muss.
Potentielle Ereignisse werden ähnlich wie Schwachstellen durch Attribute und Bedingungen modelliert. Sind die Bedingungen eines Ereignismodells erfüllt, dann liegt das potentielle Ereignis vor.
Eine Bedrohung liegt vor, wenn die Schwachstellen und potentiellen Ereignisse, die ihr zugrunde liegen, existieren und die Schwachstellen dadurch ausgenutzt werden können. Durch Attribute und Regeln kann festgelegt werden, bei welchen Schwachstellen und potentiellen Ereignissen eine bestimmte Bedrohung ermittelt wird.

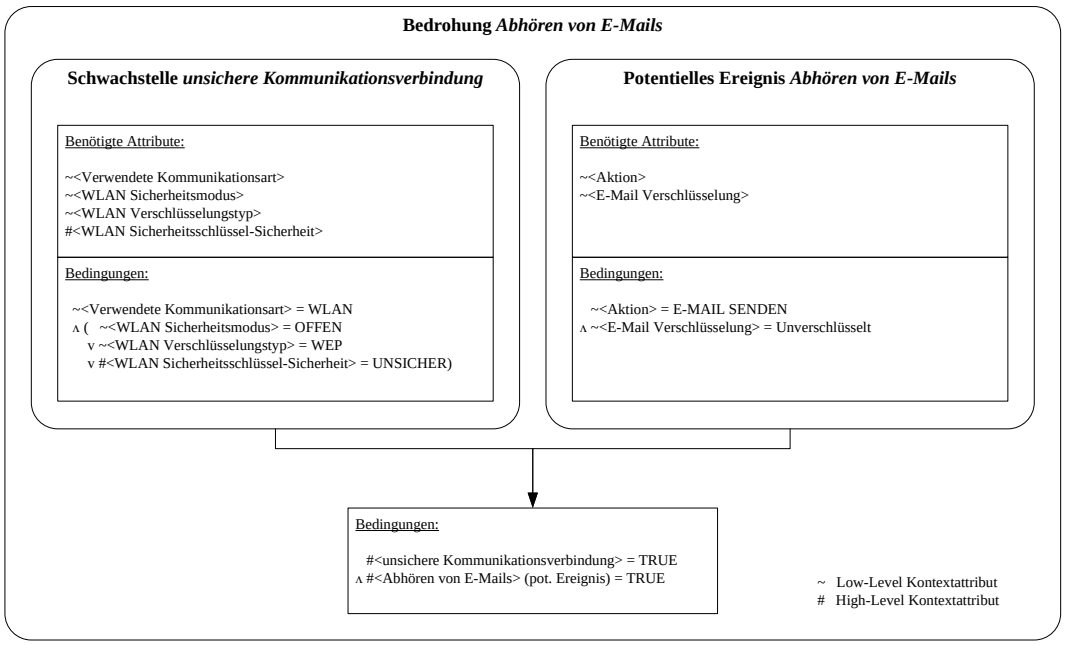

Abbildung 3.2: Bedrohungsmodell der Bedrohung *Abhören von E-Mails*

Abbildung 3.2 zeigt ein Bedrohungsmodell der Bedrohung *Abhören von E-Mails*. Das potentielle Ereignis *Abhören von E-Mails* kann eintreten, wenn eine unverschlüsselte E-Mail versendet wird. Wird die E-Mail wie in diesem Modell über eine unsiche-

re Kommunikationsverbindung versendet, liegt nach diesem Bedrohungsmodell eine Bedrohung vor.

3.3.2 Kontextmodell

Bedrohungsmodelle definieren, welche Kontextinformationen für die Ableitung von Bedrohungen benötigt werden. Ein Bedrohungsmodell benötigt eine bestimmte Anzahl von Kontextinformationen, aus denen es durch Regeln ermitteln kann, ob die Bedrohung vorliegt oder nicht. Einige dieser Kontextinformationen werden von Sensoren oder von Anwendungen bereitgestellt. Andere Kontextinformationen, die nicht direkt durch Sensoren oder durch eine Anwendung erfasst werden können, müssen durch Folgerung aus den vorliegenden Kontextinformationen abgeleitet oder anderweitig bezogen werden. Um Kontextinformationen verarbeiten zu können, müssen diese mithilfe einer formalen maschinenlesbaren Sprache in einem Kontextmodell dargestellt werden. Ein geeignetes Kontextmodell ermöglicht die Bestimmung vorliegender Schwachstellen, die Ermittlung potentieller Ereignisse sowie die Ableitung von Bedrohungen aufgrund von Regeln.
Strang et al. [SLP04] haben unterschiedliche Kontext-Modellierungsansätze nach bestimmten Kriterien untersucht. Sie sind aufgrund ihrer Analyse zu dem Schluss gekommen, dass ontologiebasierte Modelle unter allen untersuchten Ansätzen am besten zur Verarbeitung von Kontext geeignet sind.
Ontologien sind formale Beschreibungen von Wissen und wurden bereits von Xu et al. [XXW09] für die Repräsentation von Sicherheitswissen verwendet. Ontologien eigenen sich auch zur Modellierung von Bedrohungsmodellen und Sicherheitsmaßnahmen (vgl. [FK11]). Außerdem unterstützen Ontologien Context Reasoning, wodurch die Ableitung von High-Level Kontext aus Low-Level Kontext anhand definierter Regeln ermöglicht wird. Die Bestimmung vorliegender Schwachstellen und potentieller Ereignisse aus sicherheitsrelevantem Kontext sowie die Ableitung von Bedrohungen aus sicherheitsrelevantem Kontext, kann ebenfalls durch Context Reasoning erfolgen.

3.3.3 Kontextfehler

Kontextinformationen, die durch Sensoren ermittelt wurden, können aufgrund von Störungen oder Sensorausfällen falsch oder nicht verfügbar sein, während durch Anwender oder Anwendungsentwickler gelieferte Kontextinformationen menschlichen Fehlern oder Obsoleszenz unterliegen können (vgl. [HI04]).
Das Fehlen von Kontextinformationen kann ohne die Ergreifung von Maßnahmen dazu führen, dass Bedrohungen nicht korrekt erkannt werden und benötigte Sicherheitsmaßnahmen eventuell ausbleiben. Im Gegensatz zu fehlenden Kontextinforma-

tionen, sind falsche Kontextinformationen schwieriger oder teilweise gar nicht zu erkennen. So können aufgrund falscher Kontextinformationen Bedrohungen teilweise nicht erkannt werden, oder es können falsche Bedrohungen ermittelt werden, was zu unnötigen Sicherheitsmaßnahmen führt.
Der Einsatz von Kontextinformationen für sicherheitsrelevante Entscheidungen erfordert daher Maßnahmen zur Erkennung von Fehlern. Fehlende oder falsche Kontextinformationen müssen als solche gekennzeichnet werden und falls die Kontextinformationen nicht durch andere Maßnahmen ermittelt werden können, sollten die von der Manipulation bzw. vom Verlust betroffenen Bedrohungsmodelle als vorhanden angenommen werden. Würde ein Bedrohungsmodell nur aufgrund eines fehlenden Kontextattributs negativ ausgewertet, so soll es einen Wert erhalten, der dazu führt, dass die Bedingungen des Bedrohungsmodells positiv ausgewertet werden. So kann es zwar vorkommen, dass eine Bedrohung ermittelt wurde die nicht vorhanden ist, jedoch wird eine Bedrohung nicht aufgrund fehlender Kontextinformationen übersehen. Werden zu viele falsche bzw. fehlende Kontextinformationen ermittelt, so kann möglicherweise keine genaue Aussage mehr über vorliegende Bedrohungen erfolgen. Die Annahme aller definierten Bedrohungen würde zur Umsetzung aller modellierten Sicherheitsmaßnahmen führen und den Benutzer womöglich zu sehr einschränken.
Werden Kontextinformationen für sicherheitsrelevante Entscheidungen herangezogen, so kann dies ohne geeignete Maßnahmen zu Problemen führen. Die Manipulation von Kontext zur Verschaffung eines Vorteils wurde bereits von [BAF+03] anhand eines Spiels beobachtet. Daher ist es möglich, dass auch sicherheitsspezifischer Kontext manipuliert werden kann.

3.4 Ableitung von Sicherheitsmaßnahmen

Sicherheitsmaßnahmen sollen das Risiko von Bedrohungen verringern und dadurch zum Erreichen von Schutzzielen beitragen. Die Ableitung von Sicherheitsmaßnahmen auf Basis ermittelter Bedrohungen erfordert die Definition von Sicherheitsmaßnahmen durch Erfahrungswerte. Für jede definierte Bedrohung muss vorab mindestens eine Sicherheitsmaßnahme definiert werden, da sonst keine Sicherheitsmaßnahme abgeleitet werden kann. Teilweise ist die Definition mehrerer Sicherheitsmaßnahmen notwendig, um das Risiko einer Bedrohung auf das gewünschte Niveau zu senken.

3.4.1 Modellierung von Sicherheitsmaßnahmen

Um eine Ableitung von Sicherheitsmaßnahmen zu ermöglichen, müssen die aus Erfahrungswerten modellierten Sicherheitsmaßnahmen durch eine Ontologie, im Folgenden auch Sicherheitsontologie genannt, dargestellt werden. Die Sicherheitsonto-

logie muss auch die relevanten Schwachstellenmodelle, potentiellen Ereignisse und Bedrohungsmodelle enthalten, da so durch Context-Reasoning benötigte Sicherheitsmaßnahmen ermittelt werden können. Durch die Verwendung weiterer Kontextinformationen, können Sicherheitsmaßnahmen noch besser an den vorliegenden Kontext angepasst werden. Die Kontextinformationen müssen in diesem Fall auch durch die Sicherheitsontologie modelliert werden. Die Sicherheitsontologie sollte erweiterbar sein und Änderungen erlauben, um Schutz vor aktuellen Bedrohungen zu bieten.
Die Modellierung von Sicherheitsmaßnahmen erfolgt mithilfe von Erfahrungswerten durch die Definition von Bedingungen und Attributen. Die Bedingungen erlauben die Festlegung der Umstände, unter denen eine Sicherheitsmaßnahme ausgeführt werden soll. Wurden die Bedingungen einer modellierten Sicherheitsmaßnahmen erfüllt, dann sollte die Sicherheitsmaßnahme ausgeführt werden, um das Risiko von Bedrohungen zu verringern. Abbildung 3.3 zeigt eine beispielhaft modellierte Sicherheitsmaßnahme.

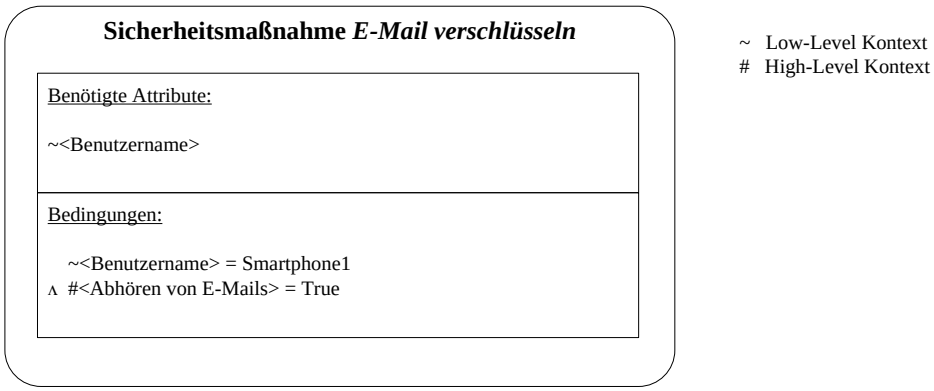

Abbildung 3.3: Beispiel eines Sicherheitsmaßnahmenmodells

Die Sicherheitsmaßnahme *E-Mail Verschlüsseln* enthält zwei Bedingungen. Das Attribut *Benutzername* muss den Wert *Smartphone1* annehmen und der Kontext *Abhören von E-Mails* muss vorliegen. Der High-Level Kontext *Abhören von E-Mails*, bei dem es sich um eine Bedrohung handelt, liegt nur vor, wenn alle seine Bedingungen erfüllt sind. Daher wird die Sicherheitsmaßnahme *E-Mail Verschlüsseln* nur ermittelt, wenn E-Mails abgehört werden könnten und das mobile Gerät dem Benutzer Smartphone1 gehört.
Da nicht davon ausgegangen werden kann, dass alle Sicherheitsmaßnahmen der Sicherheitsontologie auch der ausführenden Architekturkomponente bekannt sind, sollten auch alternative Sicherheitsmaßnahmen modelliert werden. Das Risiko der Bedrohung *Abhören von E-Mails* könnte z.B. auch durch die Nutzung einer gesicherten Verbindung verringert werden. Durch die Priorisierung von Sicherheitsmaßnahmen werden bei einer Bedrohung die abgeleiteten Sicherheitsmaßnahmen mit der höheren Priorität bevorzugt.

3.4.2 Ableitungsprozess notwendiger Sicherheitsmaßnahmen

Die Ableitung von Sicherheitsmaßnahmen erfolgt nach dem Top-Down Ansatz. Der Ablauf und die Komponenten der Ableitung sind in Abbildung 3.4 dargestellt.

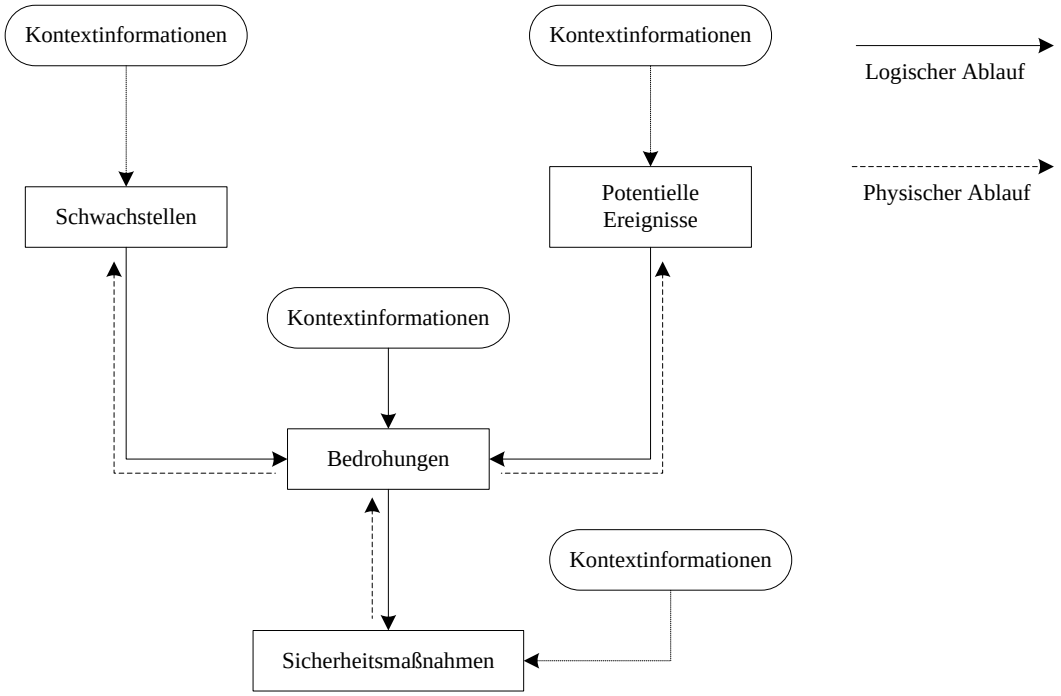

Abbildung 3.4: Ablauf der Ableitung von Sicherheitsmaßnahmen

Für jede definierte Sicherheitsmaßnahme werden die Bedingungen geprüft, die sich aus Kontextinformationen oder High-Level Kontext in Form von Bedrohungen zusammensetzen. Da die Bedrohungen zu diesem Zeitpunkt noch nicht ermittelt wurden, müssen zuerst die Bedingungen der Bedrohungen überprüft werden. Die Bedingungen der Bedrohungen können aus sicherheitsspezifischen Kontextinformationen und High-Level Kontext in Form von vorliegenden Schwachstellen und potentiellen Ereignissen bestehen. Da über die Existenz der beiden High-Level Kontexte noch keine Aussage getroffen werden kann, müssen diese erst ermittelt werden. Vorliegende Schwachstellen werden, wie auch potentielle Ereignisse, durch sicherheitsspezifische Kontextinformationen modelliert und können daher als Low-Level Kontext bezeichnet werden.
Ausgehend von den Low-Level Kontexten Schwachstelle und potentielles Ereignis, werden Bedrohungen abgeleitet. Aus den Bedrohungen ergeben sich dann die notwendigen Sicherheitsmaßnahmen.

3.5 Umsetzung von Sicherheitsmaßnahmen

Die abgeleiteten Sicherheitsmaßnahmen müssen von der umsetzenden Architekturkomponente entsprechend ihrer Definition ausgeführt werden. Die Definition der Sicherheitsmaßnahmen, die im Falle einer Bedrohung ausgeführt werden sollen, kann auf mehrere Arten erfolgen. Als einfachste Möglichkeit kann die Nutzung eines eindeutigen Bezeichners angesehen werden, durch den die Sicherheitsmaßnahmen in der Sicherheitsontologie eindeutig identifiziert werden und der die auszuführenden Aktionen repräsentiert. Die Zuordnungen der Bezeichner zu den auszuführenden Aktionen müssen auch der umsetzenden Architekturkomponente bekannt sein.
Für eine präzisere Definition der auszuführenden Aktionen eignen sich u.a. XML-Dateien, die mit den zugehörigen Sicherheitsmaßnahmen der Sicherheitsontologie verknüpft werden und die auszuführenden Aktionen spezifizieren. Eine weitere Möglichkeit der Definition von Sicherheitsmaßnahmen ist die Verwendung des auszuführenden Programmcodes. Da der Programmcode abhängig von der ihm zugrunde liegenden Plattform ist, muss er für jede Plattform angepasst werden.
Sicherheitsmaßnahmen können überall dort umgesetzt werden, wo ein Zugriff auf die Anwendungsdaten möglich ist. Einen vollständigen Zugriff auf ihre Anwendungsdaten haben üblicherweise nur die Anwendungen selbst, aber auch das Betriebssystem hat Zugriff auf die Daten der Anwendungen, jedoch fehlt diesem die Interpretationslogik für die Anwendungsdaten. Da nicht für jede Sicherheitsmaßnahme ein Zugriff auf die Anwendungsdaten nötig ist (z.B. verkleinern des Display-Sichtbarkeitswinkels), können einige Sicherheitsmaßnahmen auch ohne Interpretationslogik auf Betriebssystemebene umgesetzt werden. Einen Zugriff auf die Anwendungsdaten hat auch der Netzbetreiber, wenn Anwendungsdaten über eine Mobilfunkverbindung ausgetauscht werden, jedoch fehlt auch diesem die Interpretationslogik. Außerdem wirft die Umsetzung von Sicherheitsmaßnahmen durch den Netzbetreiber datenschutzrechtliche Bedenken auf, wenn hierzu Anwendungsdaten analysiert werden müssen. Auch Dienstanbieter haben uneingeschränkten Zugriff auf die Daten, die sie Anwendungen zur Verfügung stellen. Aus diesem Grund sind die Interpretationsvorschriften den Dienstanbietern in der Regel auch bekannt.
Welche Aktionen zur Umsetzung einer Sicherheitsmaßnahme ausgeführt werden müssen und wie diese im Detail aussehen sollen, kann im Allgemeinen nur durch Erfahrungswerte festgelegt werden und ist abhängig von der Ausführungsebene. Erfolgt die Umsetzung auf Anwendungsebene, dann obliegt es den Anwendungsentwicklern die Aktionen von Sicherheitsmaßnahmen festzulegen und den Bezeichnern der Sicherheitsontologie zuzuordnen.

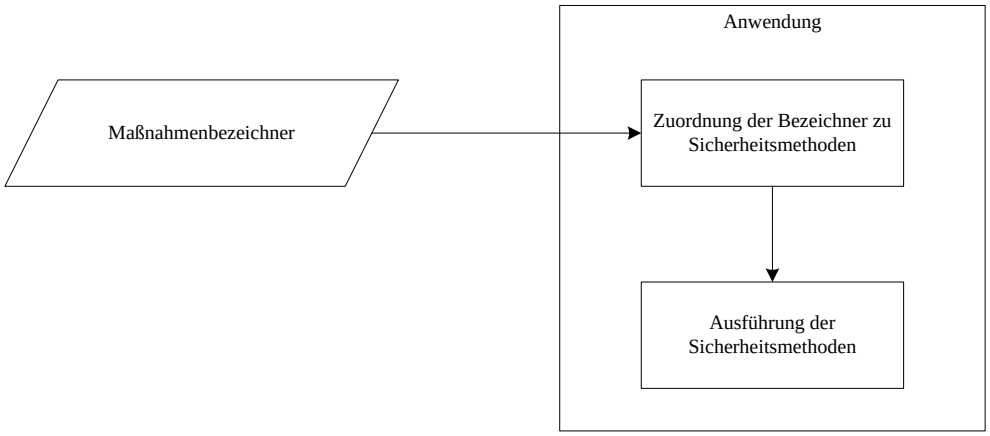

Abbildung 3.5: Ablauf der Umsetzung von Sicherheitsmaßnahmen für Anwendungen

Abbildung 3.5 stellt den Ablauf der Maßnahmenumsetzung für Anwendungen schematisch dar. Die Anwendung erhält, von der für die Ableitung von Sicherheitsmaßnahmen zuständigen Architekturkomponente, die Sicherheitsmaßnahmenbezeichner für die abgeleiteten Sicherheitsmaßnahmen. Die Zuordnung der empfangenen Bezeichner zu den auszuführenden Sicherheitsmethoden erfolgt durch eine interne Zuordnungstabelle. Wenn einem empfangenen Bezeichner Sicherheitsmethoden zugeordnet wurden, dann werden die Sicherheitsmethoden von der Anwendung ausgeführt. Ist der Bezeichner hingegen nicht in der Tabelle aufgeführt, dann wurden für diesen Bezeichner möglicherweise keine Methoden implementiert, so dass die Anwendungsdaten in der Regel nicht vor den Bedrohungen geschützt werden können. Die Umsetzung von Sicherheitsmaßnahmen auf Betriebssystemebene kann ebenfalls durch die Nutzung einer Zuordnungstabelle erfolgen, die Bezeichnern Sicherheitsmethoden zuordnet.
Umgesetzte Sicherheitsmaßnahmen sollten rückgängig gemacht werden, wenn diese nicht mehr benötigt werden, da sie keinen Mehrwert für die Sicherheit des mobilen Gerätes bieten und den Benutzer teilweise einschränken.

