Hochschule Hannover
Fakultät IV
Abteilung Informatik

Complex Event Processing im Kontext eines IF-MAP Clients auf mobilen Geräten (Android)

Bachelorarbeit im Studiengang Angewandte Informatik

Michael Felchner

Dezember 2014

Beteiligte Personen

Erstprüfer

Prof. Dr. Ralf Bruns
Hochschule Hannover
Ricklinger Stadtweg 120
30459 Hannover
E-Mail: ralf.bruns@hs-hannover.de
Tel.: +49 511 9296-1817

Zweitprüfer

Leonard Renners M. Sc.
Hochschule Hannover
Ricklinger Stadtweg 120
30459 Hannover
E-Mail: leonard.renners@hs-hannover.de
Tel.: +49 511 9296-1828

Autor

Michael Felchner
Friedrich-Meyer-Straße 11
31535 Neustadt am Rübenberge
Matrikel-Nr.: 1195571
E-Mail: michael.felchner@stud.hs-hannover.de
Tel.: +49 5032 2741

Selbstständigkeitserklärung

Hiermit erkläre ich an Eides statt, dass ich die eingereichte Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Hannover, den 29. Dezember 2014

_____
Michael Felchner

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Abkürzungsverzeichnis

AAL	Ambient Assisted Living
AR	Access Requestor
ARC	Asynchronous Receive Channel
ART	Android Runtime
CADS	Context-related Signature and Anomaly Detection for Smartphones
CEP	Complex Event Processing
CQL	Continuous Query Language
DVM	Dalvik Virtual Machine
EDA	Event-Driven Architecture
EPA	Event Processing Agent
EPL	Event Pattern Language
EPN	Event Processing Network
IF-MAP	Interface for Metadata Access Points
MAPC	Metadata Access Point Client
MAPS	Metadata Access Point Server
PDP	Policy Decision Point
PEP	Policy Enforcement Point
SIM	Subscriber Identity Module
SSRC	Synchronous Send Receive Channel
TCG	Trusted Computing Group
TNC	Trusted Network Connect
TNC-WG	Trusted Network Connect Working Group

1 Einleitung

1.1 Motivation

Seit einigen Jahren gewinnen moderne mobile Geräte wie Smartphones oder Tablets zunehmend an wirtschaftlicher Bedeutung. Durch die ständige Weiterentwicklung der zugrundeliegenden Hardwarekomponenten und die Erstellung immer komplexer werdender Software-Produkte für mobile Plattformen sind solche Geräte zu einem wichtigen Werkzeug geworden, das bereits in vielen Bereichen des alltäglichen Lebens eine entscheidende Rolle spielt. Als Beispiele für heutige mobile Aktivitäten auf Smartphones und Tablets sind dabei u. a. *Text-Messaging*, *Online-Shopping*, *Mobile-Banking*, *Mobile-Gaming*, *Navigating* oder *Social-Networking* zu nennen.

Wurden im Jahr 2010 weltweit knapp 300 Millionen Smartphones bzw. Tablets ausgeliefert, stieg die Anzahl der verkauften Geräte schon im folgenden Jahr auf rund 500 Millionen Einheiten. Bereits im Jahr 2013 belief sich der globale Absatz an Smartphones und Tablets erstmals auf mehr als eine Milliarde Geräte. Viele Marktforschungsinstitute und Analysten prognostizieren, dass sich dieses rasante Wachstum in den kommenden Jahren noch weiter fortsetzen wird. Weiterführende Statistiken dazu sind u. a. bei *Gartner* [19] oder auf der Webseite der *International Data Corporation* (IDC) [22] zu finden. Ein ähnlicher Trend zeichnet sich auch in Deutschland ab. Der *Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V.* (BITKOM) schätzt in seiner aktuellen Presseinformation [6] den Absatz an Smartphones in Deutschland im Jahr 2014 auf rund 30 Millionen Einheiten. Im Februar 2014 wurde ermittelt, dass 40,4 Millionen Menschen hierzulande mindestens ein Smartphone oder Tablet besitzen. Abbildung 1.1 zeigt einen Überblick über den Absatz mobiler Geräte in Deutschland der vergangenen fünf Jahre.