3.6 Race-Condition Problematik

Wenn die zeitliche Reihenfolge der ermittelten Kontextinformationen nicht eingehalten wird oder sich die abgeleiteten Sicherheitsmaßnahmen nicht auf die aktuelle Bedrohungslage des mobilen Gerätes beziehen, können Probleme auftreten, die zu einer fehlerhaften Umsetzung von Sicherheitsmaßnahme führen. Abhängig vom Ausführungsort der involvierten Architekturkomponenten, können die Race-Condition Probleme aufgrund von unterschiedlichen Übertragungslaufzeiten oder der Unterbrechung eines aktiven Threads durch den Scheduler auftreten. Das Problem kann sich sowohl bei der Umsetzung von Sicherheitsmaßnahmen, durch das Überschreiben

von gültigen Sicherheitsmaßnahmen mit älteren und ungültigen Sicherheitsmaßnahmen als auch bei der Ableitung von Sicherheitsmaßnahmen, durch das Überschreiben von aktuellen Kontextinformationen mit veralteten Kontextinformationen äußern. Werden für den vorliegenden Kontext des mobilen Geräts gültige Sicherheitsmaßnahmen aufgrund von später übermittelten veralteten Sicherheitsmaßnahmen deaktiviert, dann ist das mobile Gerät in der Regel nicht gegen Bedrohungen geschützt. Stattdessen werden nicht benötigte Sicherheitsmaßnahmen ausgeführt, die unnötig Ressourcen verbrauchen. Abbildung 3.6 visualisiert den Sachverhalt, nach dem gültige Sicherheitsmaßnahmen durch ungültige Sicherheitsmaßnahmen überschrieben werden. K_x stellt alle auf dem mobilen Gerät ermittelten Kontextinformationen dar, wobei die zeitliche Reihenfolge der Kontextermittlung durch den Index x dargestellt wird. S_x stellt die kontextabhängigen Sicherheitsmaßnahmen dar, die sich auf K_x beziehen.

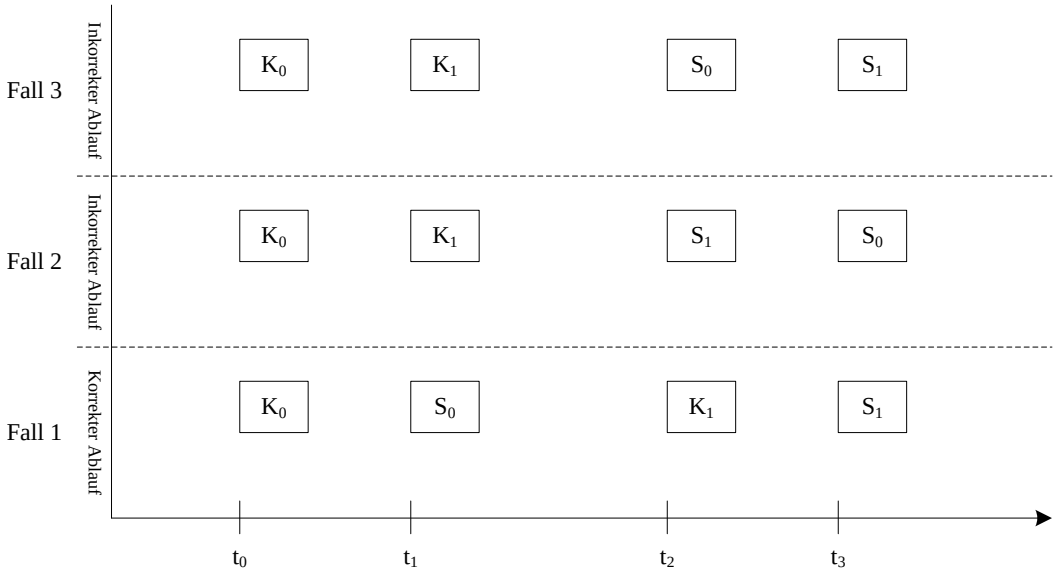

Abbildung 3.6: Race-Condition Problematik der Umsetzung

Der durch Fall 1 abgebildete Ablauf stellt eine korrekte Übermittlung von Kontextinformationen und die darauf folgende Ausführung von Sicherheitsmaßnahmen dar. K_0 stellt alle auf dem mobilen Gerät ermittelten Kontextinformationen zum Zeitpunkt t_0 dar, die an die zuständige Architekturkomponente übermittelt wurden, welche daraufhin Sicherheitsmaßnahmen ermittelt und diese an die umsetzende Architekturkomponente übermittelt. Die ermittelten Sicherheitsmaßnahmen werden zum Zeitpunkt t_1 von der umsetzenden Architekturkomponente umgesetzt. Erst nach der Umsetzung der Sicherheitsmaßnahmen werden wieder Kontextinformationen ermittelt und der Ablauf wiederholt sich.
Aufgrund unterschiedlicher Übertragungslaufzeiten oder der Unterbrechungen von Threads durch den Scheduler, können die in Fall 2 und Fall 3 dargestellten inkor-

rekten Abläufe entstehen, die zu Fehlern in der Umsetzung von Maßnahmen führen können. In Fall 2 erfolgt die Ermittlung von K_1 noch bevor S_0 umgesetzt wurde. S_1 wird vor S_0 empfangen und daher auch zuerst umgesetzt, wodurch die für den aktuellen Kontext K_1 geltenden Sicherheitsmaßnahmen durch veraltete Sicherheitsmaßnahmen überschrieben werden. Der in Fall 3 dargestellte Ablauf ist zwar nicht grundsätzlich falsch, jedoch werden zum Zeitpunkt t_2 Sicherheitsmaßnahmen für den Kontext K_0 ausgeführt, die nicht der aktuellen Bedrohungslage K_1 entsprechen und somit unnötig Ressourcen verbrauchen würden. Die Umsetzung von S_0 könnte in Fall 3 wegfallen, ohne dass sich die Bedrohungslage ändert.

Das Überschreiben von aktuellen Kontextinformationen durch veraltete Kontextinformationen stellt ein Problem für die Ableitung von Sicherheitsmaßnahmen dar, da durch falsche Kontextinformationen auch falsche Bedrohungen ermittelt werden. Die abgeleiteten Sicherheitsmaßnahmen bieten in diesem Fall keinen Schutz vor den tatsächlich vorhandenen Bedrohungen. In Abbildung 3.7 wird das Problem visualisiert, das durch die falsche zeitliche Reihenfolge bei der Übermittlung von Kontextinformationen auftreten kann.

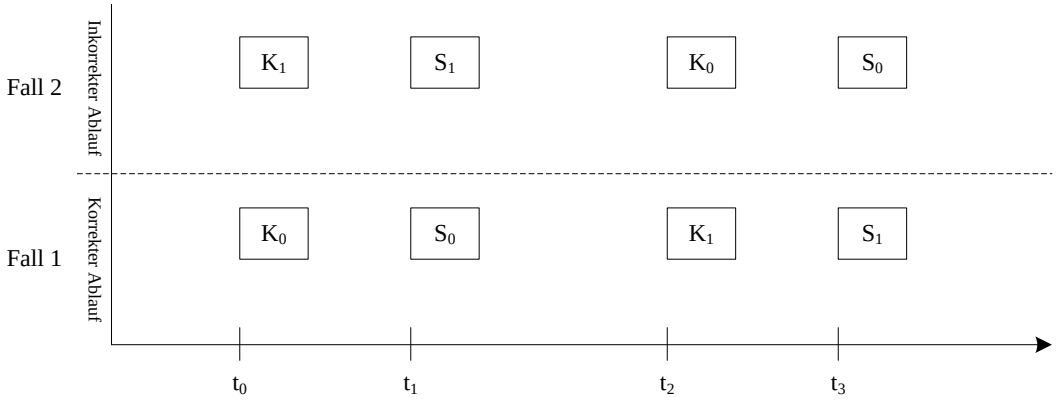

Abbildung 3.7: Race-Condition Problematik - Überschreiben von Kontextinformationen

Der durch Fall 2 dargestellte Ablauf der Kontextermittlung und Maßnahmenumsetzung visualisiert die Problematik, die auftritt, wenn aktuelle Kontextinformationen durch veraltete Kontextinformationen überschrieben werden. Zum Zeitpunkt t_0 werden die Kontextinformationen K_1 empfangen, für die in t_1 die Sicherheitsmaßnahmen S_1 ausgeführt werden. Die bereits vor dem Zeitpunkt t_0 und somit vor K_1 ermittelten Kontextinformationen K_0 werden erst zum Zeitpunkt t_2 empfangen. Das führt dazu, dass die aus den veralteten Kontextinformationen abgeleiteten Sicherheitsmaßnahmen S_0, die den aktuellen Kontextinformationen zugrunde liegenden Sicherheitsmaßnahmen S_1 überschreiben. Der korrekte Ablauf wird durch Fall 1 dargestellt.

Anhand der Fallbeispiele lässt sich erkennen, dass die Race-Condition Problematik in der Regel verhindert werden kann, wenn die Ermittlung von Kontextinformationen und die Umsetzung der daraus abgeleiteten Sicherheitsmaßnahmen zu einer Einheit zusammengefasst werden. Nach der Ermittlung der Kontextinformationen müssen diese erst an die für die Ableitung von Sicherheitsmaßnahmen zuständige Architekturkomponente übermittelt werden. Bevor weitere Kontextinformationen übermittelt werden, muss die Umsetzung der Sicherheitsmaßnahmen abgewartet werden.
Ein weiterhin bestehendes Problem stellt die Reaktion auf schnelle Kontextwechsel und die damit einhergehenden Bedrohungen dar. Wechselt der Kontext in kürzeren Intervallen als Sicherheitsmaßnahmen ermittelt und umgesetzt werden können, ist eine zeitnahe Reaktion auf Bedrohungen nicht möglich.

3.7 Zusammenfassung

Dieses Kapitel thematisierte die Ableitung kontextabhängiger Sicherheitsmaßnahmen, welche das Risiko von Bedrohungen verringern sollen. Die Bestimmung vorliegender Schwachstellen und potentieller Ereignisse sowie die Ableitung von Bedrohungen wurden ebenfalls betrachtet.
Aus der Analyse relevanter Kontextinformationen folgt, dass die vorliegenden Schwachstellen eines mobilen Gerätes aus sicherheitsspezifischen Kontextinformationen bestimmt werden können. Schwachstellenmodelle und potentielle Ereignisse erlauben die Ableitung von Bedrohungen durch Context-Reasoning, für alle in der Sicherheitsontologie definierten Bedrohungen. Die Ableitung benötigter Sicherheitsmaßnahmen erfolgt ebenfalls durch Context-Reasoning und erfordert zuvor die Modellierung der Sicherheitsmaßnahmen durch die Sicherheitsontologie. Die Umsetzung der abgeleiteten Sicherheitsmaßnahmen setzt die Verknüpfung der Maßnahmen mit auszuführenden Aktionen voraus und kann z.B. auf Betriebssystemebene oder Anwendungsebene erfolgen. Der Prozess der Ableitung und Umsetzung von Sicherheitsmaßnahmen begünstigt einige Race-Condition Probleme, auf die in Abschnitt 3.6 eingegangen wurde.

4 Architekturmodell

In diesem Kapitel werden zuerst die Anforderungen an ein Architekturmodell zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen auf Grundlage der Zielsetzung aus Kapitel 1.1 ermittelt. Aufbauend auf den Anforderungen werden zwei Architekturmodelle entworfen, die unterschiedliche Ansätze zur Umsetzung von Sicherheitsmaßnahmen verfolgen. Beide Architekturmodelle werden hinsichtlich Nutzen, Sicherheit und Realisierungsaufwand verglichen. Für ein ausgewähltes Architekturmodell werden die einzelnen Komponenten und Schnittstellen genauer betrachtet.

4.1 Anforderungen

Die Anforderungen an eine Architektur zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen für mobile Geräte leiten sich aus der Zielsetzung in Abschnitt 1.1 sowie der Betrachtung kontextabhängiger Sicherheitsmaßnahmen in Kapitel 3 ab. Demnach muss die Architektur die folgenden Anforderungen erfüllen:

- **Ermittlung von Anwendungskontext, Gerätekontext und Benutzerkontext**
  Zur Ermittlung vorliegender Schwachstellen und Ereignisse wird sicherheitsrelevanter Kontext benötigt, der den Zustand der ausgeführten Anwendungen, die Situation des Benutzers sowie den Zustand des mobilen Gerätes beschreibt. Anwendungs-, Geräte- und Benutzerkontext kann sicherheitsrelevant sein und sollte daher von der Architektur ermittelt werden.

- **Ableitung von Bedrohungen**
  Bedrohungen sollen anhand formal definierter Regeln aus vorliegenden Schwachstellen, potentiellen Ereignissen und ggf. weiteren Kontexten abgeleitet werden. Die Regeln sollen durch Ontologien ausgedrückt werden und erweiterbar sein.

- **Ableitung von Sicherheitsmaßnahmen**
  Auf Basis der erkannten Bedrohungen sollen Sicherheitsmaßnahmen ermittelt werden, die das Risiko von Bedrohungen verringern. Die Regeln zur Ableitung der Sicherheitsmaßnahmen sollen ebenfalls durch eine Sicherheitsontologie definiert werden.

- **Ableitung von High-Level Kontext**
  Zur Ermittlung einiger Bedrohungen und Sicherheitsmaßnahmen wird High-Level Kontext benötigt (z.B. Sicherheit eines WLAN Schlüssels).
- **Umsetzung von Sicherheitsmaßnahmen**
  Die ermittelten Sicherheitsmaßnahmen sollen auf Betriebssystemebene oder Anwendungsebene umgesetzt werden.
- **Ressourcenschonung**
  Aufgrund der beschränkten Ressourcen mobiler Geräte sollen Ressourcenintensive Prozesse auf einen Server ausgelagert werden. Nur die nötigsten Operationen sollen auf dem mobilen Gerät ausgeführt werden.
- **Kurze Reaktionszeiten**
  Die Umsetzung der Sicherheitsmaßnahmen soll das Risiko von Bedrohungen möglichst frühzeitig verringern, daher müssen Bedrohungen ermittelt werden sobald sie auftreten. Außerdem müssen die Sicherheitsmaßnahmen zeitnah ausgeführt werden.
- **Autonomer Ablauf**
  Der Sicherheitsdienst soll autonom sein und darf den Benutzer nicht behindern.
- **Rahmenwerk**
  Ein Rahmenwerk soll Anwendungsentwicklern die Anbindung ihrer Anwendungen an die Architektur vereinfachen.

Aufgrund der sicherheitskritischen Komponente muss die Architektur auch sicherheitsspezifische Anforderungen erfüllen:

- **Erkennung von Fehlern**
  Fehlerhafte und fehlende Kontextinformationen sollen erkannt und wenn möglich korrigiert werden. Ist dies nicht möglich, soll der Benutzer informiert werden.
- **Sichere Kommunikation**
  Die Kommunikation zwischen dem mobilen Gerät und dem Server soll so abgesichert werden, dass diese nicht manipuliert oder abgehört werden kann. Darüber hinaus muss die Authentizität des Servers sichergestellt werden.

4.2 Ansätze eigener Architekturmodelle

Ausgehend von den Anforderungen ergeben sich zwei Architekturmodelle, die sich durch die Umsetzung der Sicherheitsmaßnahmen unterscheiden. Das erste Architek-

turmodell setzt nur Sicherheitsmaßnahmen auf Betriebssystemebene um, wenn diese keine Anwendungsdaten betreffen. Andernfalls werden die Sicherheitsmaßnahmen an die zugehörige Anwendung weitergereicht. Das zweite Architekturmodell bezieht von den Anwendungen zusätzliche Informationen, welche die Umsetzung aller Sicherheitsmaßnahmen auf Betriebssystemebene ermöglichen.

4.2.1 Umsetzung von Sicherheitsmaßnahmen auf Anwendungsebene

Das im Folgenden Architekturmodell 1 genannte Modell setzt Sicherheitsmaßnahmen auf Anwendungsebene um, wenn diese Anwendungsdaten betreffen. Sicherheitsmaßnahmen die systemweite Auswirkungen haben, werden auf Betriebssystemebene umgesetzt, um gegenseitige Manipulationen zwischen den globalen Sicherheitsmaßnahmen von Anwendungen zu vermeiden. Dadurch wird z.B. verhindert, dass Anwendung A eine systemweite Sicherheitsmaßnahme aufhebt, die von Anwendung B noch benötigt wird. Im Folgenden sind die einzelnen Komponenten der Architektur sowie deren Aufgaben aufgeführt:

- Lokaler Sicherheitsdienst
  - Übermittlung von Sicherheitsmaßnahmen an Anwendungen
  - Übermittlung von Anwendungs-, Geräte- und Benutzerkontext an den Sicherheitsserver
  - Umsetzung von Sicherheitsmaßnahmen auf Betriebssystemebene
- Kontextquellen
  - Übermittlung von Kontextinformationen an den lokalen Sicherheitsdienst
- Anwendungen
  - Übermittlung von Anwendungskontext an den lokalen Sicherheitsdienst
  - Umsetzung von Sicherheitsmaßnahmen auf Anwendungsebene
- Sicherheitsserver
  - Ermittlung von Sicherheitsmaßnahmen und Übermittlung an den lokalen Sicherheitsdienst eines mobilen Gerätes

Abbildung 4.1 zeigt das Architekturmodell 1 zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen, welches Sicherheitsmaßnahmen teilweise auf Anwendungsebene umsetzt.

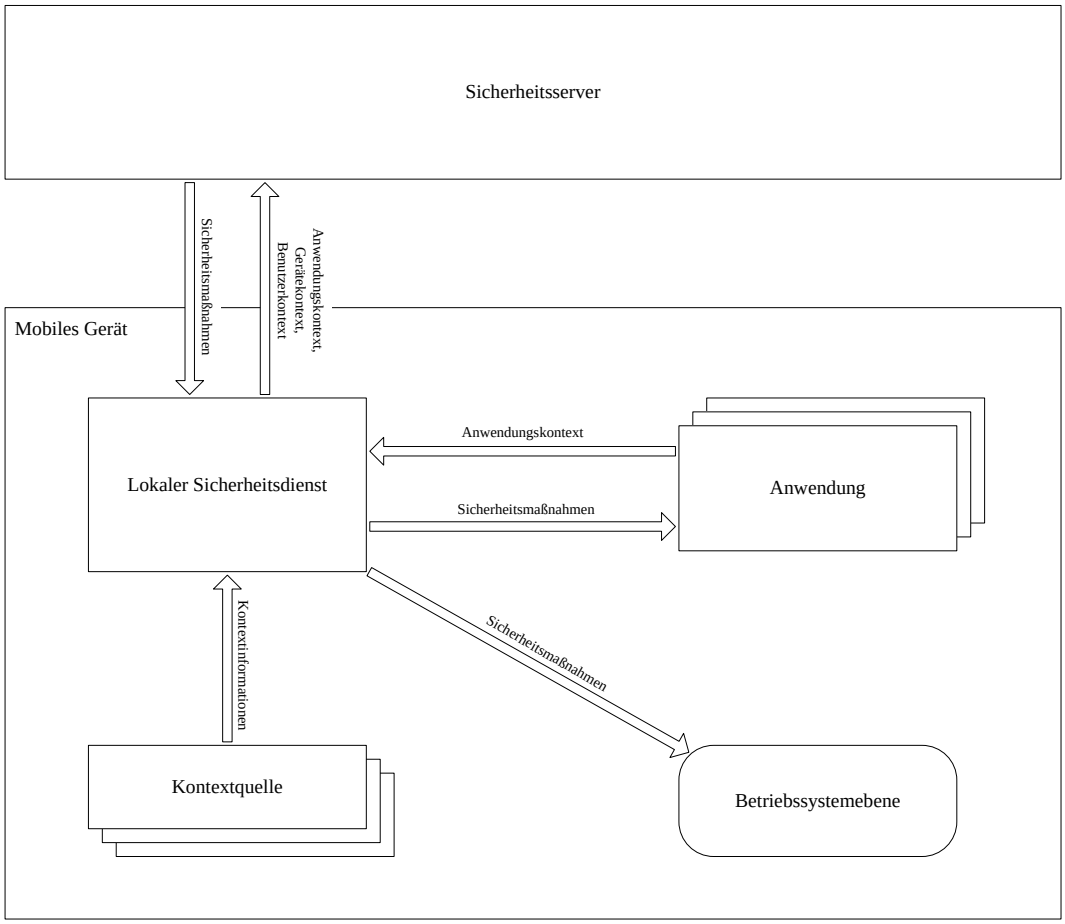

Abbildung 4.1: Architekturmodell 1

Der Prozess zur Ermittlung von Bedrohungen und Sicherheitsmaßnahmen wird entweder durch die Anwendung, indem sie dem lokalen Sicherheitsdienst ihren Anwendungskontext übermittelt oder durch den lokalen Sicherheitsdienst angestoßen. Der relevante Geräte- und Benutzerkontext wird aus den Kontextquellen des mobilen Gerätes abgefragt und zusammen mit dem Anwendungskontext an den Sicherheitsserver übermittelt. Dieser ermittelt aus den übermittelten Kontextinformationen und einer Sicherheitsontologie die bestehenden Bedrohungen für das mobile Gerät und leitet daraufhin Sicherheitsmaßnahmen ab, die das Risiko der Bedrohungen verringern sollen.
Die abgeleiteten Sicherheitsmaßnahmen werden an den lokalen Sicherheitsdienst gesendet, der diese wenn möglich auf Betriebssystemebene umsetzt (z.B. Ändern des sichtbaren Displaywinkels oder Ausweichen auf ein sicheres WLAN). Ist eine Umsetzung auf Betriebssystemebene nicht möglich, weil keine entsprechende Systemfunktion existiert oder weil Anwendungsdaten betroffen sind, deren Interpretation dem Betriebssystem nicht bekannt ist, so muss die Anwendung die Sicherheitsmaßnahmen selbst umsetzen. Die Sicherheitsmaßnahme *E-Mail Verschlüsseln* kann nicht auf Betriebssystemebene ausgeführt werden, weil dem lokalen Sicherheitsdienst in diesem

Architekturmodell die Interpretation der Anwendungsdaten nicht möglich ist und somit die Kenntnis der betroffenen E-Mail fehlt.
Das vorliegende Architekturmodell setzt voraus, dass Anwendungen die Programmierschnittstelle des lokalen Sicherheitsdienstes nutzen und ein Interesse an der Sicherheit ihrer Daten und des mobilen Gerätes haben. Malware kann durch das Architekturmodell 1 nicht ermittelt werden, da diese selbst bei einem Kommunikationszwang keinen korrekten Anwendungskontext übergeben würde.