Abbildung 1.1: Absatz mobiler Geräte in Deutschland von 2009 bis 2014, Quelle: [26]

Aus den oben aufgezeigten Statistiken resultiert die logische Konsequenz, dass Smartphones und Tablets auch im Geschäftsleben vieler Unternehmen längst einen festen Platz eingenommen haben. Die geschäftliche Nutzung beschränkt sich dabei nicht mehr nur auf die Grundfunktionen *mobiles Telefonieren* und *SMS lesen/schreiben*, sondern umfasst vielmehr auch die Verwendung von komplexen geschäftlichen Anwendungen mit teilweise sensitiven Unternehmensaufgaben. Dadurch sind diese Geräte in den letzten Jahren allerdings auch stärker in den Fokus krimineller Aktivitäten gerückt. Aufgrund der vielfältigen Anwendungsmöglichkeiten von Smartphones und Tablets entstehen letztlich neue akute Bedrohungen für die Sicherheit von Unternehmensnetzen und sensiblen Unternehmensdaten. Weitere Sicherheitsbedrohungen können sich auch dadurch ergeben, dass solche Geräte eine ganze Fülle von verschiedenen Sensoren besitzen und oftmals über einen sehr langen Zeitraum eingeschaltet sowie mit dem Internet verbunden sind. Außerdem ist es auf Smartphones und Tablets grundsätzlich möglich, diverse mobile Anwendungen von Drittanbietern, sogenannte *Apps*, zu installieren, um die Geräte an die persönlichen Bedürfnisse und Gewohnheiten der Benutzer anzupassen. Einige der verfügbaren Anwendungen weisen dabei jedoch fragwürdige Funktionen, sicherheitskritische Systemberechtigungen (*Permissions*) und einen bedenklichen Umgang mit persönlichen Daten auf. Wenn Nutzer von Smartphones solche Anwendungen arglos installieren und verwenden, kann dies zu unterschiedlichen Problemsituationen führen. Gefahren, die von heutiger mobiler Malware ausgehen, reichen dabei z. B. von einem ungewöhnlichen Anstieg des Energieverbrauchs und Systemabstürzen über *Botnetze*[1] und *DDoS-Attacken*[2] bis hin zur versteckten Nutzung kostenpflichtiger Dienste, den Diebstahl sensibler oder persönlicher Daten sowie der Spionage von Sensorinformationen durch sogenannte *Sensory-Malware*. Der Schutzbedarf vor Risiken und Bedrohungen für IT-Infrastrukturen durch moderne mobile Geräte liegt daher mindestens auf demselben Niveau eines herkömmlichen Computers. Mit dem steigenden Absatz von Smartphones ging in den letzten Jahren dementsprechend eine signifikante quantitative und qualitative Zunahme mobiler Schadprogramme aller Art einher. Dabei zeigen Funk und Garnaeva in ihrem Online-Artikel [18], dass diese Zunahme vor allem die Android-Plattform von *Google* betrifft. Dies lässt sich einerseits auf die derzeitige Popularität des Android-Betriebssystems zurückführen (84,7% weltweiter Marktanteil im zweiten Quartal 2014 [22]). Andererseits bietet Android aber auch eine offene Plattform, sodass mobile Anwendungen neben dem offiziellen *App-Store Google Play* auch über andere nicht-offizielle Quellen bezogen und installiert werden können.

Die Integration von Smartphones und Tablets für geschäftliche Tätigkeiten stellt für viele Unternehmen eine aktuelle Schlüsselaufgabe dar, um die Produktivität der eigenen Mitarbeiter zu erhöhen und damit einen Wettbewerbsvorteil gegenüber der Konkurrenz zu erreichen. Viele Unternehmen werden jedoch durch diese Integration mit einer Fülle neuartiger Bedrohungen für das eigene Unternehmensnetz und wichtige Unternehmensdaten konfrontiert. Aus den oben genannten Gründen gilt dies im besonderen Maße in Bezug auf mobile Geräte mit dem Android-Betriebssystem.