4.2.2 Umsetzung von Sicherheitsmaßnahmen auf Betriebssystemebene

Erfolgt die Umsetzung von Sicherheitsmaßnahmen ausschließlich auf Betriebssystemebene, so müssen Anwendungen nur ihren Anwendungskontext ermitteln, die Sicherheitsmaßnahmen werden für die Anwendung transparent auf Betriebssystemebene umgesetzt. Die Ermittlung von Bedrohungen wird durch den Systemaufruf-Detektor angestoßen, wenn ein Systemaufruf erkannt wird. Gegenseitige Manipulationen zwischen den Sicherheitsmaßnahmen können nicht entstehen, da alle Sicherheitsmaßnahmen zentral verwaltet werden.
Das in Abbildung 4.2 dargestellte Architekturmodell 2 zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen wurde für die ausschließliche Umsetzung von Sicherheitsmaßnahmen auf Betriebssystemebene konzipiert. Im Folgenden sind die Komponenten des Architekturmodells sowie deren Aufgaben aufgeführt:

- Lokaler Sicherheitsdienst
  - Übermittlung von Anwendungs-, Geräte- und Benutzerkontext an den Sicherheitsserver
  - Umsetzung von Sicherheitsmaßnahmen auf Betriebssystemebene
- Kontextquellen
  - Übermittlung von Kontextinformationen an den lokalen Sicherheitsdienst
- Anwendungen
  - Übermittlung von Anwendungskontext an den lokalen Sicherheitsdienst
- Systemaufruf-Detektor
  - Erkennung von Systemaufrufen
  - Umlenkung von Systemaufrufen über lokalen Sicherheitsdienst
- Sicherheitsserver
  - Ermittlung von Sicherheitsmaßnahmen und Übermittlung an den lokalen Sicherheitsdienst eines mobilen Gerätes

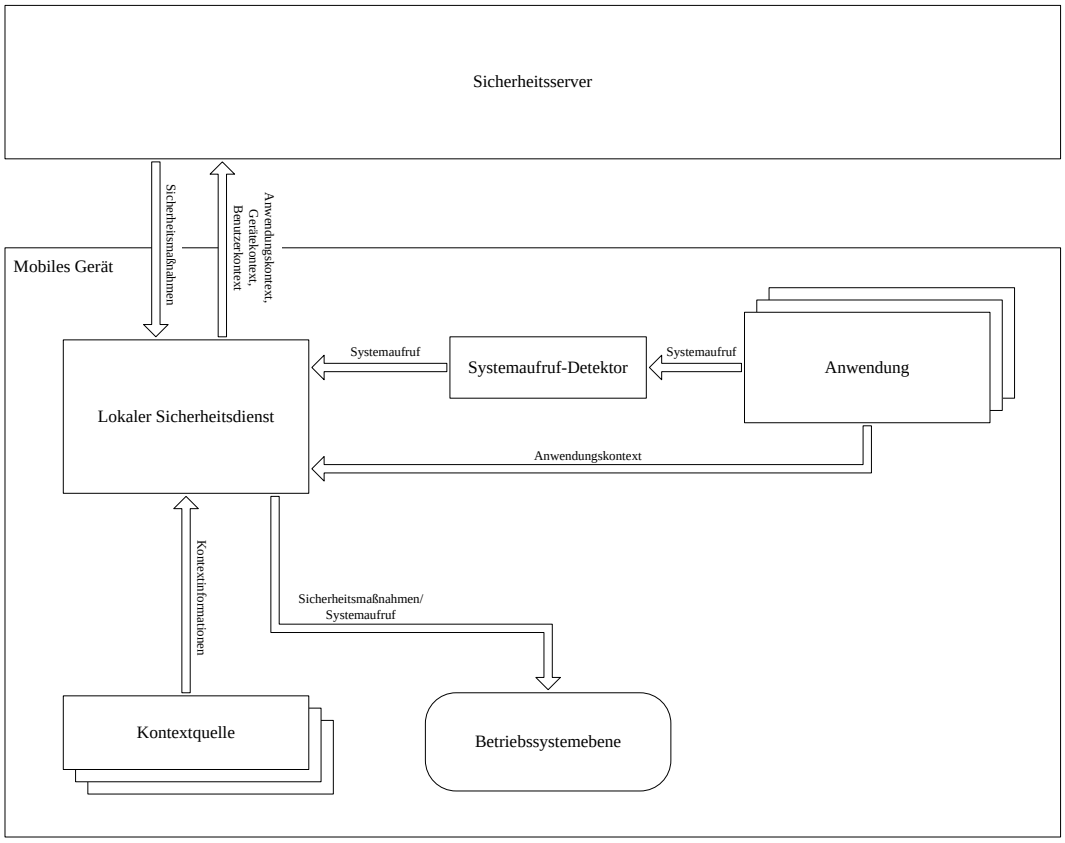

Abbildung 4.2: Architekturmodell 2

Ruft eine Anwendung eine sicherheitsrelevante Systemfunktion auf, so wird dieser Systemaufruf vom Systemaufruf-Detektor erkannt und über den lokalen Sicherheitsdienst umgelenkt. Außerdem wird der aktuelle Anwendungskontext der Anwendung bezogen, die den Systemaufruf veranlasst hat. Nachdem die Anwendung ihren aktuellen Anwendungskontext an den lokalen Sicherheitsdienst übermittelt hat, wird sie für den Zeitraum der Sicherheitsmaßnahmenermittlung pausiert. Der lokale Sicherheitsdienst ermittelt daraufhin die relevanten Geräte- und Benutzerkontextinformationen aus den vorhandenen Kontextquellen des mobilen Gerätes. Die Geräte- und Benutzerkontextinformationen werden zusammen mit dem Anwendungskontext an den Sicherheitsserver übermittelt. Der Sicherheitsserver ermittelt, wie im Fall des ersten Architekturmodells, die Bedrohungen und leitet daraus Sicherheitsmaßnahmen ab, die er an den lokalen Sicherheitsdienst sendet. Diese Sicherheitsmaßnahmen werden vom lokalen Sicherheitsdienst auf Betriebssystemebene umgesetzt. Beziehen sich die ermittelten Sicherheitsmaßnahmen auf Anwendungsdaten, wie z.B das Verschlüsseln einer E-Mail oder das Unterbinden einer E-Mail Übermittlung, so wird der Systemaufruf entsprechend der Sicherheitsmaßnahme abgeändert.
Nachdem die Sicherheitsmaßnahmen ausgeführt wurden, wird die Anwendung fortgesetzt. Die Ausführungsdauer von der Ermittlung des Anwendungskontexts bis zur Umsetzung der Sicherheitsmaßnahme, darf eine gewisse Zeit nicht überschreiten, da

sonst der Benutzer in seiner Arbeit gestört wird.

4.2.3 Auswahl eines Architekturmodells

Die in Abschnitt 4.2.1 und 4.2.2 vorgestellten Architekturmodelle stellen den Anwendungen eines mobilen Gerätes einen Sicherheitsdienst zur Verfügung, der den Anwendungen die Ermittlung von Bedrohungen und Sicherheitsmaßnahmen abnimmt. Die Anwendungen müssen jedoch ihren Anwendungskontext an den lokalen Sicherheitsdienst übermitteln, da der Anwendungskontext für die Ableitung von Sicherheitsmaßnahmen benötigt wird. Die Übermittlung des Anwendungskontexts durch die Anwendungen ist unbedingt nötig, da sowohl die Semantik, als auch der Schutzbedarf der Daten nur der Anwendung bekannt sind. Die Entscheidung welche Kontextinformationen eine Anwendung preisgibt, liegt bei beiden Architekturmodellen in der Hand der Entwickler, daher können beide Architekturmodelle keinen Schutz vor Malware bieten.
Das Architekturmodell 1 (siehe Abschnitt 4.2.1) verlangt von Anwendungsentwicklern neben der Übermittlung von Anwendungskontext die Umsetzung der Sicherheitsmaßnahmen, wenn eine Umsetzung auf Betriebssystemebene nicht möglich ist. Sicherheitsmaßnahmen die eine Manipulation der Anwendungsdaten voraussetzen (z.B. Verschlüsseln einer E-Mail), können im Architekturmodell 1 nicht auf Betriebssystemebene umgesetzt werden und sollen daher durch die Anwendungsentwickler implementiert werden. Dieser zusätzliche Aufwand entfällt durch den Einsatz des in Abschnitt 4.2.2 vorgestellten Architekturmodells 2, da alle Sicherheitsmaßnahmen auf Betriebssystemebene umgesetzt werden. Voraussetzung für die vollständige Umsetzung von Sicherheitsmaßnahmen auf Betriebssystemebene sind entweder zusätzliche Anwendungskontexte (z.B. Speicheradressen der zu schützenden Anwendungsdaten und Interpretationsvorschriften) oder die zu schützenden Anwendungsdaten selbst, die durch die jeweiligen Anwendungen bereitgestellt werden müssen.
Die zentrale Umsetzung von Sicherheitsmaßnahmen auf Betriebssystemebene erfordert weniger Aufwand für Anwendungsentwickler beim Einsatz des Architekturmodells 2, da die Anwendungen nicht an neue Sicherheitsmaßnahmen des lokalen Sicherheitsdienstes angepasst werden müssen. Der Vorteil der zentralen Umsetzung des Architekturmodells 2 und auch teilweise des Architekturmodells 1, entzieht den Entwicklern jedoch die Kontrolle über die Art der Umsetzung einer Sicherheitsmaßnahme und kann ein zentraler Angriffspunkt für Malware sein.
Die Vorteile des Architekturmodells 2 sind nur unwesentlich größer, als die Vorteile des Architekturmodells 1. Aufgrund des geringeren Implementierungsaufwands soll die Funktionalität des Konzepts zur Ableitung und Umsetzung von Sicherheitsmaßnahmen in dieser Arbeit anhand des Architekturmodells 1 nachgewiesen werden.

Das Architekturmodell 2 kann als Erweiterung des Architekturmodells 1 betrachtet werden.

4.3 Architekturkomponenten

In diesem Abschnitt werden die Komponenten des lokalen Sicherheitsdienstes und des Sicherheitsservers aus dem ausgewählten Architekturmodell 1 modelliert. Des Weiteren werden die Aufgaben der Komponenten beschrieben.

4.3.1 Lokaler Sicherheitsdienst

Der lokale Sicherheitsdienst ist eine Komponente der Architektur, die auf dem mobilen Gerät ausgeführt wird und sich um die Verwaltung der Kontextinformationen und die Umsetzung von Sicherheitsmaßnahmen kümmert. Abbildung 4.3 stellt den lokalen Sicherheitsdienst dar, der sich aus einem Dienstmanager und einem Kontextmanager zusammensetzt.

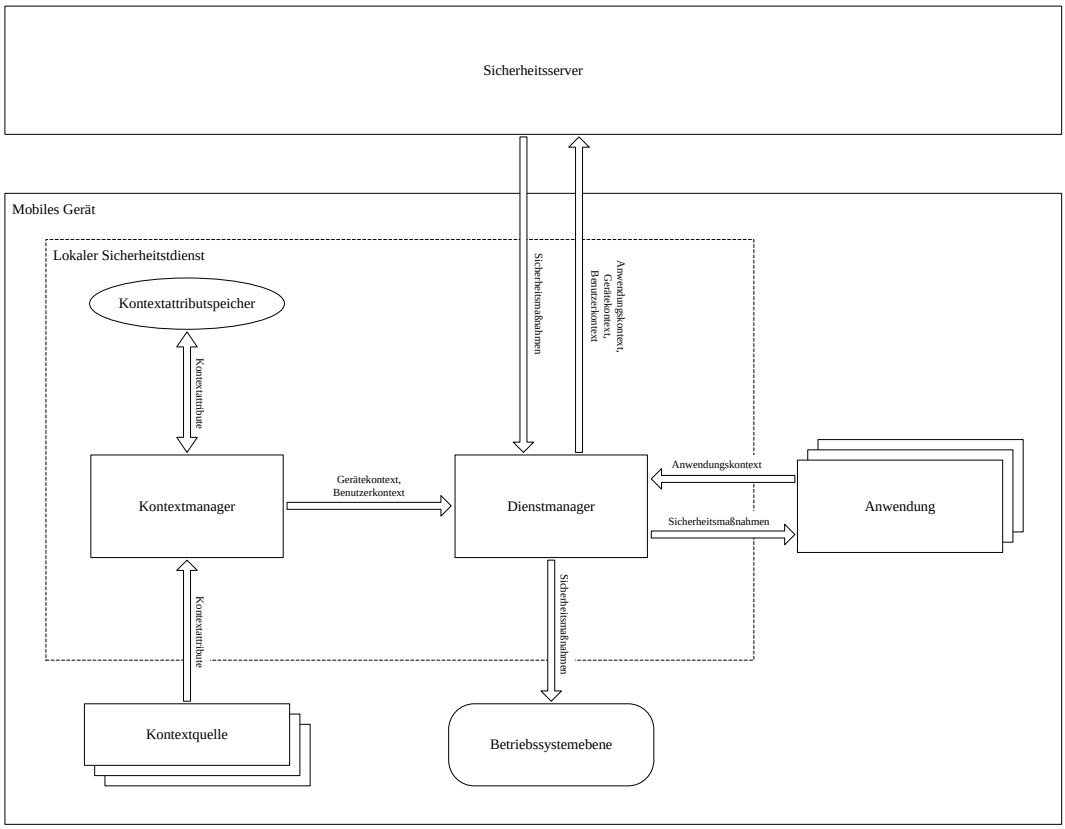

Abbildung 4.3: Lokaler Sicherheitsdienst

4.3.1.1 Dienstmanager

Der lokale Sicherheitsdienst wird vom Dienstmanager verwaltet, der die Abläufe innerhalb des lokalen Sicherheitsdienstes steuert. Als zentrale Komponente interagiert der Dienstmanager mit dem Kontextmanager, dem Sicherheitsserver und den Anwendungen und greift auf Funktionen des Betriebssystems zu. Der Dienstmanager übernimmt folgende Aufgaben:

- Empfang von Anwendungskontext
- Empfang von Geräte- und Benutzerkontext
- Übermittlung von Anwendungskontext an den Sicherheitsserver
- Übermittlung von Geräte- und Benutzerkontext an den Sicherheitsserver
- Abfrage des Sicherheitsservers nach Sicherheitsmaßnahmen
- Abfrage des Sicherheitsservers nach alternativen Sicherheitsmaßnahmen
- Zuordnung von Sicherheitsmaßnahmen zum Ausführungsziel (Anwendungen, Betriebssystem)
- Umsetzung von Sicherheitsmaßnahmen auf Betriebssystemebene
- Übermittlung von Sicherheitsmaßnahmen an eine Anwendung
- Übermittlung alternativer Sicherheitsmaßnahmen an eine Anwendung

Der Dienstmanager erhält den Geräte- und Benutzerkontext durch den Kontextmanager, wenn der Kontext sich geändert hat. Durch die Übermittlung des Geräte- und Benutzerkontexts an den Sicherheitsserver werden implizit Sicherheitsmaßnahmen für alle Anwendungen angefordert, die am lokalen Sicherheitsdienst angemeldet sind. War die Umsetzung der Sicherheitsmaßnahmen auf Betriebssystemebene nicht möglich, werden diese an die jeweiligen Anwendungen übermittelt. Da die Anwendungen in diesem Fall keine Anfragen gestellt haben, müssen diese vom Dienstmanager benachrichtigt werden.
Durch die Übermittlung von Anwendungskontext wird der Dienstmanager dazu veranlasst, nur für die betroffene Anwendung Sicherheitsmaßnahmen anzufordern. Die durch den Sicherheitsserver ermittelten Sicherheitsmaßnahmen werden, wenn möglich, auf Betriebssystemebene umgesetzt oder sonst an die betroffenen Anwendungen übergeben. War die Umsetzung seitens der Anwendung nicht möglich, dann sendet der Dienstmanager alternative Sicherheitsmaßnahmen an die Anwendung, vorausgesetzt es liegen alternative Sicherheitsmaßnahmen vor.
Die Umsetzung der Sicherheitsmaßnahmen auf Betriebssystemebene erfolgt durch den Aufruf von Systemfunktionen, die primär die Gerätehardware ansprechen. Eine

mögliche Sicherheitsmaßnahme zur Umsetzung auf Betriebssystemebene ist z.B. das Umschalten auf ein alternatives WLAN oder auf die mobile Datenverbindung.

4.3.1.2 Kontextmanager

Der Kontextmanager überprüft die Kontextinformationen aus den Kontextquellen des mobilen Gerätes, verwaltet diese und veranlasst bei Kontextänderungen die Ermittlung von Bedrohungen.
Da Bedrohungen nicht nur durch die Änderung des Anwendungskontexts entstehen können, sondern auch durch Änderungen im Benutzer- oder Gerätekontext, werden die verfügbaren Kontextquellen vom Kontextmanager in bestimmten Intervallen abgefragt und mit den gespeicherten Kontextinformationen verglichen. Die Änderungen werden in den Kontextattributspeicher übernommen und an den Dienstmanager übertragen. Eine Überprüfung des Anwendungskontexts durch den Kontextmanager ist nicht nötig, da Anwendungen ihre Kontextänderungen selbständig mitteilen.

4.3.1.3 Kontextquellen

Kontextquellen abstrahieren von den Sensoren, Systemfunktionen und anderen Quellen, die sie verwenden und ermöglichen eine einheitliche Abfrage der verfügbaren Kontextattribute. Sie ermitteln auf Anfrage des Kontextmanagers die gewünschten Kontextinformationen, soweit diese vorhanden sind und teilen die Kontextinformationen dem Kontextmanager mit.
Die Verfügbarkeit von Sensoren sowie auch die Verfügbarkeit von Systemfunktionen sind herstellerabhängig und können daher nicht spezifiziert werden. Viele mobile Geräte besitzen GPS-Sensoren, Beschleunigungssensoren, Annäherungssensoren und weitere Sensoren, die eine Ermittlung von Geräte- bzw. Benutzerkontext ermöglichen.
Systemfunktionen können genutzt werden, um Gerätekontext wie z.B. die Systemzeit oder Netzwerkinformationen zu ermitteln. Andere Quellen, wie z.B. Kontextserver, können weitere Kontextinformationen bereitstellen.

4.3.1.4 Anwendungen

Anwendungen müssen ihren aktuellen Anwendungskontext selbst an den lokalen Sicherheitsdienst übermitteln. Die Übermittlung des Anwendungskontexts erfolgt immer vor und nach einer möglicherweise kritischen Aktion. Als kritisch, im Sinne der Sicherheit, gelten alle Funktionsaufrufe, die Daten außerhalb des Prozessspeichers ablegen oder Daten entgegennehmen und in letzter Instanz Systemaufrufe verwenden.

Für die Ermittlung von Bedrohungen wird unter anderem die kritische Aktion benötigt, die von der Anwendung ausgeführt werden soll. Die Aktion muss durch einen definierten Wert dargestellt werden, der vom Sicherheitsserver interpretiert werden kann. Weitere benötigte Attribute sind abhängig von der Sicherheitsontologie.
Versendet z.B. eine E-Mail Anwendung eine E-Mail, so sollte die Anwendung vor dem Absenden der E-Mail, also vor dem Aufruf der entsprechenden Funktion, ihren Anwendungskontext an den lokalen Sicherheitsdienst senden. Dieser teilt der Anwendung mit, ob Sicherheitsmaßnahmen notwendig sind.
Sind keine Sicherheitsmaßnahmen erforderlich, so kann die Anwendung die kritische Aktion ausführen und dem lokalen Sicherheitsdienst ihren neuen Anwendungskontext mitteilen. Wurden jedoch Sicherheitsmaßnahmen übermittelt, so sollten diese von der Anwendung umgesetzt werden. Zwar sollten Sicherheitsmaßnahmen umgesetzt werden bevor Bedrohungen entstehen, jedoch darf der Benutzer nicht durch diesen Prozess eingeschränkt werden. Daher sollen alle Sicherheitsmaßnahmen in einem eigenen Thread ausgeführt werden, so dass die Anwendungen weiterhin auf Benutzereingaben reagieren können.
In einigen Fällen muss der kritische Funktionsaufruf solange zurückgehalten werden, bis die Sicherheitsmaßnahme umgesetzt wurde. Das kann gegebenenfalls zu Beeinträchtigungen für den Benutzer führen, die dem Benutzer durch die Einblendung einer entsprechenden Meldung mitgeteilt werden sollten.

Nicht nur die Änderung des Anwendungskontexts kann zu einer Bedrohung führen, sondern auch die Änderung von Benutzerkontext oder Gerätekontext. Daher muss eine Anwendung, die den lokalen Sicherheitsdienst nutzt, auch Sicherheitsmaßnahmen entgegennehmen, ohne dass sie diese direkt angefordert hat. Realisieren lässt sich diese Anforderung beispielsweise durch einen Callback-Mechanismus (vgl. [BP10]).
Hat sich der Geräte- oder Benutzerkontext geändert, so wird die Bedrohungssituation erneut vom Sicherheitsserver geprüft. Die abgeleiteten Sicherheitsmaßnahmen werden vom lokalen Sicherheitsdienst an die betroffenen Anwendungen gesendet, die innerhalb ihrer Callback-Methoden die Sicherheitsmaßnahmen umsetzen sollten.
Nicht jede Sicherheitsmaßnahme kann von einer Anwendung umgesetzt werden, weil z.B. die nötige Funktion nicht implementiert wurde. In diesem Fall muss die Anwendung dem Dienstmanager die Unmöglichkeit der Umsetzung mitteilen, woraufhin die Anwendung eine alternative Sicherheitsmaßnahme erhält, wenn der Dienstmanager über Alternativen verfügt.