[1] Ferngesteuerte Computernetzwerke z. B. zur Durchführung von *DDoS-Attacken*

[2] Überlastung von IT-Infrastrukturen, um ihre Verfügbarkeit zu beeinträchtigen

1.2 Ziele der Arbeit

Der Umgang mit mobilen Geräten im Kontext sicherheitskritischer Unternehmensnetze und -ressourcen bildet die zentrale Ausgangsbasis der vorliegenden Arbeit. Für die sichere Integration von modernen mobilen Geräten wie Smartphones oder Tablets in bestehende Unternehmensnetze müssen über die jeweiligen Geräte verschiedene Informationen gesammelt werden, die eine Beurteilung über den aktuellen Sicherheitszustand ermöglichen. Dadurch sollen Sicherheitsprobleme der Geräte möglichst frühzeitig erkannt werden, sodass z. B. der Zugang zu unternehmensinternen Ressourcen und sensiblen Unternehmensdaten für die betroffenen Geräte verwehrt bleibt und somit kein Schaden für das Unternehmen entsteht.

Zur Lösung dieser Aufgabe existieren bereits einige netzwerkbasierte Produkte und Strategien im Kontext dieser Arbeit. Diese ermöglichen u. a. die Integration mobiler Geräte mit dem Android-Betriebssystem in das IF-MAP[3] Umfeld bestehender IT-Infrastrukturen von Unternehmen. Die entsprechenden Lösungsansätze werden dazu in dieser Arbeit näher erläutert und bewertet (siehe Abschnitt 3.1). Ein Teilaspekt dieser Lösungen besteht darin, dass verschiedene sicherheitsrelevante Informationen auf den mobilen Geräten gesammelt und an eine zentrale Komponente im IF-MAP Umfeld, den sogenannten MAP-Server[4], übertragen werden müssen. Andere Systeme im Kontext von IF-MAP, die wiederum eine Beurteilung des Sicherheitszustands der Geräte durchführen, können ihrerseits die notwendigen Daten von diesem MAP-Server beziehen. Sollte jedoch die Menge an Informationen, die auf den Smartphones und Tablets gesammelt und an den zentralen MAP-Server übertragen werden muss, sehr groß sein, kann dies unter Umständen zu Problemen im Kontext der bereits bestehenden Ansätze führen. Die Bereitstellung sicherheitsrelevanter Informationen an andere Systeme und die damit verbundene Analyse des Sicherheitszustands könnte dann ggf. nicht mehr zeitnah möglich sein. Zudem drohen je nach Anwendungsfall auch Konflikte mit Datenschutzaspekten bzgl. der Sammlung, Verarbeitung und Übertragung personenbezogener Daten.

Die Zielsetzung der vorliegenden Arbeit umfasst somit den Entwurf eines erweiterten Konzepts für einen IF-MAP Client, der die oben beschriebenen Defizite bereits bestehender Lösungen unter Verwendung von *Complex Event Processing* eliminiert. Die zentralen Ziele dieses Konzepts bestehen letztlich in der Notwendigkeit, die an den MAP-Server übertragenen Daten möglichst stark zu reduzieren und personenbezogene Daten bereits vor der Übertragung auf den mobilen Geräten zu anonymisieren.

Zur Erreichung dieser Ziele befasst sich diese Arbeit mit der Lösung, eine mobile Anwendung für die Android-Plattform zu entwickeln, die sicherheitsrelevante Informationen über die zugrundeliegenden Geräte sammelt und diese über eine IF-MAP Anbindung an einen MAP-Server für die weitergehende Sicherheitsanalyse durch andere Systeme im IF-MAP Umfeld zur Verfügung stellt. Dabei soll die Menge der Informationen, die von den mobilen Geräten an den MAP-Server übertragen werden muss, möglichst gering gehalten und zusätzlich bestimmte Datenschutzaspekte für personenbezogene Da-

[3] Interface for Metadata Access Points; Protokoll zum Austausch verschiedener Statusinformationen
[4] *Metadata Access Point Server*; zentraler Speicherort für sämtliche *Metadaten*