4.3.2 Sicherheitsserver

Der Sicherheitsserver ist eine externe Komponente des Architekturmodells, die aus Gründen der einfacheren Administration der Sicherheitsontologie und der Ressour-

censchonung mobiler Geräte ausgelagert wurde. Der Sicherheitsserver leitet aus den Kontextinformationen, die er von den mobilen Geräten erhält, Sicherheitsmaßnahmen ab und stellt diese den zugehörigen mobilen Geräten zur Verfügung. Die Anzahl der mobilen Geräte, die der Sicherheitsserver mit Sicherheitsmaßnahmen bedienen kann, wird nicht durch die Architektur begrenzt und ist daher nur abhängig von der Serverhardware, welche ausreichend dimensioniert werden sollte, so dass die Wartezeit für ein mobiles Gerät minimal ist.
Die Architektur des Sicherheitsservers unterteilt sich in die Komponenten Dienstmanager und Reasoner. Abbildung 4.4 veranschaulicht den Aufbau des Sicherheitsservers.

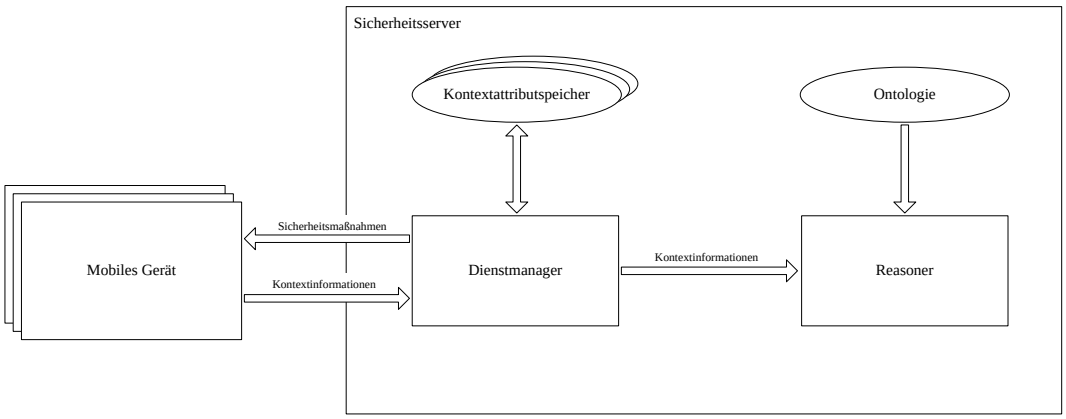

Abbildung 4.4: Sicherheitsserver

4.3.2.1 Dienstmanager

Der Dienstmanager des Sicherheitsservers ist das Gegenstück zum Dienstmanager des lokalen Sicherheitsdienstes eines mobilen Geräts. Wie auch der Dienstmanager eines mobilen Geräts verwaltet und steuert er die Abläufe. Als zentrale Komponente des Sicherheitsservers nimmt er Anfragen mobiler Geräte entgegen und beantwortet diese.
Die Anfrage eines mobilen Gerätes enthält Kontextinformationen, die der Dienstmanager im Kontextattributspeicher ablegt. Zur Bestimmung von Bedrohungen ergänzt der Dienstmanager die vom mobilen Gerät erhaltenen Kontextinformationen, um die bereits im Kontextattributspeicher vorliegenden Kontextinformationen und übermittelt diese an den Reasoner. Die vorliegenden Kontextinformationen stellen aktuelle Kontextinformationen des mobilen Gerätes dar, die bereits bei einer früheren Anfrage übermittelt wurden und sich seitdem nicht geändert haben.
Wurden durch den Reasoner Bedrohungen ermittelt, so erhält der Dienstmanager vom Reasoner Sicherheitsmaßnahmen, die der Dienstmanager an das anfragende mobile Gerät übermittelt. War die Umsetzung einer Sicherheitsmaßnahme durch ein

mobiles Gerät nicht möglich, so soll der Dienstmanager die Ermittlung alternativer Sicherheitsmaßnahmen durch den Reasoner veranlassen und diese an das anfragende mobile Gerät übermitteln.

4.3.2.2 Reasoner

Der Reasoner bedient sich einer Sicherheitsontologie und ermittelt aus den ihm zur Verfügung gestellten Kontextinformationen Bedrohungen für ein mobiles Gerät. Sind Bedrohungen vorhanden, dann bestimmt der Reasoner auf Basis der Sicherheitsontologie Sicherheitsmaßnahmen, die das Risiko der Bedrohungen minimieren sollen. Alternative Sicherheitsmaßnahmen werden, soweit definiert, ebenfalls vom Reasoner ermittelt, falls der Dienstmanager diese anfragt. Voraussetzung für die Ermittlung von Sicherheitsmaßnahmen ist deren Beschreibung durch eine Sicherheitsontologie.

4.4 Schnittstellen

Im Folgenden werden die Schnittstellen der Architekturkomponenten beschrieben, die einen Informationsaustausch zwischen den Komponenten ermöglichen, ohne dass eine Architekturkomponente die Implementierung des Kommunikationspartners kennen muss. Der Einsatz von Schnittstellen erlaubt einen Austausch von Architekturkomponenten durch Komponenten, die die definierten Schnittstellen implementieren.

4.4.1 Schnittstellen des lokalen Sicherheitsdienstes

Der lokale Sicherheitsdienst verfügt über drei Schnittstellen, jeweils eine Schnittstelle für den Informationsaustausch mit Anwendungen und Kontextquellen und eine Schnittstelle zur Steuerung der Kontextüberwachung.

4.4.1.1 Schnittstelle für Anwendungen

Die Übermittlung von Anwendungskontext und Sicherheitsmaßnahmen zwischen dem lokalen Sicherheitsdienst und den Anwendungen auf dem mobilen Gerät erfolgt über eine Schnittstelle. Da es sich bei einer Anwendung und dem lokalen Sicherheitsdienst um zwei eigenständige Prozesse handelt, wird die Kommunikation durch Interprozesskommunikation (IPC) realisiert. Die folgenden Funktionen muss die Schnittstelle des Dienstmanagers den Anwendungen bereitstellen:

- Übermittlung von Anwendungskontext an den Dienstmanager
- Übermittlung von Sicherheitsmaßnahmen an Anwendungen

  - Anfrage der Anwendung
  - Benachrichtigung durch lokalen Sicherheitsdienst

- Rückmeldung über Erfolg bzw. Misserfolg bei Umsetzung von Sicherheitsmaßnahmen

Die Übermittlung von Anwendungskontext erfolgt durch die Anwendung und zwar immer dann, wenn sich der Anwendungskontext ändert. Die Übermittlung von Sicherheitsmaßnahmen erfolgt auf Anfrage der Anwendung durch den Dienstmanager, sobald die Sicherheitsmaßnahmen vorliegen. Die Schnittstelle muss dem Dienstmanager außerdem die Möglichkeit bieten, Anwendungen Sicherheitsmaßnahmen zu übermitteln, wenn diese keine Sicherheitsmaßnahmen angefragt haben. Diese Funktion wird benötigt, falls sich der Geräte- oder Benutzerkontext ändert und diese Änderung in Verbindung mit dem aktuellen gespeicherten Anwendungskontext zu einer Bedrohung führt.
Eine Anwendung sollte dem lokalen Sicherheitsdienst mitteilen, ob sie die übermittelten Sicherheitsmaßnahmen umsetzen konnte oder ob die Umsetzung fehlgeschlagen ist. Nur so kann der lokale Sicherheitsdienst der Anwendung alternative Sicherheitsmaßnahmen übermitteln.

4.4.1.2 Schnittstelle des Kontextmanagers

Der Kontextmanager muss dem Dienstmanager über seine Schnittstelle Funktionen anbieten, durch welche der Dienstmanager die automatische Kontextüberwachung steuern kann. Die Funktionen der Schnittstelle sind im Folgenden aufgelistet:

- Starten der Kontextüberwachung
- Beenden der Kontextüberwachung
- Festlegung des Abfrageintervalls

4.4.1.3 Schnittstelle der Kontextquellen

Die Abfrage der Kontextquellen erfolgt durch den Kontextmanager des lokalen Sicherheitsdienstes. Die Anforderungen an eine Schnittstelle zur Abfrage von Kontextquellen werden von Müller [Mül10] folgendermaßen beschrieben:

- “Ermittlung aller verfügbaren Kontextattribute“
- “Abfrage aller aktuellen Attributwerte“
- “Abfrage eines spezifischen aktuellen Kontextattributwerts“

Eine Kontextquelle muss demnach eine Liste aller verfügbaren Kontextattribute bereitstellen, aus denen der lokale Sicherheitsdienst entweder einen bestimmten oder alle Attributwerte der Kontextquelle abfragen kann.

4.4.2 Schnittstellen des Sicherheitsservers

Der Sicherheitsserver besitzt zwei Schnittstellen, jeweils eine externe und eine interne. Die externe Schnittstelle realisiert die Kommunikation zwischen mobilen Geräten und dem Sicherheitsserver. Die interne Schnittstelle ermöglicht den Informationsaustausch zwischen dem Reasoner und dem Dienstmanager des Sicherheitsservers.

4.4.2.1 Externe Schnittstelle des Dienstmanagers

Die Kommunikation zwischen dem mobilen Gerät und dem Sicherheitsserver erfolgt über den Dienstmanager des lokalen Sicherheitsdienstes auf dem mobilen Gerät und den Dienstmanager auf dem Sicherheitsserver. Die Kommunikation entspricht dem Client-Server-Modell. Die Auflistung enthält alle Funktionen, die der Sicherheitsserver gegenüber einem mobilen Gerät anbieten muss:

- Übermittlung einer ID zur Identifikation des mobilen Gerätes
- Übermittlung einer ID zur Identifikation der Anfrage
- Übermittlung einer ID zur Identifikation des Anwendungskontexts
- Übermittlung einer Menge von Kontextinformationen
- Abfrage vorliegender Schwachstellen
- Abfrage ermittelter Bedrohungen
- Abfrage ermittelter Sicherheitsmaßnahmen
- Abfrage alternativer Sicherheitsmaßnahmen

Kontextinformationen können seitens des lokalen Sicherheitsdienstes auf einem mobilen Gerät als Menge an den Dienstmanager des Sicherheitsservers übertragen werden. Die Menge kann $1..n$ Kontextinformationen enthalten, die gemäß ihrer Kontextart zu kennzeichnen sind. Um das Datenvolumen gering zu halten, sollten nur Kontextinformationen übermittelt werden, die sich seit der letzten Übermittlung geändert haben. Der Sicherheitsserver hält die aktuellen Kontextinformationen der mobilen Geräte vor.
Die auf Basis der übermittelten Kontextinformationen ermittelten Bedrohungen müssen durch das zugehörige mobile Gerät abgefragt werden können. Dies gilt auch für

die abgeleiteten Sicherheitsmaßnahmen. Eine eindeutige Zuordnung der Kontextinformationen sowie der vorliegenden Schwachstellen, Bedrohungen und benötigten Sicherheitsmaßnahmen zu einem mobilen Gerät erfolgt durch einen übermittelten Identifikator.
Die Übertragung von sicherheitsrelevanten Informationen erfordert die Einhaltung der Schutzziele Authentizität und Datenintegrität. Der Nachweis über die Authentizität des Sicherheitsservers kann durch eine Signatur erbracht werden. Datenmanipulationen können in Netzen nicht verhindert werden, jedoch können Datenmanipulationen durch den Einsatz kryptografisch sicherer Hashfunktionen erkannt werden (vgl. [Eck09]). Außerdem soll das Schutzziel Vertraulichkeit durch den Einsatz von Verschlüsselungstechniken erfüllt werden, da potentielle Angreifer aus den gewonnenen Informationen einen Nutzen ziehen könnten.

4.4.2.2 Schnittstelle des Reasoners

Der Reasoner erhält vom Dienstmanager alle aktuellen vorliegenden Kontextinformationen eines Gerätes, aus denen er basierend auf der vorliegenden Sicherheitsontologie Bedrohungen ermittelt und bei vorhandenen Bedrohungen, ebenfalls aufbauend auf der Sicherheitsontologie, Sicherheitsmaßnahmen ableitet. Da nicht davon ausgegangen werden kann, dass jede Anwendung alle in der Sicherheitsontologie definierten Sicherheitsmaßnahmen umsetzen kann, sollte die Sicherheitsontologie auch alternative Sicherheitsmaßnahmen enthalten, die als solche gekennzeichnet sind und gemäß ihres Wirkungsgrades priorisiert sind. Der Reasoner muss in der Lage sein, auch alternative Sicherheitsmaßnahmen auf Anfrage des Dienstmanagers zu ermitteln. Es ergeben sich die folgenden Anforderungen an die Schnittstelle:

- Übermittlung einer Menge von Kontextinformationen
- Abfrage vorliegender Schwachstellen
- Abfrage ermittelter Bedrohungen
- Abfrage ermittelter Sicherheitsmaßnahmen
- Abfrage alternativer Sicherheitsmaßnahmen

4.5 Zusammenfassung

In diesem Kapitel wurde ein Architekturmodell zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen konzipiert. Zu Beginn des Kapitels wurden die funktionalen Anforderungen an ein Architekturmodell zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen aus der Zielsetzung dieser Arbeit

sowie der Konzeption zur Ableitung von Sicherheitsmaßnahmen aus Kapitel 3 ermittelt. Außerdem wurden sicherheitsspezifische Anforderungen formuliert, die eine Manipulation des Verarbeitungsprozesses verhindern sollen. Anhand der ermittelten Anforderungen wurde ein Architekturmodell zur Umsetzung von Sicherheitsmaßnahmen auf Anwendungsebene und ein Architekturmodell zur Umsetzung von Sicherheitsmaßnahmen auf Betriebssystemebene konzipiert. Nach einer Betrachtung von Nutzen, Sicherheit und Realisierungsaufwand fiel die Entscheidung zugunsten von Architekturmodell 1, welches eine Umsetzung von Sicherheitsmaßnahmen auf Anwendungsebene vorsieht. Zuletzt wurden die Komponenten für das ausgewählte Architekturmodell modelliert sowie die Schnittstellen spezifiziert.

5 Beispielimplementierung des Architekturmodells

In diesem Kapitel wird die prototypische Implementierung des ausgewählten Architekturmodells betrachtet. Voraussetzung für die Implementierung des Architekturmodells ist die Auswahl einer Zielplattform, die in Abschnitt 5.1 erfolgt. Die Implementierung des lokalen Sicherheitsdienstes erfolgt als Beispielimplementierung in Abschnitt 5.2. In Abschnitt 5.3, wird der Dienst des Sicherheitsservers ebenfalls in Form einer Beispielimplementierung implementiert. Die Implementierung des Architekturmodells soll zeigen, inwieweit dieses die ermittelten Anforderungen auch erfüllen kann.

5.1 Auswahl einer Zielplattform

Um die Weiterentwicklung des Architekturmodells zu vereinfachen, ist es sinnvoll, das Architekturmodell auf einer praxisrelevanten mobilen Plattform zu implementieren. Die Anzahl mobiler Plattformen ist mittlerweile so groß, dass nicht alle im Detail betrachtet werden können. Daher sollen nur einige marktführende mobile Betriebssysteme hinsichtlich ihrer Eignung als Zielplattform untersucht werden. Für die Implementierung des Architekturmodells muss die Zielplattform einige Anforderungen erfüllen, die sich aus der Konzeption des Architekturmodells ergeben und im Folgenden aufgelistet sind:

- Unterstützung von Hintergrundprozessen
- Unterstützung des Mehrprozessbetriebs
- Unterstützung der Interprozesskommunikation

Der lokale Sicherheitsdienst muss als Hintergrundprozess ausgeführt werden, da er seine Aufgaben für den Benutzer möglichst unsichtbar verrichten soll. Des Weiteren müssen sowohl der Dienstmanager als auch der Kontextmanager des lokalen Sicherheitsdienstes parallel zueinander und auch zu den Anwendungen ausgeführt werden, um ihre Aufgaben zeitnah auszuführen. Ohne den Mehrprozessbetrieb ist keine parallele Ausführung von Prozessen bzw. Threads möglich. Voraussetzung

für die Kommunikation zwischen dem Dienstmanager des lokalen Sicherheitsdienstes und den Anwendungen ist die Unterstützung der Interprozesskommunikation, da so Anwendungskontext und Sicherheitsmaßnahmen zeitnah übermittelt werden können. Für die Kommunikation zwischen dem lokalen Sicherheitsdienst und dem Sicherheitsserver müssen rechnerübergreifende Methoden der Interprozesskommunikation unterstützt werden.
Die Analysten von Gartner zählen Android, Windows Mobile/Windows Phone, Symbian, RIM und iOS zu den Marktführern im Bereich der mobilen Betriebssysteme. Nach einer Prognose [Gar11] sollte das mobile Betriebssystem Android bis zum Jahresende 2011 einen Marktanteil von 38,5 % erreichen, gefolgt von iOS mit 19,4 %, Symbian mit 19,2 %, RIM mit 13,4 % und Windows Mobile/Windows Phone mit 5,6 %.
Alle aufgeführten mobilen Betriebssysteme unterstützen die Anforderungen der Architektur und würden sich somit als Zielplattform eignen. Aufgrund des großen Marktanteils und der gut dokumentierten Programmierschnittstelle wird für die Beispielimplementierung das Open-Source[1] Betriebssystem Android verwendet. Da Android Linux-basiert und quelloffen ist, eignet es sich besonders gut um Änderungen am Betriebssystemkern vorzunehmen. Somit würde die Zielplattform auch eine Erweiterung des Architekturmodells auf das in Abschnitt 4.2.2 vorgestellte Architekturmodell 2, für welches Eingriffe in den Betriebssystemkern nötig wären, ermöglichen. Welche Änderungen am Betriebssystemkern erfolgen müssen, um Systemaufrufe abzufangen, wurde in [Bec09] für das Betriebssystem Windows Mobile gezeigt.
Aus Kompatibilitätsgründen wird für die Beispielimplementierung die vierte Ebene der Programmierschnittstelle verwendet, die zu allen Android Versionen ab Version 1.6 kompatibel ist und von 98,9 % der Android Geräte unterstützt wird (vgl. [Andd]).
Die Programmierung erfolgt in der Entwicklungsumgebung Eclipse und, wie für Android üblich, in der Programmiersprache Java.

Die Implementierung der Architektur soll als Erweiterung eines Kontext-Rahmenwerks für mobile kontextsensitive Anwendungen erfolgen. Die Nutzung eines Kontext-Rahmenwerks ist sinnvoll, da Kontext-Rahmenwerke bereits Funktionen zur Verarbeitung von Kontext bereitstellen.
In seiner Master-Thesis [Mül10] entwickelte Felix Müller ein Kontext-Rahmenwerk für mobile kontextsensitive Applikationen, nach dem Konzept einer Middleware. Die Architektur des Rahmenwerks besteht aus einem Kontextdienst, der auf dem mobilen Gerät angesiedelt ist und einem Kontextserver, der Ontologien zur Abbildung des Kontexts verwendet. Die Implementierung erfolgte prototypisch auf Apples iOS,

[1] http://source.android.com/source/licenses.html

das zum Zeitpunkt der Implementierung keinen Mehrprozessbetrieb unterstützte, so dass der Kontextdienst als statische Bibliothek implementiert wurde.
Benjamin Krumnow portierte in seiner Bachelorarbeit [Kru10] das Kontext-Rahmenwerk von [Mül10] auf das mobile Betriebssystem Android und erweiterte es um die Unterstützung des Mehrprozessbetriebs. Die Unterstützung des Mehrprozessbetriebs ermöglicht die Ausführung des Kontextdienstes als eigenständige Instanz, die parallel zu den sie nutzenden Anwendungen ausgeführt wird.

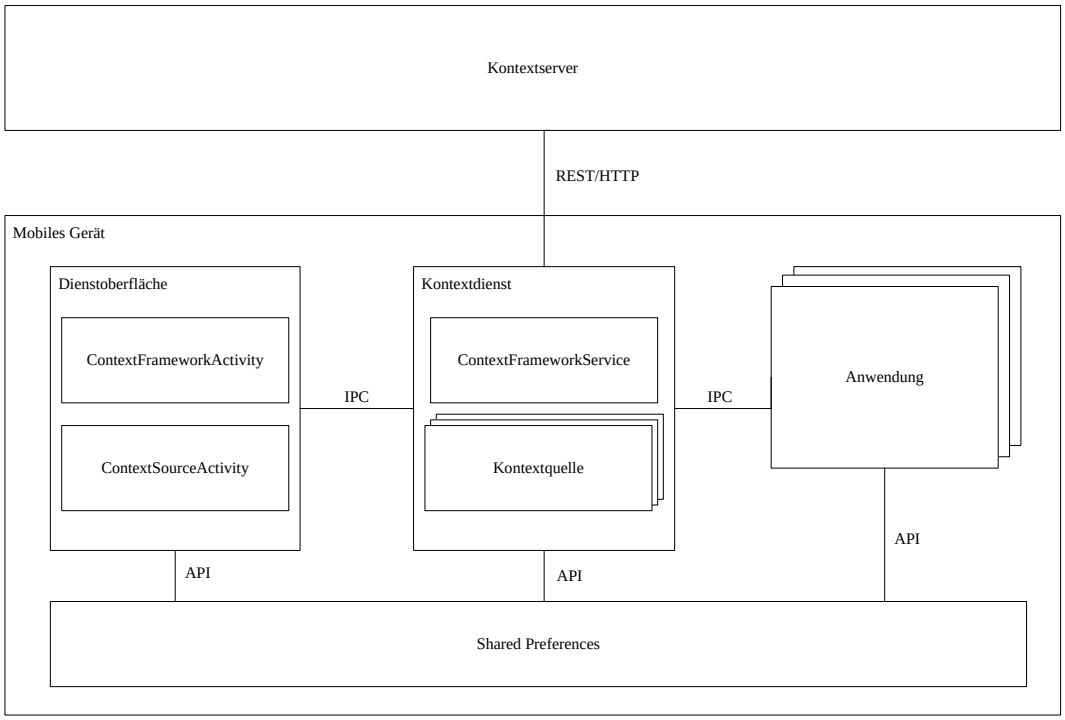

Abbildung 5.1: Architektur des Kontextdienstes (nach [Kru10])

Die in Abbildung 5.1 abgebildete Architektur setzt sich zusammen aus einem Kontextdienst, der von einem *ContextFrameworkService* verwaltet wird und eine oder mehrere Kontextquellen abfragt, einer Dienstoberfläche mit zwei Activities, einer oder mehreren Anwendungen und einem Kontextserver. Die Kommunikation mit dem Kontextdienst erfolgt sowohl für die Dienstoberfläche als auch für die Anwendungen durch Interprozesskommunikation. Außerdem können die Anwendungen und die Dienstoberfläche lesend auf die Shared Preferences[2] des Kontextdienstes zugreifen. Zur Ermittlung von Kontexten kommuniziert der Kontextdienst mit dem Kontextserver über REST/HTTP.
Durch den modularen Aufbau der Architektur aus [Kru10] können die Komponenten ausgetauscht und erweitert werden. Da die Architektur bereits einige Anforderungen aus Abschnitt 4.1 erfüllt, eignet sie sich als Grundgerüst für die Implementierung des Architekturmodells aus Abschnitt 4.2.1.

[2] Komponente zur persistenten Datenspeicherung (vgl. [Andc])

5.2 Beispielimplementierung des lokalen Sicherheitsdienstes

Der lokale Sicherheitsdienst wurde als eigenständiger Hintergrunddienst auf dem mobilen Gerät implementiert, da der Dienst für den Anwender unsichtbar im Hintergrund laufen muss. Unter Android wird ein eigenständiger Hintergrunddienst als remote Service bezeichnet. Als Grundlage für die Implementierung des lokalen Sicherheitsdienstes diente der Kontextdienst aus [Kru10]. Der lokale Sicherheitsdienst setzt sich, entsprechend dem Architekturmodell, aus den beiden Komponenten Kontextmanager und Dienstmanager zusammen.
Zur Steuerung des lokalen Sicherheitsdienstes wurde die Anwendung *ContextSecurityController* implementiert, die aus einer Activity[3] besteht und als eigenständiger Prozess über Remote Procedure Calls (RPC) mit dem Dienst kommuniziert. Der lokale Sicherheitsdienst wird automatisch gestartet, wenn eine Anwendung auf ihn zugreift. Da er als eigenständiger Prozess läuft, wird der Dienst so lange ausgeführt bis er manuell beendet wird.

5.2.1 Kontextmanager

Der Kontextmanager wurde durch die Klasse *ContextManager.java* implementiert. Er ist für die selbständige Überwachung und Übermittlung der Geräte- und Benutzerkontextinformationen, die aus den Kontextquellen abgefragt werden, an den Dienstmanager zuständig. Um sicherzustellen, dass gemäß dem Architekturmodell nur ein Kontextmanager existiert, wurde der Kontextmanager nach dem Singelton Entwurfsmuster implementiert, wodurch nur eine Instanz des Kontextmanagers erzeugt werden kann. Der Kontextmanager implementiert die in Listing 5.1 dargestellte Schnittstelle *IContextManager*, deren Methoden für die Kommunikation mit dem Dienstmanager notwendig sind.

Listing 5.1: Schnittstelle *IContextManager*

```
1 public interface IContextManager
2 {
3   /**
4    * Startet die Kontextüberwachung
5    * @return true, wenn der Thread erfolgreich gestartet wurde, sonst false
6    */
7   public boolean startMonitoring();
8
9   /**
```

[3] Komponente zur Interaktion mit Benutzern [Anda]

```
10      * Beendet die Kontextüberwachung
11      * @return true, wenn der Thread erfolgreich beendet wurde, sonst false
12      */
13     public boolean stopMonitoring();
14
15     /**
16      * Ändert das Abfrageintervall für die Kontextüberwachung
17      * @param Abfrageintervall in Millisekunden
18      */
19     public void changeMonitoringIntervall(int intervall);
20
21     /**
22      * Fragt die Attribute aller aktiven Kontextquellen ab
23      * @return Ein Array mit allen Kontextattributen
24      */
25     public Attribute[] getAllAttributeValues();
26
27     /**
28      * Fragt die Namen aller aktiven Kontextquellen ab
29      * @return Ein Array mit den Namen der Kontextquellen
30      */
31     public String[] getContextSources();
32 }
```

5.2.1.1 Kontextquellen

Der Kontextmanager verwendet zum Abfragen von Kontextquellen Funktionsaufrufe, die durch die in [Kru10] definierte Schnittstelle *IContextSource* festgelegt wurden und daher von jeder Kontextquelle implementiert werden müssen. Für Testzwecke wurden die Kontextquellen *GpsSimContextSource* und *NetworkSimContextSource* implementiert.

Die Kontextquelle *GpsSimContextSource* simuliert die Bewegung des mobilen Gerätes, indem sie die in einer Datenstruktur gespeicherten Positionsdaten auf Anfrage weitergibt. Netzwerkinformationen werden von der Kontextquelle *NetworkSimContextSource* bereitgestellt, die Attribute wie z.B. die aktive Kommunikationsart simuliert. Die Simulation eines zeitlichen Ablaufs erfolgt durch die Iteration eines Arrays. Der Grund für die Simulation der Kontextquellen liegt in der besseren Reproduzierbarkeit der aus den Kontextinformationen ermittelten Sicherheitsmaßnahmen und dient lediglich der Vereinfachung der Testläufe.

Die Attribute der Kontextquellen sind vom Datentyp *Attribute*, der in [Kru10] definiert wurde. Ein Attribut des Datentyps *Attribute* wird durch die in Tabelle 5.1 aufgeführten Parameter beschrieben.

Parameter	Beschreibung	Datentyp
type	Eindeutiger Bezeichner des Attributs	String
value	Attributwert	String
lastTimestamp	Zeitstempel der Abfrage	long
accuracy	Genauigkeit des Attributwerts	int
correctness	Korrektheit des Attributwerts	int

Tabelle 5.1: Parameter des Datentyps *Attribute*

Die Parameter *accuracy* und *correctness* wurden für die Implementierung des Architekturmodells nicht berücksichtigt. Durch die Implementierung der Schnittstelle *Parcelable*, können die Attribute durch RPC-Aufrufe übertragen werden.

5.2.1.2 Überwachung von Kontextänderungen

Um eine zeitnahe Ermittlung aktueller Bedrohungen für alle angebundenen Anwendungen zu ermöglichen, muss dem Sicherheitsserver immer der aktuelle Geräte- und Benutzerkontext vorliegen. Daher werden Änderungen im Geräte- und Benutzerkontext durch den Kontextmanager überwacht und im Fall einer Änderung an den Dienstmanager weitergeleitet, der die Kontextinformationen an den Sicherheitsserver übermittelt. Die Überwachung des Anwendungskontexts ist nicht nötig, da jede Anwendung ihren Anwendungskontext bei Änderungen über den Dienstmanager an den Sicherheitsserver sendet.
Die Überwachung und Übermittlung der Kontextänderungen wird in einem eigenen Thread ausgeführt, um Anfragen des *ContextSecurityControllers* nicht zu behindern. Das Abfrageintervall des Threads kann durch die Methode *changeMonitoringIntervall(int intervall)* festgelegt und die Überwachung durch die Methode *startMonitoring()* gestartet bzw. durch die Methode *stopMonitoring()* gestoppt werden. Die Übermittlung der Kontextänderungen an den Dienstmanager erfolgt nach dem Observer-Muster, mithilfe der Java-Klasse *Obervable*.

5.2.1.3 Minimierung des Datenvolumens

Der Kontextmanager übermittelt nicht den gesamten Geräte- und Benutzerkontext an den Dienstmanager, sondern nur die Kontextinformationen, die sich seit der letzten Übermittlung verändert haben. So kann das Datenvolumen bei der Übertragung durch den Dienstmanager an den Sicherheitsserver möglichst gering gehalten werden. Bereits vorhandene Attributwerte werden auf dem Sicherheitsserver mit den aktuellen Werten überschrieben.
Für den Vergleich werden vom Kontextmanager die Kontextinformationen der letzten Anfrage herangezogen, die in einer *Hashtable* gespeichert wurden. Die *Hashtable*

ermöglicht die Speicherung aller Attribute einer Kontextquelle unter dem eindeutigen Namen der Kontextquelle. Der Zugriff auf die in der *Hashtable* gespeicherten Attribute erfolgt im mittleren Fall mit einer Laufzeit von O(1) (vgl. [CLRS07]), wenn die Schlüssel (Namen der Kontextquellen) eindeutig sind. Abhängig von der verwendeten Hashfunktion können auch zwei unterschiedliche Attributnamen zu demselben Hashwert und somit zu einer Kollision führen. In diesem Fall müssen die Attribute, die unter dem betroffenen Schlüssel gespeichert wurden, sequentiell durchlaufen werden (vgl. [Ora]).
Die Methode *updateAttributes()* fragt alle verfügbaren Kontextquellen ab und vergleicht die Attribute mit den Attributen der letzten Abfrage. Die Änderungen gegenüber der letzten Abfrage werden durch die Observer Methode *notifyObservers(Object o)* an den Dienstmanager übertragen. Kann der Dienstmanager die Kontextinformationen nicht an den Sicherheitsserver übermitteln, so gehen diese verloren, da bei einer weiteren Abfrage wiederum nur Änderungen gegenüber der letzten Abfrage übertragen werden. Eine mögliche Maßnahme wäre z.B. die Zwischenspeicherung der Kontextinformationen durch den Kontextmanager, bis der Dienstmanager eine erfolgreiche Übertragung meldet. Der Verlust von Kontextinformationen aufgrund von Übertragungsfehlern wurde in der Beispielimplementierung nicht berücksichtigt.

5.2.2 Dienstmanager

Der Dienstmanager steuert als zentrale Komponente des lokalen Sicherheitsdienstes die Abläufe zwischen dem Kontextmanager, den Anwendungen und dem Sicherheitsserver. Um Anfragen im Hintergrund zu beantworten, wurde der Dienstmanager durch die Klasse *ServiceManager.java* als remote Service implementiert. Voraussetzung für die Erstellung eines Dienstes unter Android ist die Ableitung der Dienstklasse von der Oberklasse *Service*.
Der Dienstmanager unterscheidet zwei Arten von Abläufen bei der Bestimmung von Sicherheitsmaßnahmen. Bei Anfragen des Kontextmanagers werden Sicherheitsmaßnahmen für alle angemeldeten Anwendungen angefordert und umgesetzt. Erfolgt die Anfrage durch eine Anwendung wird nur diese bei der Ermittlung und Umsetzung von Sicherheitsmaßnahmen berücksichtigt.
Anwendungen werden durch eine Callback-Methode über auszuführende Sicherheitsmaßnahmen benachrichtigt, wenn der Dienstmanager die Sicherheitsmaßnahmen nicht auf Betriebssystemebene umsetzen konnte.

5.2.2.1 Dienstinitiierte Maßnahmenbestimmung

Der Kontextmanager ruft in festgelegten Intervallen die Attribute der verfügbaren Kontextquellen ab und übermittelt die Kontextattribute bei Kontextänderungen an

den Dienstmanager, wodurch die Ermittlung von Sicherheitsmaßnahmen initiiert wird. Um vom Kontextmanager über Kontextänderungen informiert zu werden, muss der Dienstmanager sich durch den Aufruf *contextmanager.addObserver(this)* beim Kontextmanager registrieren. Die Benachrichtigung über Kontextänderungen erfolgt nach dem Beobachter-Muster.
Der Kontextmanager ruft im Falle einer Kontextänderung die Methode *update(Observable observable, Object data)* des Dienstmanagers auf und übermittelt im Übergabeparameter data die Kontextänderungen. Alle durch den Kontextmanager ermittelten Kontextinformationen werden zur weiteren Verarbeitung an einen *RequestPool* übermittelt. Die Klasse *RequestPool* nimmt Anfragen vom Dienstmanager und Anwendungen entgegen und bearbeitet diese.
Der *RequestPool* speichert die Kontextinformationen in der Reihenfolge ihres Eintreffens in einer Liste ab und informiert den Thread der Klasse *Worker*, der die Kontextinformationen an den Sicherheitsserver übermittelt und die empfangenen Sicherheitsmaßnahmen umsetzt.
Der Einsatz eines Threads ermöglicht dem *RequestPool* weiterhin Kontextinformationen anzunehmen und parallel Sicherheitsmaßnahmen anzufordern und umzusetzen. Durch die Beachtung der zeitlichen Reihenfolge der Kontextinformationen sowie die zusammenhängende Verarbeitung der Kontextinformationen und die Umsetzung der daraus abgeleiteten Sicherheitsmaßnahmen wird verhindert, dass aktuelle Sicherheitsmaßnahmen von älteren Sicherheitsmaßnahmen überschrieben werden. Die Problematik wurde bereits in Abschnitt 3.6 betrachtet. Der *Worker-Thread* übermittelt die ihm vorliegenden Kontextinformationen durch den Aufruf der Methode *getAllSecMeasures(SecurityRequest request)*, die als Übergabeparameter einen *SecurityRequest* erwartet, an den Sicherheitsserver. Ein *SecurityRequest* enthält neben den Kontextinformationen eine eindeutige Identifikationsnummer, die bei jeder Anfrage inkrementiert wird. Die Antwort des Sicherheitsservers wird vom *Worker-Thread* in ein Array des Datentyps *SecurityMeasure* überführt.
Wenn Sicherheitsmaßnahmen ermittelt wurden, wird die Methode *executeSecMeasures(SecurityMeasure[] securityMeasures)* aufgerufen. Die Methode versucht, die Sicherheitsmaßnahmen auf Betriebssystemebene umzusetzen. War die Umsetzung einer Maßnahme nicht erfolgreich, wird die Sicherheitsmaßnahme durch den Aufruf einer Callback-Methode an die zugehörige Anwendung weitergeleitet. Das Ablaufdiagramm in Abbildung 5.2 illustriert den Ablauf der dienstinitiierten Maßnahmenbestimmung.

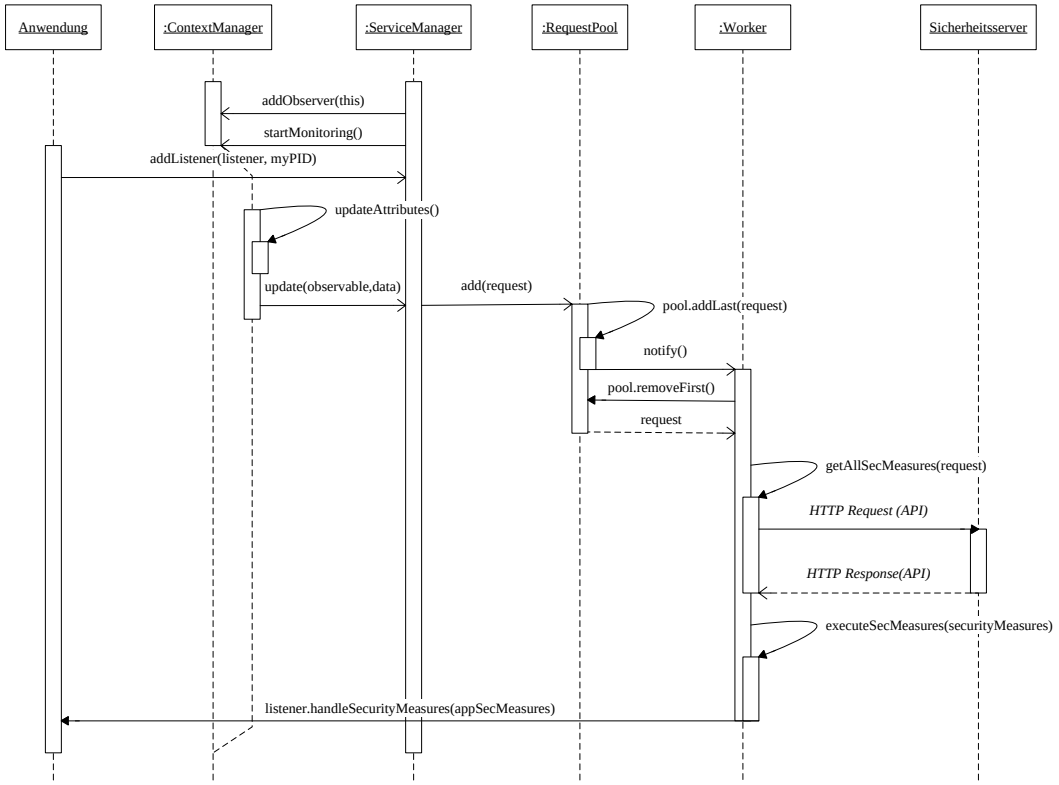

Abbildung 5.2: Ablauf der dienstinitiierten Maßnahmenbestimmung

5.2.2.2 Anwendungsinitiierte Maßnahmenbestimmung

Die Kommunikation zwischen dem Prozess des Dienstmanagers und den Prozessen der Anwendungen erfolgt durch die Nutzung des Remote Procedure Call (RPC) Mechanismus. Der Einsatz von RPC erlaubt es einem Client, die Methoden eines Servers synchron auszuführen (vgl. [EKR+05]). Bei einem synchronen Aufruf wird der Client solange blockiert, bis der Server das Ergebnis zurückgibt. Die Kommunikation zwischen den Anwendungen (Client) und dem Dienstmanager (Server) erfolgt über Stubs[4]. Da sowohl der Dienstmanager als auch die Anwendungen in unterschiedlichen Adressräumen arbeiten, können keine Referenzen übertragen werden. Daher müssen Objekte die über RPC übertragen werden die Schnittstelle *Parcelable* implementieren (vgl. [BP10]). Des Weiteren müssen Objekte durch eine AIDL-Schnittstelle spezifiziert werden, so dass sie zwischen dem Dienstmanager und den Anwendungen übertragen werden können (vgl. [Andb]).
Der Dienstmanager implementiert das Interface *IContextSecurityFramework*, dessen Methoden er den Anwendungen durch Bereitstellung des Server-Stubs zur Verfügung stellt. Die Methoden zur Bereitstellung des Server-Stubs wurden ebenso wie die Methoden zur Bereitstellung des Client-Stubs aus [Kru10] übernommen. Bevor

[4] RPC Endpunkt

Anwendungen die Methoden des Dienstmanagers aufrufen können, müssen sie einen Client-Stub erstellen und sich mit dem Dienst verbinden. Listing 5.2 zeigt den Quellcode den Anwendungen einbinden müssen, um eine Verbindung zum Dienstmanager herzustellen.

Listing 5.2: Verbindungsaufbau zum lokalen Sicherheitsdienst(nach [Kru10])

```
1  public class FrameworkConnection implements ServiceConnection
2  {
3    public void onServiceConnected(ComponentName className, IBinder binder)
4    {
5        contextSecurityFramework = IContextSecurityFramework.Stub.asInterface(binder);
6    }
7
8    public void onServiceDisconnected(ComponentName arg0)
9    {
10
11   }
12 }
13
14 // Verbindung zum Dienst aufbauen
15 private void connectToFramework()
16 {
17   contextSecurityFrameworkConnection = new FrameworkConnection();
18     Intent i = new Intent();
19     i.setClassName("de.contextsecurityframework", "de.contextsecurityframework.
           ServiceManager");
20     bindService(i, contextSecurityFrameworkConnection, Context.BIND_AUTO_CREATE);
21 }
22
23 // Verbindung beenden
24 private void disconnectFromService()
25 {
26     unbindService(contextSecurityFrameworkConnection);
27 }
```

Anwendungen die eine Verbindung zum Dienstmanager aufgebaut haben, können alle Methoden der Schnittstelle *IContextSecurityFramework* aufrufen. Listing 5.3 zeigt einen Auszug aus dem Quellcode der Schnittstelle *IContextSecurityFramework*.

Listing 5.3: Schnittstelle *IContextSecurityFramework*

```
1  interface IContextSecurityFramework
2  {
3    void sendApplicationContext(int applicationID, long messageId, in Attribute[] attributes);
4
```

```
5   void addListener(ISecurityMeasureListener listener, int pid);
6
7   void removeListener(ISecurityMeasureListener listener);
8 }
```

Anwendungen können durch den Aufruf der Methode *sendApplicationContext(int applicationID, long messageId, Attribute[] attributes)* ihren Anwendungskontext an den *RequestPool* übermitteln, der einen *ApplicationSecurityRequest* erzeugt. Die Klasse *ApplicationSecurityRequest* wurde von der Klasse *SecurityRequest* abgeleitet. Sie enthält neben dem Anwendungskontext und einer eindeutigen Identifikationsnummer die Prozess-ID der Anwendung.
Der *Worker-Thread* sendet den *ApplicationSecurityRequest* an den Sicherheitsserver, sobald dieser in der Warteschlange des *RequestPools* an erster Stelle steht. Die ermittelten Sicherheitsmaßnahmen bei der anwendungsinitiierten Maßnahmenbestimmung beziehen sich nur auf die Anwendung, die den Vorgang angestoßen hat. Die Antwort des Sicherheitsservers wird, wie bei der dienstinitiierten Maßnahmenbestimmung, durch den *Worker-Thread* in ein Array des Datentyps *SecurityMeasure* transformiert.
Enthält das Array des Datentyps *SecurityMeasure* mindestens ein Element, dann wurden vom Sicherheitsserver Maßnahmen für die Anwendung ermittelt und der *Worker-Thread* ruft die Methode *executeAppSecMeasures(SecurityMeasure[] securityMeasures)* auf. Die Methode führt alle bekannten Sicherheitsmaßnahmen, soweit möglich, auf Betriebssystemebene aus. Alle Sicherheitsmaßnahmen, die nicht auf Betriebssystemebene ausgeführt werden konnten, werden durch den Aufruf der zur Anwendung gehörenden Callback-Methode an die Anwendung übertragen. Wenn keine Sicherheitsmaßnahmen ermittelt wurden, wird ebenfalls die Methode *executeAppSecMeasures(SecurityMeasure[] securityMeasures)* aufgerufen und der Anwendung wird eine leeres Array übergeben. Das Ablaufdiagramm in Abbildung 5.3 illustriert den Ablauf der anwendungsinitiierten Maßnahmenbestimmung.