ten beachtet werden. Um die Anzahl der zu übertragenden Informationen zu reduzieren, werden die gesammelten Daten direkt auf den Smartphones mittels einer mobilen CEP[5]-Komponente und eines entsprechenden *Regelwerks* verdichtet. Anschließend werden die verarbeiteten Daten anderen Systemen für die Sicherheitsanalyse über eine IF-MAP Anbindung an einen MAP-Server in nahezu Echtzeit zur Verfügung gestellt. Damit haben diese Analysekomponenten auch bei einer großen Menge an Informationen, die auf den mobilen Geräten gesammelt wird, die Möglichkeit, frühzeitig den Sicherheitszustand der jeweiligen Geräte festzustellen und geeignete Maßnahmen einzuleiten, wie z. B. den Zugriff auf bestimmte Ressourcen des Unternehmens zu limitieren. Das entworfene Konzept beschränkt sich jedoch ausdrücklich nicht nur auf sicherheitsrelevante Daten, sondern bietet vielmehr eine große Flexibilität zur Unterstützung diverser Anwendungsdomänen, bei denen beliebige Daten der mobilen Geräte gesammelt, verarbeitet und übertragen werden. Als Endprodukt dieser Arbeit wird das entworfene Konzept in Form eines Prototyps für Android implementiert, der exemplarisch die Funktionsweise des Clients anhand einiger Sicherheitsdaten demonstriert. Der Prototyp bildet somit ein Teilsystem im IF-MAP Umfeld, das Sicherheitsdaten der mobilen Geräte für andere Komponenten zur Verfügung stellt.

1.3 Einordnung in den Kontext

Der entwickelte IF-MAP Client für mobile Geräte mit dem Android-Betriebssystem gliedert sich fachlich primär in das Themengebiet der IT-Sicherheit ein. Das von der *Trusted Computing Group* (TCG) spezifizierte IF-MAP Kommunikationsprotokoll wird dabei für den Austausch der sicherheitsrelevanten Informationen zwischen dem MAP-Client und einem zentralen MAP-Server verwendet. Sowohl ein MAP-Client als auch ein MAP-Server sind spezifische Komponenten der TNC-Architektur[6], die von der *Trusted Network Connect Working Group* (TNC-WG), einer Untergruppierung der TCG, in [31] entwickelt wurde. Der MAP-Server dient hierbei als zentrale Netzwerkdatenbank zur Speicherung diverser Informationen über den aktuellen Netzwerk- und Sicherheitszustand, die von unterschiedlichen MAP-Clients an den MAP-Server übertragen werden können. Diese Informationen bilden die Form einer ungerichteten und ungewichteten Graphenstruktur, die aus *Identifiern*, *Metadaten* und *Links* besteht. Andere Komponenten, die im Kontext von IF-MAP und der TNC-Architektur zum Einsatz kommen, dienen wiederum dazu, Anomalien auf den mobilen Geräten oder im zugrundeliegenden Netzwerk zu erkennen und entsprechende Gegenmaßnahmen einzuleiten. Diese Systeme können die für ihre Sicherheitsanalyse notwendigen Informationen sowohl vom MAP-Server abonnieren als auch explizit nach bestimmten *Metadaten* im MAP-Server suchen. In der TNC-Architektur sind solche Analysekomponenten ebenfalls als MAP-Clients definiert. Auch hierbei wird für die Kommunikation mit dem MAP-Server das IF-MAP Protokoll verwendet.

[5] *Complex Event Processing*
[6] *Trusted Network Connect* Architektur

Für die technische Umsetzung des Informationsaustauschs zwischen dem entworfenen Client und einem zentralen MAP-Server wird die von der *Trust@HsH*-Forschungsgruppe entwickelte Bibliothek *ifmapj* verwendet. Diese Bibliothek bietet eine umfangreiche Unterstützung zur Erstellung eines MAP-Clients in der Programmiersprache *Java*. Auch unter Android kann die *ifmapj*-Bibliothek verwendet werden. Als Gegenstelle zum MAP-Client kann z. B. der MAP-Server *irond* zum Einsatz kommen, der ebenfalls an der Hochschule Hannover entwickelt wird. Zum Zeitpunkt der Erstellung dieser Arbeit basiert *irond* auf der Version 2.2 der offiziellen Spezifikation des IF-MAP Kommunikationsprotokolls [33] der TCG. Die Analyse des Sicherheitszustands der mobilen Geräte kann z. B. mit der *Correlation-Engine irondetect* umgesetzt werden, wobei die dafür benötigten Informationen entsprechend vom MAP-Server abonniert werden müssen. Dabei entstand *irondetect* als eine Hauptkomponente des von Bente in [5] entwickelten *CADS*[7]-Systems. Weiterhin ermöglicht die Applikation *VisITMeta* der *Trust@HsH*-Forschungsgruppe zusätzlich die Visualisierung der im MAP-Server verwalteten Daten. Der in der vorliegenden Arbeit entwickelte Client ordnet sich entsprechend in die Anwendungen der *Trust@HsH*-Forschungsgruppe ein.