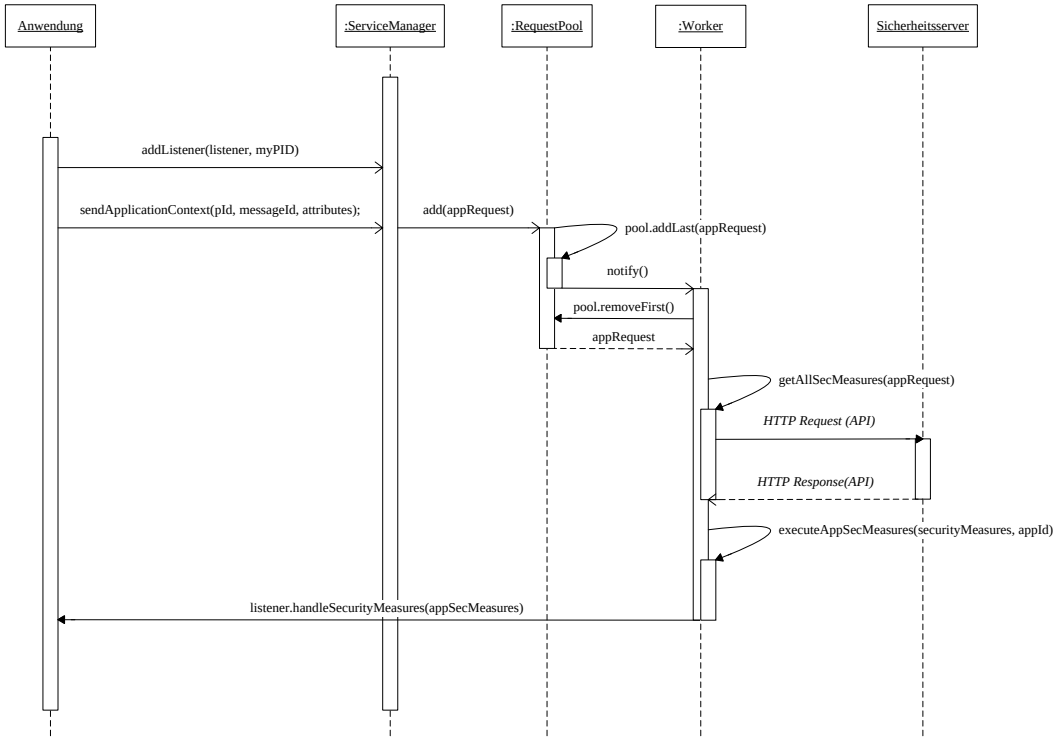

Abbildung 5.3: Ablauf der anwendungsinitiierten Maßnahmenbestimmung

5.2.2.3 Callback an Anwendungen

Der *Worker-Thread* des *RequestPools* übermittelt Sicherheitsmaßnahmen an Anwendungen, wenn diese nicht auf Betriebssystemebene ausgeführt werden konnten. Um Sicherheitsmaßnahmen empfangen zu können, müssen Anwendungen, die in Listing 5.3 aufgeführte Methode *onServiceConnected(ComponentName className, IBinder binder)* der Klasse *FrameworkConnection* wie in Listing 5.4 dargestellt, um den Aufruf der *IContextSecurityFramework* Methode *addListener(ISecurityMeasureListener listener, int pid)* erweitern.

Listing 5.4: Registrierung der Callback-Methode beim Dienstmanager

```
1  public class FrameworkConnection implements ServiceConnection
2  {
3    public void onServiceConnected(ComponentName className, IBinder binder)
4    {
5      contextSecurityFramework = IContextSecurityFramework.Stub.asInterface(binder);
6      try {
7        contextSecurityFramework.addListener(securityMeasureListener, android.os.Process.
             myPid());
8      } catch (RemoteException e) {
9        e.printStackTrace();
10     }
```

```
11      }
12
13    // ...
14  }
15  // ...
```

Außerdem müssen Anwendungen wie in Listing 5.5 aufgeführt, einen Stub der Schnittstelle *ISecurityMeasureListener* implementieren, der dem Dienstmanager durch die Methode *addListener* mitgeteilt wird. Die Stubs werden vom Dienstmanager in einer Liste gespeichert, um Sicherheitsmaßnahmen über die zugehörigen Callback-Methoden an die Anwendungen zu übermitteln.

Listing 5.5: Implementierung der Callback-Funktionalität

```
1  private ISecurityMeasureListener.Stub securityMeasureListener = new ISecurityMeasureListener
       .Stub()
2  {
3    @Override
4    public void handleSecurityMeasures(SecurityMeasure[] securityMeasure)
5    {
6      // Anwendungsinterne Verarbeitung der Sicherheitsmaßnahmen
7    }
8  };
```

Die Methode *handleSecurityMeasures(SecurityMeasure[] securityMeasure)* ist eine Callback-Methode, die vom *Worker-Thread* des *RequestPools* aufgerufen wird, wenn Sicherheitsmaßnahmen für eine Anwendung vorliegen. Wurde zuvor Anwendungskontext an den Dienstmanager übermittelt, so wird die Callback-Methode auch aufgerufen, wenn keine Sicherheitsmaßnahmen für die Anwendung vorliegen. Dadurch wird den Anwendungsentwicklern die Möglichkeit geboten, ihre Anwendung bis zum Eintreffen der Sicherheitsmaßnahmen zu blockieren bzw. nicht mehr benötigte Sicherheitsmaßnahmen zu deaktivieren. War die Maßnahmenbestimmung dienstinitiiert, wird die Callback-Methode der Anwendung ebenfalls aufgerufen, um die Anwendung darüber zu informieren, dass keine Sicherheitsmaßnahmen für die Anwendung vorliegen. Wird die Callback-Funktionalität nicht mehr benötigt, jedoch spätestens beim Beenden der Anwendung, sollte sie durch den Aufruf der Methode *removeListener(ISecurityMeasureListener listener)* deaktiviert werden.

5.2.2.4 Umsetzung von Sicherheitsmaßnahmen auf Betriebssystemebene

Die Ausführung von Sicherheitsmaßnahmen auf Betriebssystemebene erfolgt sowohl für die vom Dienstmanager ausgehenden *SecurityRequests* als auch für die von Anwendungen ausgehenden *ApplicationSecurityRequests* durch den *Worker-Thread* des

RequestPools. Für jede vom Sicherheitsserver ermittelte Sicherheitsmaßnahme wird die statische Methode *executeSecurityMeasure(String securityMeasure, boolean activate)* der Klasse *SecurityActions* aufgerufen. Die Methode *executeSecurityMeasure* erwartet einen Sicherheitsmaßnahmenbezeichner vom Datentyp *String* und den Wert *true*, um die Sicherheitsmaßnahme auf Betriebssystemebene auszuführen. Wird anstatt *true false* übergeben, dann wird die entsprechende Sicherheitsmaßnahme deaktiviert.
Die Methode *executeSecurityMeasure* überprüft anhand des übermittelten *Strings*, ob eine Sicherheitsmaßnahmen-Methode zur Ausführung der überreichten Sicherheitsmaßnahme implementiert wurde. Wurde eine Sicherheitsmaßnahmen-Methode gefunden, wird der Aufruf an die zugehörige Methode weitergeleitet und der Rückgabewert, der abhängig von der Ausführung dieser Methode ist, wird zurückgegeben. Wurde hingegen keine Methode gefunden, wird *false* zurückgegeben.
Die auf Betriebssystemebene aktiven Sicherheitsmaßnahmen werden vom *RequestPool* in einer Tabelle des Datentyps *SecurityMeasureTable* verwaltet, um die auf Betriebssystemebene nicht mehr benötigten Sicherheitsmaßnahmen deaktivieren zu können. Handelt es sich bei der Anfrage um einen *ApplicationSecurityRequest*, werden die auf eine Anwendung bezogenen Sicherheitsmaßnahmen, soweit diese auf Betriebssystemebene ausführbar waren, der Tabelle hinzugefügt. Bei *SecurityRequests* werden sowohl neue Sicherheitsmaßnahmen der Tabelle hinzugefügt als auch nicht mehr benötigte Sicherheitsmaßnahmen aus der Tabelle gelöscht.

5.2.2.5 Kommunikation mit dem Sicherheitsserver

Die Kommunikation zwischen dem lokalen Sicherheitsdienst und dem Sicherheitsserver erfolgt über HTTP nach dem REST Programmierparadigma. Das Prinzip der Kommunikation wurde aus [Mül10] übernommen. Anfragen an den Sicherheitsserver werden vom *Worker-Thread* des *RequestPools*, durch die HTTP-Request Methode *GET* an den Sicherheitsserver gesendet. Die Antworten auf die Anfragen werden im XML-Format an den Aufrufer übertragen. Bevor eine Anfrage über HTTP in Form eines *SecurityRequests* für alle Anwendungen oder eines *ApplicationSecurityRequests* für eine bestimmte Anwendung an den Sicherheitsserver gesendet werden kann, muss diese in URL-Form gebracht werden. Die URL für eine Anwendungsbezogene Anfrage enthält die in Tabelle 5.2 aufgeführten Parameter.

Die Parameter *aid*, *mid* und *attributes* entsprechen den Parametern *appId*, *messageId* und *attributes* der Klasse *ApplicationSecurityRequest*. Der Parameter *user* ermöglicht die Umsetzung benutzer- und gerätespezifischer Sicherheitsmaßnahmen. Listing 5.6 zeigt die Struktur der URL für eine anwendungsbezogene Anfrage anhand eines Beispiels.

Parameter	Beschreibung
user	Name des Benutzers/Gerätes für den/das Sicherheitsmaßnahmen ermittelt werden sollen
aid	Eindeutige Identifikationsnummer der Anwendung
mid	In Bezug auf aid eindeutige Identifikationsnummer der Anfrage
type	Kontexttyp. Zur Anfrage von Sicherheitsmaßnahmen: type = SecurityMeasure
attributes	Anwendungskontext der Anwendung

Tabelle 5.2: Parameter einer anwendungsbezogenen Anfrage an den Sicherheitsserver

Listing 5.6: Beispiel-URL einer anwendungsbezogenen Anfrage

```
1 http://10.1.1.100:4567/contexts.xml?user=Smartphone1&aid=254&mid=2&type=
      SecurityMeasure&attributes={'event'=>'sendMail','mailtype'=>'business'}
```

Eine mögliche Antwort auf die Anfrage ist in Listing 5.7 dargestellt. Der aktuelle Geräte- und Benutzerkontext lag dem Sicherheitsserver bereits aus einer früheren Anfrage vor.

Listing 5.7: Beispiel-Antwort auf eine anwendungsbezogenen Anfrage

```
1  <?xml version="1.0"?>
2  <contexts type="array">
3    <context>
4      <timestamp type="datetime">Mon Nov 21 06:15:23 +0100 2011</timestamp>
5      <type>SecurityMeasure</type>
6      <name>encryptConnection</name>
7      <aid>254</aid>
8      <mid>2</mid>
9    </context>
10   <context>
11     <timestamp type="datetime">Mon Nov 21 06:15:23 +0100 2011</timestamp>
12     <type>SecurityMeasure</type>
13     <name>useCellular</name>
14     <aid>254</aid>
15     <mid>2</mid>
16   </context>
17 </contexts>
```

Das Feld *name* enthält eine Sicherheitsmaßnahme, die aus dem Anwendungskontext und den auf dem Sicherheitsserver vorliegenden Geräte- und Benutzerkontexten ermittelt wurde. Alternative Sicherheitsmaßnahmen wurden nicht implementiert, daher sind alle Sicherheitsmaßnahmen, die für eine Anwendung ermittelt werden, auch obligatorisch.
Die URL für eine dienstinitiierte Anfrage enthält die in Tabelle 5.3 aufgeführten

Parameter.

Parameter	Beschreibung
user	Name des Benutzers/Gerätes für den/das Sicherheitsmaßnahmen ermittelt werden sollen
mid	In Bezug auf den Dienstmanager eindeutige Identifikationsnummer der Anfrage
type	Kontexttyp. Zur Anfrage von Sicherheitsmaßnahmen: type = SecurityMeasure
attributes	Kontextänderungen (Geräte- und Benutzerkontext aus den verfügbaren Kontextquellen)

Tabelle 5.3: Parameter einer dienstinitiierten Anfrage an den Sicherheitsserver

Die Parameter *mid* und *attributes* entsprechen den Parametern *messageId* und *attributes* der Klasse *SecurityRequest*. Der Parameter *appId* wird nicht benötigt, da die Anfrage sich auf alle Anwendungen bezieht, die den lokalen Sicherheitsdienst nutzen.
Listing 5.8 zeigt die Struktur der URL für eine dienstinitiierte Anfrage an einem Beispiel.

Listing 5.8: Beispiel-URL einer dienstinitiierten Anfrage

```
1 http://10.1.1.100:4567/contexts.xml?user=Smartphone1&mid=6&type=SecurityMeasure&
    attributes={'communicationType'=>'wlan','wlanMode'=>'open','wlanSecurityType'=>'
    wep'}
```

Eine mögliche Antwort auf die Anfrage ist in Listing 5.9 dargestellt. Die zur Ermittlung einbezogenen Anwendungskontexte lagen dem Sicherheitsserver bereits aus früheren anwendungsinitiierten Anfragen vor.

Listing 5.9: Beispiel-Antwort auf eine dienstinitiierte Anfrage

```
1  <?xml version="1.0"?>
2  <contexts type="array">
3    <context>
4      <timestamp type="datetime">Mon Nov 21 06:38:56 +0100 2011</timestamp>
5      <type>SecurityMeasure</type>
6      <name>encryptConnection</name>
7      <aid>254</aid>
8      <mid>6</mid>
9    </context>
10   <context>
11     <timestamp type="datetime">Mon Nov 21 06:38:56 +0100 2011</timestamp>
12     <type>SecurityMeasure</type>
13     <name>denyMailAccess</name>
```

```
14      <aid>262</aid>
15      <mid>6</mid>
16    </context>
17  </contexts>
```

Die Antwort auf eine dienstinitiierte Anfrage enthält Sicherheitsmaßnahmen für alle beim lokalen Sicherheitsdienst registrierten Anwendungen, deren Anwendungskontext dem Sicherheitsserver vorliegt und für die auch Sicherheitsmaßnahmen ermittelt wurden. Die Zuordnung der Sicherheitsmaßnahmen zu den Anwendungen erfolgt über den Parameter *aid*, der die Prozess-ID einer Anwendung repräsentiert.

5.3 Beispielimplementierung des Sicherheitsservers

Für den Sicherheitsserver wurde der Kontextdienst aus [Mül10] entsprechend den Anforderungen zur Ermittlung von Sicherheitsmaßnahmen, sowie unter Berücksichtigung des Architekturmodells angepasst und erweitert. Die Erweiterung des Kontextdienstes zum Sicherheitsdienst erfolgte in der Programmiersprache Ruby, die [Mül10] bereits für die Implementierung des Kontextdienstes verwendet hat. Voraussetzung für die Ausführung des Sicherheitsdienstes als Webanwendung ist die Installation des *Sinatra Frameworks* als Laufzeitumgebung.
Vorhandene Schwachstellen, potentielle Ereignisse, Bedrohungen und Sicherheitsmaßnahmen wurden durch eine Sicherheitsontologie modelliert, auf die der Sicherheitsdienst über *SPARQL-Abfragen* zugreift. Abbildung 5.4 zeigt die Beziehungen zwischen den einzelnen Komponenten des Sicherheitsservers.

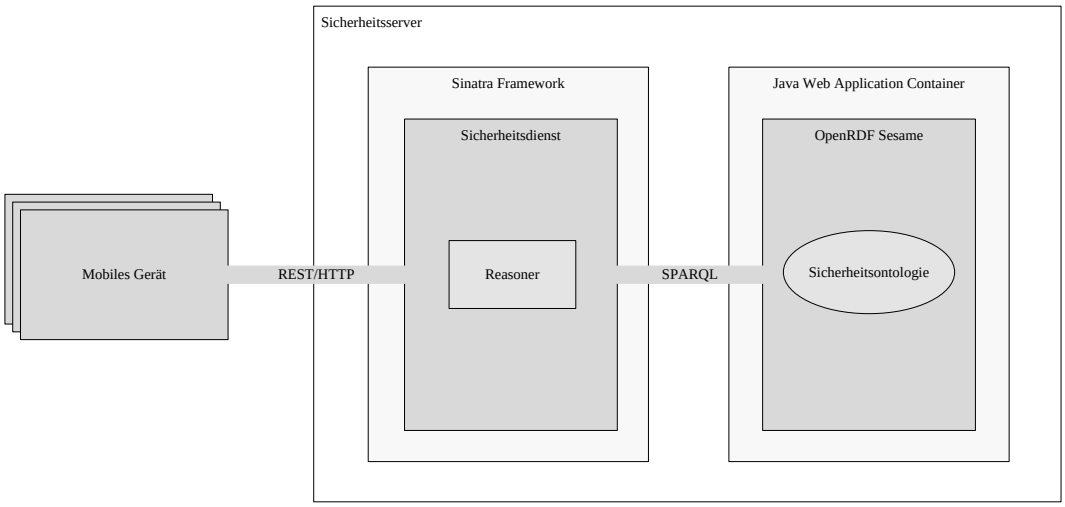

Abbildung 5.4: Komponenten des Sicherheitsservers (nach [Mül10])

5.3.1 Sicherheitsontologie

Aufbauend auf den Überlegungen aus Kapitel 3 wurde eine Beispielontologie erstellt, die eine Ableitung von Sicherheitsmaßnahmen für bestehende Bedrohungen ermöglicht, soweit diese auch von der Ontologie abgedeckt werden. Schwachstellen (*Vulnerability*), potentielle Ereignisse (*Event*), Bedrohungen (*Threat*), Anwendungskontexte (*ApplicationContext*), Gerätekontexte (*DeviceContext*) und Sicherheitsmaßnahmen (*SecurityMeasure*) wurden in der Beispielontologie als Subklassen der Oberklasse *Context* modelliert, da alle Subklassen auch einen Kontext darstellen. Ein Auszug aus der Beispielontologie ist in Abbildung 5.5 dargestellt.

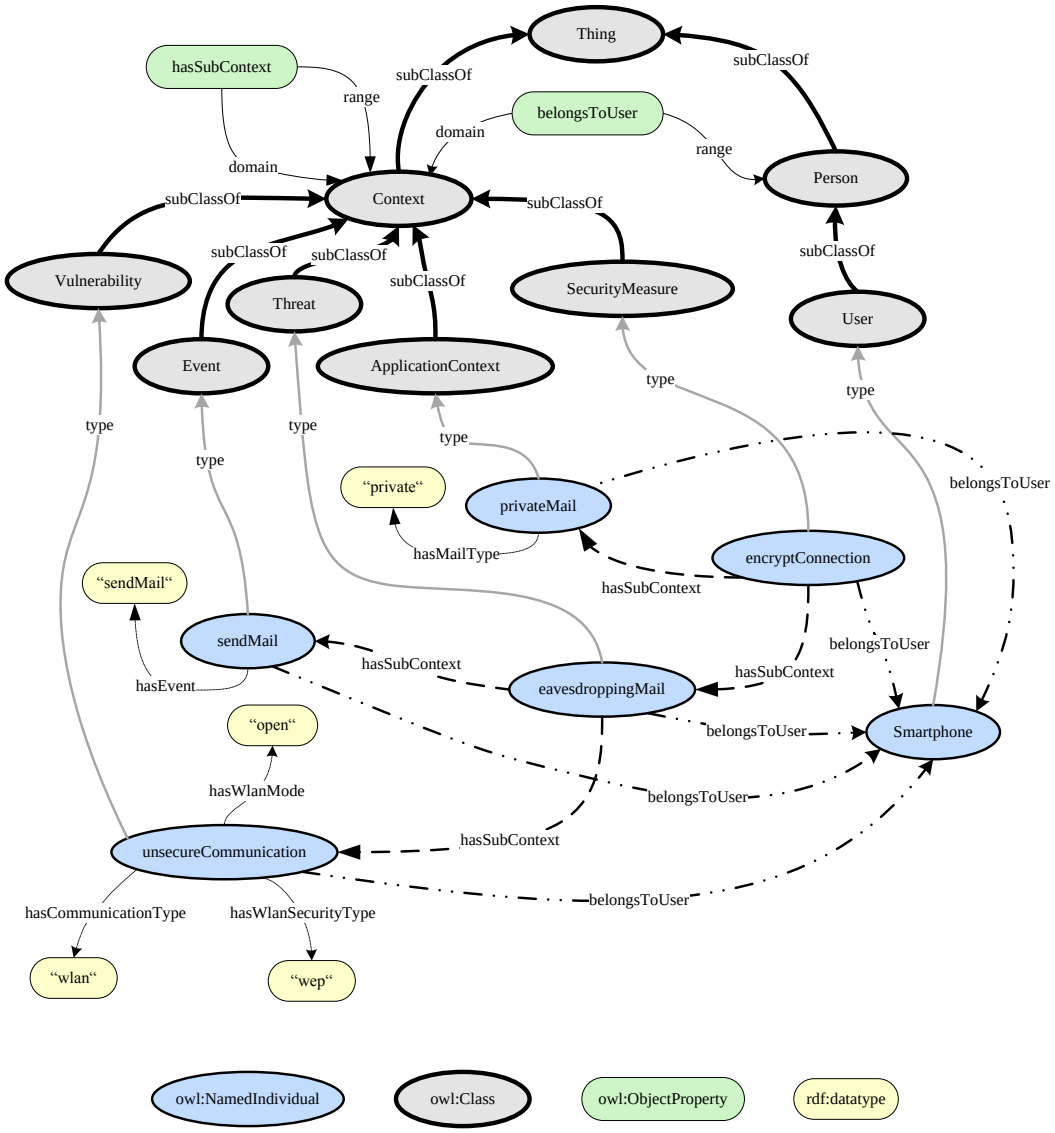

Abbildung 5.5: Auszug aus der Beispielontologie

Die Objekteigenschaften *belongsToUser* und *hasSubContext* des Typs *owl:ObjectProperty* der Oberklasse *Context* werden von den Subklassen geerbt. Die Instanzen

der Subklassen (*owl:NamedIndividual*) enthalten zusätzlich Dateneigenschaften des Typs *owl:DatatypeProperty*, mit denen Regeln definiert werden können. Die Werte, die Dateneigenschaften annehmen müssen um den Regeln zu entsprechen, werden durch den Typ *rdf:datatype* festgelegt.
Bei den Instanzen *unsecureCommunication*, *privateMail* und *sendMail* handelt es sich um Low-Level Kontexte. Low-Level Kontexte werden ermittelt, indem die übermittelten Attribute nach den festgelegten Regeln abgeglichen werden. Sind die Bedingungen erfüllt, dann liegt der Kontext vor. High-Level Kontexte wie *eavesdroppingMail* und *encryptConnection* setzen durch ihre Bedingungen die Existenz anderer Kontexte voraus. Die Bedrohung *eavesdroppingMail* liegt nur dann vor, wenn das Ereignis *sendMail* und die Schwachstelle *unsecureCommunication* vorliegen. Die Schwachstelle *unsecureCommunication* liegt nur vor, wenn als Kommunikationstyp *wlan* übermittelt wurde, der Sicherheitstyp *wep* ist, es sich um ein offenes Netz handelt und der Benutzer *Smartphone1* die Attribute übermittelt hat. Die im Beispielauszug der Sicherheitsontologie modellierten Regeln sind vollständig in Tabelle 5.4 aufgeführt.

Kontext	Regeln
#unsecureCommunication type: Vulnerability	hasCommunicationType: “wlan“ ∧ hasWlanSecurityType: “wep“ ∧ hasWlanMode: “open“ ∧ belongsToUser: #Smartphone1
#sendMail type: Event	hasEvent: “sendMail“ ∧ belongsToUser: #Smartphone1
#privateMail type: ApplicationContext	hasMailType: “private“ ∧ belongsToUser: #Smartphone1
#eavesdroppingMail type: Threat	hasSubContext: #sendMail ∧ hasSubContext: #unsecureCommunication ∧ belongsToUser: #Smartphone1
#encryptConnection type: SecurityMeasure	hasSubContext: #privateMail ∧ hasSubContext: #eavesdroppingMail ∧ belongsToUser: #Smartphone1

Tabelle 5.4: Regeln der Beispielontologie

Die Sicherheitsontologie wird durch den *Triple-Store OpenRDF Sesame* bereitgestellt, der als Java Webanwendung in einem *Apache Tomcat Servlet Container* ausgeführt wird. Ein *Triple-Store* ist eine Datenbank zur Speicherung von RDF-Daten, die als Subjekt-Prädikat-Objekt Beziehung dargestellt werden (vgl. [Rus11]). Der *OpenRDF Sesame Triple-Store* ermöglicht den Zugriff auf die Sicherheitsontologie durch *SPARQL-Abfragen. SPARQL (Protocol and RDF Query Language)* ist eine Abfragesprache und ein Protokoll zur Abfrage von RDF-Daten (vgl. [PS08], [CFT08]).

5.3.2 Dienstmanager des Sicherheitsservers

Zur Umsetzung des Sicherheitsdienstes auf dem Sicherheitsserver wurde der Kontextdienst von [Mül10] entsprechend den Anforderungen der Architektur angepasst. Wie im Architekturmodell vorgesehen, verwaltet der Dienstmanager, als zentrale Komponente die Abläufe des Dienstes. In Abbildung 5.6 sind die Klassen des Sicherheitsdienstes dargestellt, die in den Prozess der Maßnahmenermittlung involviert sind. Die Klassenstruktur wurde aus [Mül10] übernommen.

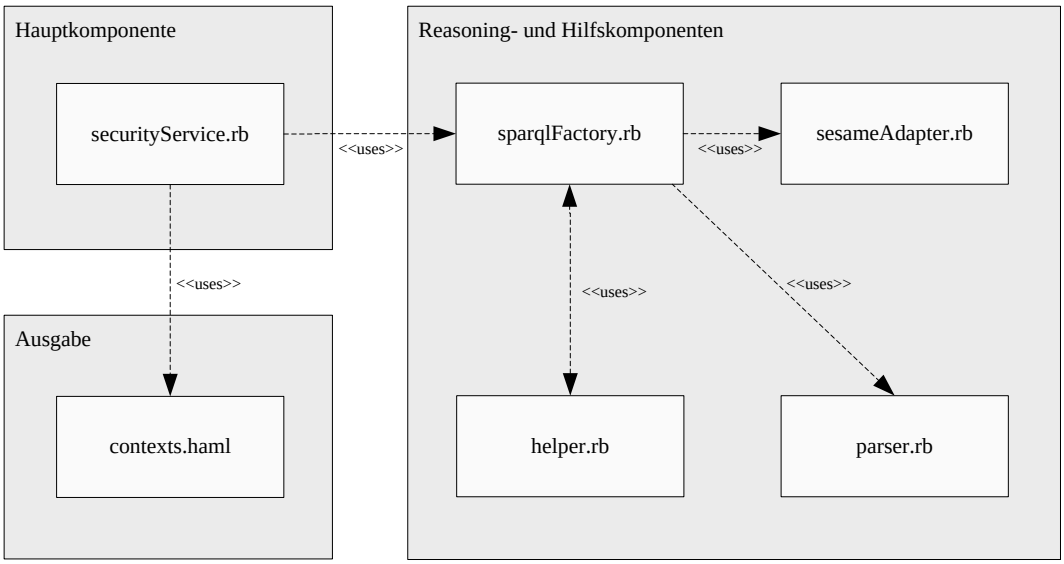

Abbildung 5.6: Klassendiagramm des Sicherheitsservers (nach [Mül10])

Die Hauptkomponente des Sicherheitsservers ist die Klasse *securityService*, die den Dienstmanager implementiert. Die Kontextinformationen der mobilen Geräte werden durch den Dienstmanager entgegengenommen und von diesem verarbeitet. Zur Bestimmung von vorliegenden Schwachstellen, potentiellen Ereignissen, Bedrohungen und Sicherheitsmaßnahmen werden die Kontextinformationen an die Klasse *sparqlFactory* übergeben, die unter Zuhilfenahme der Klasse *sesameAdapter*, Anfragen an den *Triple-Store* stellt. Die Antworten des *Triple-Stores* werden von der Klasse *parser* übersetzt, so dass diese weiter verarbeitet werden können. Die Klasse *helper* filtert die Ergebnisse entsprechend den in der Sicherheitsontologie definierten Regeln. Die implementierte Architektur enthält keine als Reasoner gekennzeichnete Komponente, stattdessen wird die Funktionalität des Reasoners durch die Hilfskomponenten realisiert.
Um Sicherheitsmaßnahmen zuverlässig zu bestimmen, müssen dem Sicherheitsserver alle auf dem mobilen Gerät ermittelten Kontextinformationen vorliegen. Da der lokale Sicherheitsdienst nur Kontextänderungen übermittelt, werden die Kontextinformationen vom Sicherheitsserver gespeichert und aktualisiert. Alle Kontextinformationen werden benutzerbasiert gespeichert, so dass Sicherheitsmaßnahmen durch

eine Sicherheitsontologie dargestellt und benutzerspezifisch ermittelt werden können. Der Sicherheitsdienst unterscheidet zwischen dienstinitiierten und anwendungsinitiierten Anfragen mobiler Geräte. Die Unterscheidung ist nötig, da bei dienstinitiierten Anfragen für alle Anwendungen des mobilen Gerätes, Sicherheitsmaßnahmen ermittelt werden, wohingegen bei anwendungsinitiierten Anfragen nur für die anfragende Anwendung Sicherheitsmaßnahmen ermittelt werden. Die Unterscheidung erfolgt anhand des Parameters *aid* in der URL, der bei anwendungsinitiierte Anfragen die Prozess-ID enthält, während dienstinitiierte Anfragen den Parameter *aid* nicht übermitteln.
Handelt es sich um eine anwendungsinitiierten Anfrage, wird der übermittelte Anwendungskontext der Anwendung vom Sicherheitsdienst gespeichert. Zur Ermittlung der anwendungsspezifischen Sicherheitsmaßnahmen werden alle für den Benutzer vorliegenden Anwendungskontexte der Anwendung sowie alle für den Benutzer vorliegenden Geräte- und Benutzerkontextinformationen herangezogen. Die ermittelten Sicherheitsmaßnahmen beziehen sich nur auf die anfragende Anwendung. Bei dienstinitiierten Anfragen wird nur Geräte- und Benutzerkontext übertragen und gespeichert. Zur Ableitung von Sicherheitsmaßnahmen werden bei dienstinitiierten Anfragen, die für den Benutzer vorliegenden Anwendungskontexte aller Anwendungen sowie die vorliegenden Geräte- und Benutzerkontextinformationen einbezogen. Die Sicherheitsmaßnahmen können in diesem Fall alle Anwendungen betreffen, für die Anwendungskontext vorgelegen hat. Daher werden die Sicherheitsmaßnahmen mit der Prozess-ID der zugehörigen Anwendung gekennzeichnet. Die Rückgabe der Sicherheitsmaßnahmen erfolgt im XML-Format durch Verwendung der im HAML-Format[5] definierten Schablone *contexts*.

5.3.3 Reasoner

Die Ermittlung von Schwachstellen sowie die Ableitung von Bedrohungen und Sicherheitsmaßnahmen erfolgen durch den Reasoner. Der Reasoner aus [Mül10] wurde nicht als eigenständige Komponente, sondern durch Hilfskomponenten realisiert. Für die Ermittlung von High-Level Kontext mussten die für das Reasoning zuständigen Komponenten erweitert werden, um die Ableitung von Bedrohungen aus vorliegenden Schwachstellen und potentiellen Ereignissen sowie die Ableitung von Sicherheitsmaßnahmen zu realisieren.
Regeln zur Bestimmung von vorhandenen Schwachstellen, potentiellen Ereignissen und Bedrohungen werden durch die Verknüpfung von Attributen, Schwachstellen und potentiellen Ereignissen durch den logischen Operator $\wedge$ definiert. Sollen weitere logische Operatoren unterstützt werden, dann muss der Reasoner entsprechend

[5] HTML Abstraction Markup Language

erweitert werden.
Um die Existenz eines der definierten Elemente zu überprüfen, müssen abhängig von der Definition, zuerst die Subkontexte Anwendungskontext, Gerätekontext, Schwachstelle, potentielles Ereignis und Bedrohung ermittelt werden, die für die Existenz des Elements vorausgesetzt werden. Eine direkte Aussage über die Existenz eines Kontextes kann nur gemacht werden, wenn dieser nur aus Attributen besteht, daher werden High-Level Kontexte solange rekursiv durchlaufen, bis nur noch Attribute vorliegen.
Bei Sicherheitsmaßnahmen handelt es sich wie bei Bedrohungen, um High-Level Kontext. Zur Ermittlung von Sicherheitsmaßnahmen werden alle Subkontexte, dazu gehören in erster Instanz Bedrohungen, Schwachstellen und potentielle Ereignisse, rekursiv ermittelt. Liegen alle definierten Subkontexte vor, dann sind die Bedingungen der Sicherheitsmaßnahme erfüllt, und die Maßnahme muss auf dem mobilen Gerät ausgeführt werden.
Die Ermittlung vorliegender Kontexte erfolgt durch die Methode *filterResults(results, attributes, predicates)* der Klasse *helper.rb*, indem geprüft wird, ob die übergebenen Attribute(*attributes*), die in der Ontologie definierten Bedingungen (*predicates*) erfüllen. Der Übergabeparameter *results* enthält die Zielkontexte (z.B. alle Sicherheitsmaßnahmen) die überprüft werden sollen. Um auch High-Level Kontext zu verarbeiten, wurde die Methode *filterResults* erweitert. Handelt es sich bei einem der Zielkontexte um High-Level Kontext, wie es z.B. bei Sicherheitsmaßnahmen der Fall ist, wird die Methode *reasonSubContext(node, attributes)* aufgerufen, die in Listing 5.10 abgebildet ist.

Listing 5.10: Reasoning-Methode des Sicherheitsservers

```
1  def self.reasonSubContext(node, attributes)
2    res = 0
3
4    #Bestimme Kontexttyp des Knotens
5    type = SparqlFactory.getType(node)
6    d = type[0]
7
8    #Wenn es sich um High-Level Kontext handelt
9    if (d == "Threat" || d == "SecurityMeasure")
10     #Ermittle alle Elemente (Variablen und Subkontexte)
11     children = SparqlFactory.getSubContexts(node)
12
13     #Für jedes Kind rekursiv Subkontext ermitteln
14     children.each do |child|
15       puts child
16       h = child.split('#').last
17       res = res + reasonSubContext(h, attributes)
```

```
18      end
19    else
20      #Attribute Überprüfen
21      res = checkVar(node, attributes)
22    end
23
24    #Wenn res negativ, dann liegt der Kontext nicht vor
25    return res
26  end
```

Die Methode *reasonSubContext* erwartet den Startknoten sowie die zu vergleichenden Attribute als Übergabeparameter. Sie prüft rekursiv für den Startknoten, ob alle Subkontexte vorliegen und somit auch, ob der Kontext des Startknotens vorliegt. Dazu steigt die Methode solange rekursiv über alle Subkontexte ab, bis es sich nicht mehr um High-Level Kontext, sondern Low-Level Kontext mit Attributen handelt. Die Überprüfung der Attribute erfolgt durch die Methode *checkVar(node, attributes)*, die null zurück gibt, wenn ein Attributwert die Bedingungen (*predicates*) erfüllt, sonst wird ein negativer Wert zurückgegeben. Wurde für eines der Subkontext-Attribute des Startknotens ein negativer Wert ermittelt, so liegt auch der Kontext des Startknotens nicht vor.

5.4 Zusammenfassung

In diesem Kapitel wurde das in Kapitel 4 entworfene Architekturmodell 1 prototypisch als Erweiterung eines Kontext-Rahmenwerks implementiert. Zu den implementierten Komponenten gehören der lokale Sicherheitsdienst für die Plattform Android, der auf dem mobilen Gerät angesiedelt ist sowie der Sicherheitsdienst des Sicherheitsservers in Form einer Ruby Webapplikation. Die Implementierung umfasst auch eine Beispielontologie, die das benötigte Wissen zur Ableitung von Sicherheitsmaßnahmen bereitstellt.
Die Beispielimplementierung zeigt, dass sich das Architekturmodell umsetzen lässt und die festgelegten Anforderungen, soweit diese implementiert wurden, auch erfüllt. Auf die Umsetzung der in Abschnitt 4.1 definierten sicherheitsspezifischen Anforderungen wurde verzichtet, da diese für die Beurteilung der grundlegenden Realisierbarkeit nicht relevant sind. Abgesehen von der Anforderung *kurze Reaktionszeiten*, über die mangels repräsentativer Tests keine Aussage gemacht werden kann, werden alle sonstigen Anforderungen erfüllt.

6 Zusammenfassung und Ausblick

In diesem Kapitel werden die Ergebnisse, die im Rahmen dieser Diplomarbeit erarbeitet wurden zusammengefasst. Außerdem wird ein Ausblick auf mögliche Anknüpfungspunkte für weiterführende Arbeiten gegeben.

6.1 Zusammenfassung

Das Ziel dieser Arbeit bestand in der Konzeption einer Architektur für mobile Geräte, welche Anwendungen, abhängig von den Bedrohungen, Sicherheitsmaßnahmen bereitstellt. Die Sicherheitsmaßnahmen sollen an den Kontext angepasst sein und dadurch das Risiko von Bedrohungen verringern. Vor der Konzeption der Architektur wurde ermittelt, ob Sicherheitsmaßnahmen kontextadaptiv bereitgestellt werden können und welche Schritte hierzu notwendig sind.
Die Ableitung benötigter Sicherheitsmaßnahmen setzt die Erkennung von Bedrohungen voraus, bevor diese durch geeignete Sicherheitsmaßnahmen beseitigt werden können. Bedrohungen können durch Context-Reasoning ermittelt werden, wenn die Bedingungen, unter denen die Bedrohungen auftreten, zuvor durch eine Ontologie modelliert wurden. Die Zuordnung der Sicherheitsmaßnahmen zu den Bedrohungen, deren Risiko verringert werden soll, erfordert auch eine Modellierung der Sicherheitsmaßnahmen durch eine Ontologie. Die Ableitung benötigter Sicherheitsmaßnahmen erfolgt ebenfalls durch Context-Reasoning, indem für alle modellierten Sicherheitsmaßnahmen geprüft wird, ob die ihnen zugeordneten Bedrohungen vorliegen. Die Umsetzung der abgeleiteten Sicherheitsmaßnahmen kann auf Anwendungsebene oder Betriebssystemebene erfolgen. Zuvor muss den Bezeichnern, die sich aus der Ableitung der Sicherheitsmaßnahmen ergeben, eine Menge von Aktionen zugeordnet werden, deren Ausführung das Risiko der Bedrohungen verringert. In Zusammenhang mit der Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen kann es zu Race-Conditions kommen, die größtenteils vermieden werden können, wenn die Ableitung und Umsetzung von Sicherheitsmaßnahmen als logische Einheit zusammengefasst werden.
Aufbauend auf den Grundlagen und der Betrachtung kontextabhängiger Sicherheitsmaßnahmen, wurden Anforderungen an eine Architektur zur Ableitung und Umsetzung kontextabhängiger Sicherheitsmaßnahmen für mobile Geräte zusammenge-

stellt. Aufbauend auf den Anforderungen wurde ein Architekturmodell konzipiert, welches die Umsetzung kontextabhängiger Sicherheitsmaßnahmen auf Anwendungsebene vorsieht, wenn die Sicherheitsmaßnahmen Anwendungsdaten betreffen. Sicherheitsmaßnahmen die systemweite Auswirkungen haben, werden auf Betriebssystemebene umgesetzt. Das Architekturmodell untergliedert sich in die Komponenten lokaler Sicherheitsdienst, der auf dem mobilen Gerät angesiedelt ist und den Sicherheitsserver, der ausgelagert wurde.
Die prototypische Implementierung des Architekturmodells erfolgte für das mobile Betriebssystem Android unter Erweiterung des Kontext-Rahmenwerks von [Mül10], das von [Kru10] auf Android portiert wurde. Anwendungen können ihren Anwendungskontext an den lokalen Sicherheitsdienst übermitteln, wenn sie zuvor eine Verbindung, durch den Aufruf einer vom Rahmenwerk bereitgestellten Methode, aufgebaut haben. Die Bereitstellung von Sicherheitsmaßnahmen auf Anwendungsebene erfordert außerdem die Registrierung einer von der Anwendung implementierten Callback-Funktion beim lokalen Sicherheitsdienst.
Die Implementierung der Architektur zeigt, dass Sicherheitsmaßnahmen kontextadaptiv bereitgestellt werden können, wenn diese zuvor durch eine Ontologie modelliert und durch Context-Reasoning aus den vorliegenden Kontextinformationen abgeleitet wurden. In diesem Fall können die Sicherheitsmaßnahmen den Schutz bieten, der auch benötigt wird; vorausgesetzt die Ontologie enthält alle relevanten Bedrohungen und Sicherheitsmaßnahmen. Eine Reaktion auf Bedrohungen, die zuvor nicht modelliert wurden, ist nach diesem Ansatz nicht möglich. Fehler in der Modellierung von Bedrohungen oder Sicherheitsmaßnahmen, können ebenso wie fehlende oder fehlerhafte Kontextinformationen, zu inkorrekten oder ausbleibenden Sicherheitsmaßnahmen führen.

6.2 Ausblick

Die vorliegende Arbeit zeigt, dass Sicherheitsmaßnahmen kontextadaptiv und damit abhängig von den Bedrohungen, bereitgestellt werden können. Dadurch werden, abhängig von der zugrunde liegenden Ontologie, nur die Sicherheitsmaßnahmen ausgeführt, die nach den ermittelten Bedrohungen notwendig sind. Gegenüber dem Ansatz immobiler Systeme, kann in der Regel bei weniger Ressourcenverbrauch, der gleiche Schutzbedarf erfüllt werden. Der Einsatz kontextadaptiver Sicherheitsmaßnahmen kann auch auf immobilen Systemen von Vorteil sein, da dadurch Energiekosten eingespart werden können. Wie groß der Vorteil der geringeren Prozessorbelastung und des daraus folgenden geringeren Energieverbrauchs gegenüber dem immobilen Ansatz auf mobilen Geräten in der Praxis ist, wurde in dieser Arbeit nicht untersucht. Die manuelle Modellierung von Bedrohungsmodellen und Sicherheitsmaßnahmen

durch eine Ontologie ist anfällig für Fehler, die aufgrund von Unachtsamkeit oder Unwissenheit entstehen können. Fehler, die bei der Erstellung der Ontologie auftreten, können zu falschen bzw. ausbleibenden Sicherheitsmaßnahmen führen. Die Bereitstellung spezieller Modellierungswerkzeuge, die den Prozess der Modellierung von Bedrohungen und Sicherheitsmaßnahmen unterstützen, könnte die Fehlerrate verringern.
Ansätze zur Erkennung bzw. Behebung von Kontextfehlern wurden in dieser Arbeit nicht betrachtet. Da falsche Kontextinformationen auch zu falschen oder ausbleibenden Sicherheitsmaßnahmen führen können, ist eine Auseinandersetzung mit dieser Thematik erforderlich. Ohne zusätzliche Maßnahmen zur Erkennung oder Behebung von Kontextfehlern könnte das Sicherheitssystem durch die Manipulation der Kontextinformationen umgangen werden.
Auf dem Übertragungsweg zwischen lokalem Sicherheitsdienst und Sicherheitsserver, können Kontextinformationen bzw. Sicherheitsmaßnahmen abhanden kommen. Der lokale Sicherheitsdienst, der die Ableitung benötigter Sicherheitsmaßnahmen initiiert, erhält in beiden Fällen keine Sicherheitsmaßnahmen vom Sicherheitsserver. Um den Schutzbedarf der Anwendungen dennoch zu erfüllen ist es notwendig, Übertragungsfehler zu erkennen und angemessen darauf zu reagieren.
Wechselt der Kontext in kürzeren Intervallen, als Sicherheitsmaßnahmen ermittelt und umgesetzt werden können, ist eine zeitnahe Reaktion auf Bedrohungen nicht möglich. Eine weiterführende Betrachtung dieser Problematik sollte Aufschluss über die praktische Relevanz geben und mögliche Lösungswege aufzeigen.
Das implementierte Architekturmodell sieht die Umsetzung von Sicherheitsmaßnahmen auf Anwendungsebene vor, wenn sich die Sicherheitsmaßnahmen auf Anwendungsdaten beziehen. Eine Erweiterung auf das in Abschnitt 4.2.2 vorgestellte Architekturmodell 2 würde den Implementierungsaufwand einer kompatiblen Anwendung gegenüber dem gewählten Ansatz reduzieren, da alle Sicherheitsmaßnahmen auf Betriebssystemebene umgesetzt werden. Um durch den lokalen Sicherheitsdienst vor Bedrohungen geschützt zu werden, müssten Anwendungen dann nur noch ihren Anwendungskontext an den lokalen Sicherheitsdienst übermitteln.

Literaturverzeichnis

[ABKS09] An, Gaeil, Guantae Bae, Kiyoung Kim und Dongil Seo: *Context-aware Dynamic Security Configuration for mobile Communication Device*. New Technologies, Mobility and Security, 2009.