Ein entscheidender technischer Aspekt dieser Arbeit bezieht sich auf den Einsatz der *Event Processing Engine Asper* [3] direkt auf den mobilen Geräten. Dabei soll die Menge der Informationen, die an den MAP-Server übertragen werden muss, durch ein entsprechendes *Regelwerk* zur Datenverdichtung deutlich reduziert werden. Dieses besteht aus *Ereignisregeln* in einer um CEP-Befehle erweiterten SQL-Syntax. Die Definition dieser *Ereignisregeln* basiert auf der fachlichen Analyse der sicherheitsrelevanten Informationen von mobilen Geräten im Umfeld sensibler Unternehmensnetze. Die CEP-Komponente *Asper* ist eine Portierung der Open-Source *Event Processing Engine Esper* [16] für die Android-Plattform, wobei *Asper* zum Zeitpunkt der Erstellung dieser Arbeit in der Version 4.9.0 vorliegt. Innerhalb des entwickelten Clients bildet die *Datenverarbeitung* durch *Asper* das Bindeglied zwischen der *Datensammlung* und der *Datenübertragung*. Die gesammelten Informationen des mobilen Geräts werden dabei zunächst durch *Asper* verdichtet und erst im Anschluss daran an den MAP-Server *irond* übertragen.

Die technische Basis für den entwickelten Client bildet die mobile Plattform Android von *Google* [20]. Für die Analyse des Sicherheitszustands der mobilen Geräte müssen verschiedene Sensor-, System-, Kontext- und Anwendungsinformationen mit Hilfe der Android-Plattform gesammelt werden. Zum Zeitpunkt der Erstellung dieser Arbeit liegt Android in der Version 4.4.4 (KitKat) vor.

Zur Integration von mobilen Geräten in das IF-MAP Umfeld existieren bereits verschiedene Lösungen, wie z. B. das oben angesprochene *CADS*-System. Ein Teilaspekt von *CADS* umfasst einen MAP-Client für Android, der Daten auf den mobilen Geräten sammelt und diese direkt an einen MAP-Server überträgt. Mit der Software *ironcontrol* bietet die *Trust@HsH*-Forschungsgruppe eine weitere Applikation an, die den Einsatz eines MAP-Clients unter Android mit der *ifmapj*-Bibliothek demonstriert. Alle bestehenden Systeme bilden somit eine wichtige Basis zur Entwicklung eines IF-MAP Clients für Android unter Verwendung einer *Event Processing Engine* zur Datenverdichtung.

[7] Context-related Signature and Anomaly Detection for Smartphones

1.4 Aufbau der Arbeit

Grundlagen

Das Kapitel 2 bietet aufbauend auf die in der Einleitung erarbeitete Zielsetzung zunächst eine Einführung in mobile Geräte sowie mobile Plattformen. Im Anschluss daran erfolgt eine Analyse der vom Client verwendeten Basistechnologien und Softwarekomponenten.

Bestehende Lösungen im fachlichen und technischen Umfeld

Nachdem im vorherigen Kapitel die Grundlagen für den eigenen Client vorgestellt wurden, werden in Kapitel 3 die bereits bestehenden fachlichen und technischen Lösungen im Kontext des Clients analysiert, bewertet und gegen das eigene Konzept abgegrenzt.

Anforderungsanalyse zum entwickelten IF-MAP Client

In Kapitel 4 werden die Anforderungen an den eigenen Client anhand eines Anwendungsszenarios erarbeitet. Dazu erfolgt auch eine Auflistung der unter Android sammelbaren Daten. Abschließend werden Datenschutzaspekte im Kontext des Clients untersucht.

Konzepte des entwickelten IF-MAP Clients

Nachdem die Anforderungen an den eigenen Client spezifiziert wurden, erfolgt in Kapitel 5 der konzeptionelle Entwurf des Clients. Dazu werden der allgemeine Aufbau des Clients sowie Lösungsstrategien zur *Datensammlung*, *-verarbeitung* und *-übertragung* erläutert.