[Anda] Android Developer: *Activities*. `http://developer.android.com/guide/topics/fundamentals/activities.html`. (Abgerufen am 15.10.2011).

[Andb] Android Developer: *Android Interface Definition Language (AIDL)*. `http://developer.android.com/guide/developing/tools/aidl.html`. (Abgerufen am 20.11.2011).

[Andc] Android Developer: *Data Storage*. `http://developer.android.com/guide/topics/data/data-storage.html`. (Abgerufen am 15.10.2011).

[Andd] Android Developer: *Platform Versions*. `http://developer.android.com/resources/dashboard/platform-versions.html`. (Abgerufen am 18.10.2011).

[AvH09] Antoniou, Grigoris und Frank van Harmelen: *Web Ontology Language: OWL*. In: *Handbook on Ontologies*, Seiten 91–110. Springer Verlag, 2009.

[Ay07] Ay, Feruzan: *Context Modeling and Reasoning using Ontologies*. `http://www.ponnuki.de/cmaruo/cmaruo.pdf`, 2007. (Abgerufen am 19.12.2011).

[BAF+03] Benford, S., R. Anastasi, M. Flintham, C. Greenhalgh, N. Tandavanitj, M. Adams und J. Row-Farr: *Coping with uncertainty in a location-based game*. Pervasive Computing, 2003.

[BBC97] Brown, Peter J., John D. Bovey und Xian Chen: *Context-Aware Applications: From the Laboratory to the Marketplace*. IEEE Personal Communications, 1997.

[BBH+10] Bettini, Claudio, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan und

Daniele Riboni: *A Survey of Context Modelling and Reasoning Techniques.* Pervasive and Mobile Computing, Volume 6, 2010.

[Bec09] Becher, Michael: *Security of Smartphones at the Dawn of their Ubiquitousness.* Doktorarbeit, Universität Mannheim, 2009.

[BP10] Becker, Arno und Marcus Pant: *Android 2 Grundlagen und Programmierung.* dpunkt.verlag, zweite Auflage, 2010.

[Bro92] Brockhaus Verlag: *Der Brockhaus in einem Band.* F.A. Brockhaus GmbH, vierte Auflage, 1992.

[CFT08] Clark, Kendall Grant, Lee Feigenbaum und Elias Torres: *SPARQL Protocol for RDF.* `http://www.w3.org/TR/rdf-sparql-protocol/`, Januar 2008. (Abgerufen am 26.11.2011).

[CJB99] Chandrasekaran, B., John R. Josephson und V. Richard Benjamins: *What Are Ontologies, and Why Do We Need Them?* IEEE Intelligent Systems and their Applications, Volume 14, 1999.

[CK00] Chen, Guanling und David Kotz: *A Survey of Context-Aware Mobile Computing Research.* Dartmouth Computer Science Technical Report TR2000-381, 2000.

[CLRS07] Cormen, Thomas H., Charles E. Leiserson, Ronald Rivest und Clifford Stein: *Algorithmen - Eine Einführung.* Oldenbourg Verlag, zweite Auflage, 2007.

[DA00] Dey, Anind K. und Gregory D. Abowd: *Towards a better understanding of context and context-awareness.* Workshop on The What, Who, Where, When, Why and How of Context-Awareness, Conference on Human Factors in Computer Systems, 2000.

[Dar06] Dargie, Waltenegus: *A Distributed Architecture for Computing Context in Mobile Devices.* Doktorarbeit, Technische Universität Dresden, 2006.

[Dey00] Dey, Anind K.: *Providing Architectural Support for Building Context-Aware Applications.* Doktorarbeit, Georgia Institute of Technology, 2000.

[DH06] Dargie, Waltenegus und Thomas Hamann: *A Distributed Architecture for Reasoning about a Higher-Level Context.* IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, 2006.

[Eck09] Eckert, Claudia: *IT-Sicherheit.* Oldenbourg Verlag, Sechste Auflage, 2009.

[EKR+05] Ehses, Erich, Lutz Köhler, Petra Riemer, Horst Stenzel und Frank Victor: *Betriebssysteme.* Pearson Studium, 2005.

[FK11] Fischer, Kristian und Stefan Karsch: *Modelling Security Relevant Context - An approach towards Adaptive Security in Volatile Mobile Web Environments.* In Proceedings of the ACM WebSci'11, 2011.

[Gar11] Gartner: *Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012.* `http://www.gartner.com/it/page.jsp?id=1622614`, April 2011. (Abgerufen am 12.10.2011).

[HI04] Henricksen, Karen und Jadwiga Indulska: *Modelling and Using Imperfect Context Information.* Pervasive Computing and Communications Workshops, 2004.

[JAB10] Johnson, Gleneesha M., Ashok Agrawala und Elodie Billionniere: *A Framework for Shrink-Wrapping Security Services.* IEEE International Conference on Services Computing, 2010.

[Joh09] Johnson, Gleneesha M.: *Towards shrink-wrapped security: A taxonomy of security-relevant context.* IEEE International Conference on Pervasive Computing and Communications, 2009.

[Kar10] Karsch, Stefan: *Vorlesungsskript IT-Sicherheit.* 2010.

[Kru10] Krumnow, Benjamin: *Mehrprozessbetrieb für mobile kontextsensitive Anwendungen - Konzeption und prototypische Implementierung auf Basis eines bestehenden Rahmenwerks.* Bachelor Thesis, Fachhochschule Köln, 2010.

[MB03] Mostéfaoui, Ghita Kouadri und Patrick Brézillon: *A Generic Framework for Context-Based Distributed Authorizations.* In: *Modeling and Using Context*, Nummer 2680, Seiten 204–217. Springer Verlag, 2003.

[Mül10] Müller, Felix: *Modellierung und Repräsentation des Kontextes von mobilen Nutzungsszenarien - Ein Rahmenwerk für mobile kontextsensitive Applikationen.* Master Thesis, Fachhochschule Köln, 2010.

[NF11] Nurmi, Petteri und Patrik Floréen: *Reasoning in Context-Aware Systems.* `http://www.cs.helsinki.fi/u/ptnurmi/papers/positionpaper.pdf`, 2011. (Abgerufen am 20.12.2011).

[Ora] Oracle: *Java Platform, Standard Edition 6 API Specification.* `http://download.oracle.com/javase/6/docs/api/java/util/Hashtable.html`. (Abgerufen am 22.10.2011).

[Pas98] Pascoe, Jason: *Adding generic contextual capabilities to wearable computers.* In the Proceedings of the 2nd IEEE International Symposium on Wearable Computers, 1998.

[PS08] Prud'hommeaux, Eric und Andy Seaborne: *SPARQL Query Language for RDF.* `http://www.w3.org/TR/rdf-sparql-query/`, Januar 2008. (Abgerufen am 26.11.2011).

[Rot08] Rothermel, Kurt: *Kontextbezogene Systeme – die Welt im Computer modelliert.* In: *Digitale Visionen*, Seiten 31–42. Springer Verlag, 2008.

[Rus11] Rusher, Jack: *Triple Store.* `http://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html`, 2011. (Abgerufen am 22.11.2011).

[SAW94] Schilit, Bill N., Norman Adams und Roy Want: *Context-Aware Computing Applications.* In the Proceedings of the 1st International Workshop on Mobile Computing Systems and Applications, 1994.

[SBG98] Schmidt, Albrecht, Michael Beigl und Hans-W. Gellersen: *There is more to Context than Location.* In Proceedings of Workshop on Interactive Applications of Mobile Computing, 1998.

[Sit09] Sitou, Wassiou: *Requirements Engineering kontextsensitiver Systeme.* Softwaretechnik-Trends, Band 29(1), Februar 2009.

[SLP04] Strang, Thomas und Claudia Linnhoff-Popien: *A Context Modelling Survey.* Workshop Proceedings: First International Workshop on Advanced Context Modelling, 2004.

[Spr04] Springer, Thomas: *Ein komponentenbasiertes Meta-Modell kontextabhängiger Adaptionsgraphen für mobile und ubiquitäre Anwendungen.* Doktorarbeit, Technische Universität Dresden, 2004.

[ST93] Spreitzer, Mike und Marvin Theimer: *Providing Location Information in a Ubiquitous Computing Environment.* SOSP '93 Procee-

dings of the fourteenth ACM symposium on Operating systems principles, 1993.

[ST94] Schilit, Bill N. und Marvin M. Theimer: *Disseminating active map information to mobile hosts.* IEEE Network Volume 8, 1994.

[Top] Topcu, Ferit: *Context Modeling and Reasoning Techniques.* `http://www.snet.tu-berlin.de/fileadmin/fg220/courses/SS11/snet-project/context-modeling-and-reasoning_topcu.pdf`. (Abgerufen am 19.12.2011).

[Tur06] Turjalei, Mirwais: *Integration von Context-Awareness in eine Middleware für mobile Systeme.* Doktorarbeit, Universität Hamburg, 2006.

[VB94] Voelker, Geoffrey M. und Brian N. Bershad: *Mobisaic: An Information System for a Mobile Wireless Computing Environment.* In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications, 1994.

[Voß03] Voss, Jakob: *Begriffssysteme – Ein Vergleich verschiedener Arten von Begriffssystemen und Entwurf des integrierenden Datenmodells.* `http://eprints.rclis.org/bitstream/10760/8308/1/begriffssysteme.pdf`, 2003. (Abgerufen am 19.12.2011).

[vSvHW+06] Sinderen, Marten J. van, Aart T. van Halteren, Maarten Wegdam, Hendrik B. Meeuwissen und E. Henk Eertink: *Supporting Context-Aware Mobile Applications: An Infrastructure Approach.* IEEE Communications Magazine, Volume 44, 2006.

[WGZP04] Wang, Xiao Hang, Tao Gu, Da Qing Zhang und Hung Keng Pung: *Ontology Based Context Modeling and Reasoning using OWL.* Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.

[XXW09] Xu, Hui, Debao Xiao und Zheng Wu: *Application of Security Ontology to Context-Aware Alert Analysis.* Eighth IEEE/ACIS International Conference on Computer and Information Science, 2009.

A Anhang

A.1 Inhalt der beigefügten DVD

Das Verzeichnis *Dokumente* enthält eine digitale Version der Diplomarbeit, sowie jeweils einen in deutscher und englischer Sprache verfassten Abstract. Alle Dateien im Verzeichnis *Dokumente* liegen im PDF-Format vor.
Im Verzeichnis *Software* befinden sich die im Rahmen dieser Arbeit implementierten Architekturkomponenten. Dazu gehört der lokale Sicherheitsdienst, der Sicherheitsserver und eine Beispielanwendung. Die Kompilierung des lokalen Sicherheitsdienstes sowie der Beispielanwendung setzt die Installation der Eclipse Entwicklungsumgebung und des Android SDK voraus. Die Inbetriebnahme des Sicherheitsserver erfordert unter anderem die Installation des Sinatra Frameworks.

A.2 Klassendiagramm des lokalen Sicherheitsdienstes

Der lokale Sicherheitsdienst aus Abschnitt 5.2 besteht aus den Paketen *de.contextsecurityframework*, *de.contextsecurityframework.contextsources*, *de.contextsecurityframework.data* und *de.contextsecurityframework.helpers*. Das Paketkonzept des lokalen Sicherheitsdienstes wurde aus [Kru10] übernommen.
Das Paket *de.contextsecurityframework* enthält den Kontextmanager, den Dienstmanager, die statische Klasse *SecurityActions* zur Ausführung von Sicherheitsmaßnahmen auf Betriebssystemebene, die RPC-Schnittstellen und die Activities. Das Paket *de.contextsecurityframework.contextsources* enthält die Kontextquellen und das Paket *de.contextsecurityframework.data* enthält die Datenstrukturen sowie die AIDL-Dateien zur Übertragung der Datenstrukturobjekte zwischen dem Dienstmanager und den Anwendungen. Die Klasse *RequestPool* wurde als Hilfsklasse des Dienstmanagers im Paket *de.contextsecurityframework.helpers* abgelegt. Die Klassen des lokalen Sicherheitsdienstes und deren Beziehungen sind im Klassendiagramm in Abbildung A.1 und Abbildung A.2 aufgeführt.

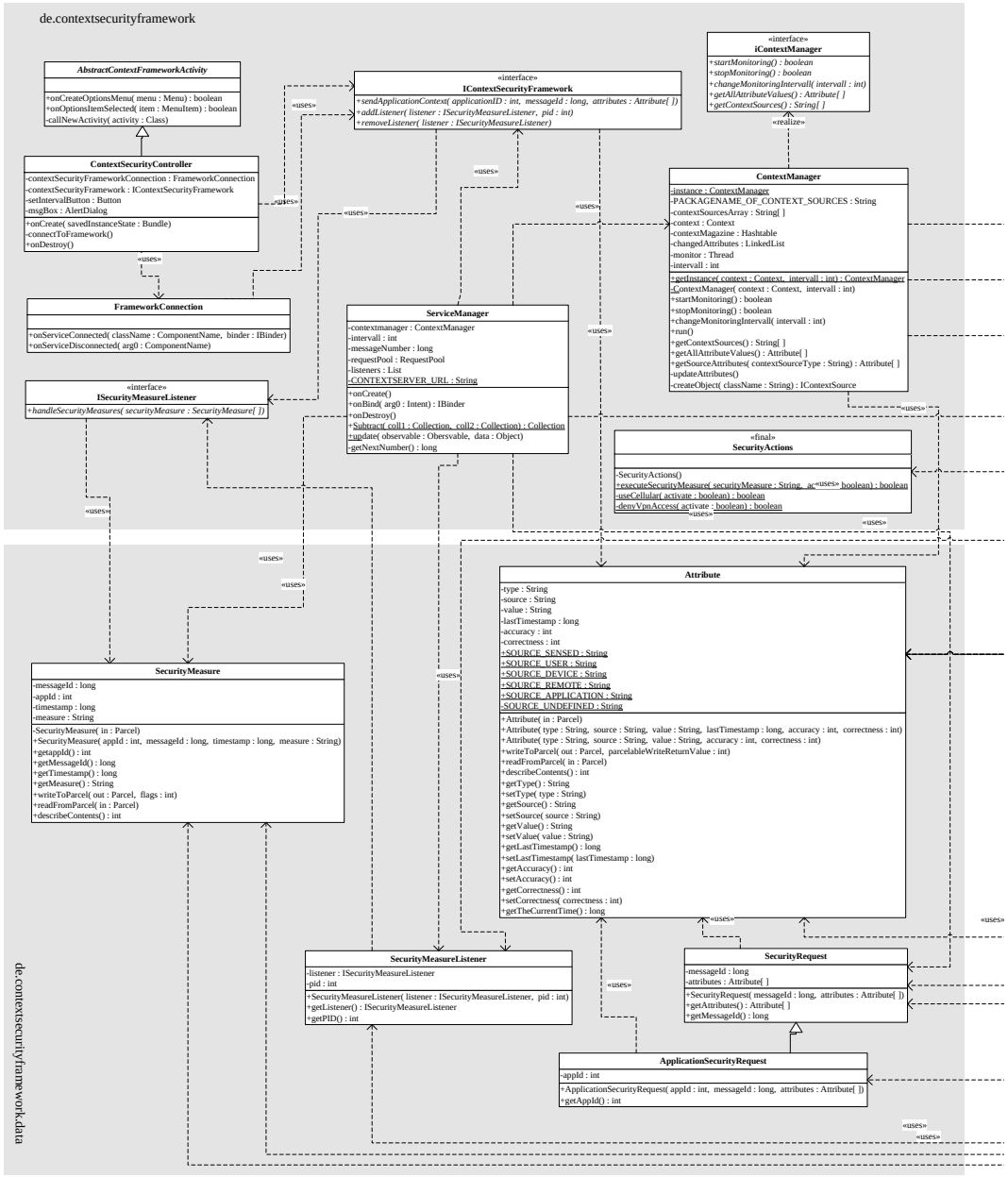

Abbildung A.1: Klassendiagramm des lokalen Sicherheitsdienstes (Teil 1)

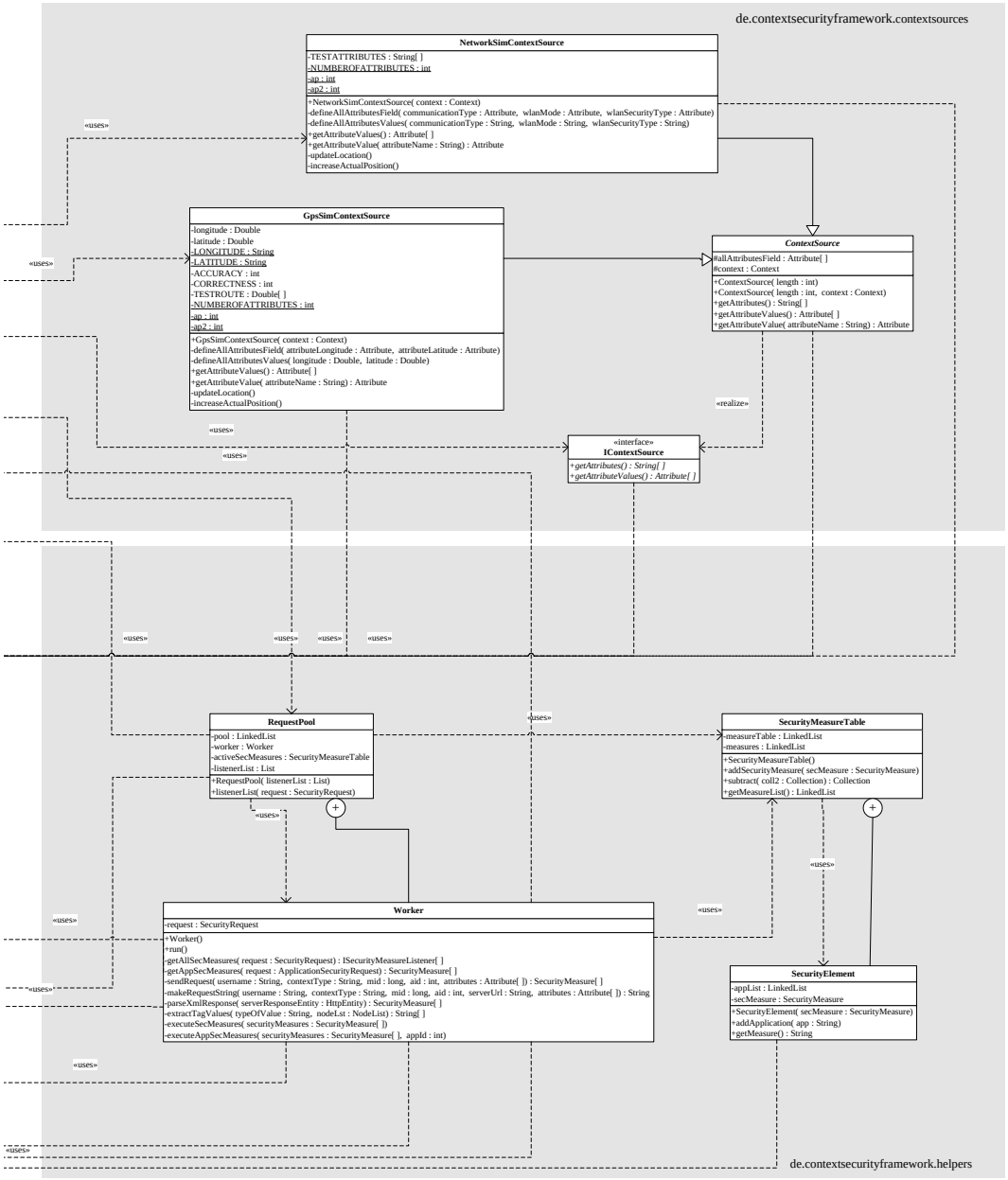

Abbildung A.2: Klassendiagramm des lokalen Sicherheitsdienstes (Teil 2)

A.3 Regeln der Beispielontologie

Anwendungskontexte

Kontext	Regeln
#businessMail	hasMailType: “business“ ∧ belongsToUser: #Smartphone1
#privateMail	hasMailType: “private“ ∧ belongsToUser: #Smartphone1
#confidentialMail	hasMailSecurity: “confidential“ ∧ belongsToUser: #Smartphone1

Tabelle A.1: Anwendungskontexte der Beispielontologie

Gerätekontext

Kontext	Regeln
#CampusLocationSmartphone1	hasLatitudeMin: 51.022 ∧ hasLatitudeMax: 51.025 ∧ hasLongitudeMin: 7.561 ∧ hasLongitudeMax: 7.565 ∧ belongsToUser: #Smartphone1

Tabelle A.2: Gerätekontext der Beispielontologie

Schwachstellen

Kontext	Regeln
#unsecureCommunication	hasCommunicationType: “wlan“ ∧ hasWlanSecurityType: “wep“ ∧ hasWlanMode: “open“ ∧ belongsToUser: #Smartphone1
#displayVisibility	belongsToUser: #Smartphone1

Tabelle A.3: Schwachstellen der Beispielontologie

potentielle Ereignisse

Kontext	Regeln
#sendMail	hasEvent: “sendMail“ ∧ belongsToUser: #Smartphone1
#openMail	hasEvent: “openMail“ ∧ belongsToUser: #Smartphone1

Tabelle A.4: Potentielle Ereignisse der Beispielontologie

Bedrohungen

Kontext	Regeln
#eavesdroppingMail	hasSubContext: #sendMail ∧ hasSubContext: #unsecureCommunication ∧ belongsToUser: #Smartphone1
#informationTheft	hasSubContext: #displayVisibility ∧ hasSubContext: #openMail ∧ belongsToUser: #Smartphone1

Tabelle A.5: Bedrohungen der Beispielontologie

Sicherheitsmaßnahmen

Kontext	Regeln
#denyMailAccess	hasSubContext: #businessMail ∧ hasSubContext: #confidentialMail ∧ belongsToUser: #Smartphone1
#encryptConnectionBusiness	hasSubContext: #businessMail ∧ hasSubContext: #eavesdroppingMail ∧ belongsToUser: #Smartphone1
#encryptConnection	hasSubContext: #privateMail ∧ hasSubContext: #eavesdroppingMail ∧ belongsToUser: #Smartphone1
#useCellular	hasSubContext: #businessMail ∧ hasSubContext: #eavesdroppingMail ∧ belongsToUser: #Smartphone1
#denyVpnAccess	hasSubContext: #CampusLocationSmartphone1 ∧ belongsToUser: #Smartphone1

Tabelle A.6: Sicherheitsmaßnahmen der Beispielontologie