Entwicklung des Prototyps

Aufbauend auf den konzeptionellen Entwurf des eigenen Clients erfolgt in Kapitel 6 die Entwicklung eines Prototyps zur Sammlung, Verarbeitung und Übertragung sicherheitsrelevanter Smartphone-Daten im Kontext des *CADS*-Systems.

Bewertung von CEP im Kontext des entwickelten IF-MAP Clients

Nachdem der Prototyp für den eigenen Client vorgestellt wurde, bietet das Kapitel 7 einen Ausblick auf zusätzliche Erweiterungsmöglichkeiten der CEP-Komponente des Clients. Es werden aber auch die Grenzen von CEP im Kontext des Clients erläutert.

Fazit

Die erzielten Ergebnisse dieser Arbeit werden schließlich in Kapitel 8 zusammengefasst und kritisch bewertet. Abschließend erfolgt ein Ausblick auf zukünftige Entwicklungen.

1.5 Typographische Konventionen

- *kursiver Text* Schlüsselwörter, Technologien und Softwarekomponenten
- `teletype` Namen für Klassen, Methoden und Attribute

2 Grundlagen

Aufbauend auf die in der Einleitung erarbeitete Zielsetzung der vorliegenden Arbeit bietet dieses Kapitel zunächst eine Einführung in mobile Geräte sowie mobile Plattformen (siehe Abschnitt 2.1). Im Anschluss daran erfolgt eine Analyse der zugrundeliegenden Basistechnologien und Softwarekomponenten, die vom entwickelten Client verwendet werden. Dazu werden in Abschnitt 2.2 die grundlegenden Eigenschaften der Android-Plattform vorgestellt. Den Abschluss dieses Kapitels bilden die Abschnitte 2.3 zum Themenbereich *Complex Event Processing* und 2.4 zur Erläuterung des IF-MAP Kommunikationsprotokolls.

2.1 Mobile Geräte

2.1.1 Begriffsdefinition

Seit vielen Jahren existieren mobile Geräte in verschiedenen Ausprägungen. *Motorola* stellte beispielsweise mit dem *DynaTAC 8000X* bereits im Jahr 1973 das erste handliche Mobiltelefon für den nordamerikanischen Markt vor [25]. Doch gerade in den vergangenen Jahren hat sich das Umfeld für mobile Geräte durch die Entwicklung immer neuer Modelle und Konzepte stark verändert. Zur Förderung des fachlichen Verständnisses werden nachfolgend die Begriffe „Mobiles Gerät“, „Mobiltelefon“ und „Smartphone“ erläutert sowie gegeneinander abgegrenzt.

Mobiles Gerät

Mobile Geräte werden definiert als elektronische Einheiten, die sich durch ihre geringe Größe sowie ihr geringes Gewicht auszeichnen und dadurch tragbar bzw. mobil einsetzbar sind [37]. Die benötigte Energie beziehen sie aus einem eingebauten Akku. Die Bezeichnung „Mobiles Gerät“ stellt einen Oberbegriff für verschiedenste mobile Gerätekategorien und -typen dar. Neben Smartphones und Tablets zählen somit z. B. auch Notebooks, Subnotebooks, Personal Digital Assistants (PDAs), MP3[1]-Player, Navigationsgeräte, E-Book-Reader und seit neuestem sogenannte Wearables[2] zu den mobilen Geräten. Der Fokus dieser Arbeit liegt dabei ausschließlich auf der Integration mobiler Geräte wie Smartphones oder Tablets mit dem Android-Betriebssystem in das IF-MAP Umfeld bestehender IT-Infrastrukturen.

[1] ISO MPEG-1 Audio Layer 3

[2] Am Körper getragene Computersysteme, z. B. Datenbrillen, Computeruhren oder Fitness-Armbänder

Mobiltelefon

Der Verwendungszweck von reinen Mobiltelefonen beruht hauptsächlich auf den beiden Grundfunktionen *mobiles Telefonieren* und *SMS lesen/schreiben*. Um diese beiden Funktionen anbieten zu können, verwenden Mobiltelefone typischerweise eine SIM[3]-Karte eines entsprechenden Netzanbieters. Charakteristisch für Mobiltelefone sind u. a. ein relativ kleiner Bildschirm, stark begrenzte Prozessor- und Speicherkapazitäten sowie ein Betriebssystem mit einer eingeschränkten Funktionsvielfalt. Anwendungsfälle, die über die oben genannten Grundfunktionen hinausgehen, sind daher auf reinen Mobiltelefonen oftmals nicht möglich. Aufgrund der limitierten Hardwareressourcen besitzen viele Modelle jedoch eine teilweise lange Akkulaufzeit.

Neben den reinen Mobiltelefonen existieren außerdem sogenannte Feature-Phones. Diese Geräte bieten, zusätzlich zu den oben aufgezählten Grundfunktionen, weitere Anwendungsmöglichkeiten, z. B. durch eine eingebaute Kamera, einen größeren Bildschirm oder einen vorinstallierten Webbrowser. Es ist allerdings nur bedingt möglich, weitere mobile Anwendungen von Drittanbietern auf solchen Geräten zu installieren, da keine *App-Stores* o. Ä. existieren. Der durch die erweiterten Hardwarekomponenten gestiegene Energiebedarf führt jedoch zu entsprechend kürzeren Akkulaufzeiten [5].

Smartphone

Gegenüber reinen Mobiltelefonen oder Feature-Phones weisen Smartphones deutlich gestiegene Hardwarekapazitäten auf. Aufgrund der ständigen Weiterentwicklung der Modelle in den vergangenen Jahren besitzen aktuelle Geräte inzwischen Hardwarespezifikationen, die durchaus mit denen herkömmlicher Computer verglichen werden können. Dazu zählt neben einer verbesserten Prozessorleistung sowie einer erhöhten Kapazität des Arbeits- und Festspeichers auch der Einsatz diverser Kommunikationsschnittstellen wie z. B. WIFI, Bluetooth oder NFC[4].

Zur Grundausstattung jedes Smartphones gehören ergänzend zu dem Mikrofon und dem Lautsprecher zum Telefonieren auch eine oder mehrere eingebaute Kameras. Zusätzlich bieten alle Geräte die Möglichkeit, durch verschiedene Datennetzverbindungen (z. B. WLAN, GPRS[5], EDGE[6], HSDPA[7] oder LTE[8]) auf das Internet zuzugreifen. Viele moderne Smartphones verfügen über zahlreiche Sensoren zur Analyse ihrer physikalischen Umgebung. Dazu zählen z. B. GPS[9]-, Bewegungs-, Orientierungs-, Beschleunigungs-, Näherungs-, Temperatur- oder Lichtsensoren. Einige dieser Sensoren werden dabei in Form eines Hardwaremoduls in dem Smartphone verbaut, andere Sensoren lassen sich wiederum durch entsprechende Softwarekomponenten simulieren. Weitere externe Sensoren z. B. zur Messung des Herzschlags oder des Blutdrucks sind in Form von Wearables

[3] Subscriber Identity Module
[4] Near Field Communication
[5] General Packet Radio Service; bis zu 114 kbit/s
[6] Enhanced Data Rates for GSM Evolution; bis zu 400 kbit/s
[7] High Speed Download Packet Access; bis zu 7,2 Mbit/s
[8] Long Term Evolution; bis zu 100 Mbit/s
[9] Global Positioning System

verfügbar und werden meist über Bluetooth mit den Smartphones verbunden.

Hochentwickelte Smartphone-Betriebssysteme wie Android von *Google* oder iOS von *Apple* unterstützen die Entwicklung eigener mobiler Anwendungen (*Apps*) durch die Bereitstellung entsprechender Systemschnittstellen. Damit bieten diese aktuellen Plattformen den Benutzern der Smartphones die Möglichkeit, neben den bereits vorinstallierten Systemanwendungen weitere mobile Anwendungen diverser Drittanbieter zu installieren und zu verwenden. Diese *Apps* werden dabei typischerweise aus den *App-Stores* der Betriebssystem- oder Geräte-Hersteller bezogen.

Somit besitzen heutige Smartphones durch die Kombination aus leistungsstarker Hardware, hochentwickelten Betriebssystemen und der Verwendung weiterer mobiler Anwendungen von Drittanbietern eine enorme Funktionsvielfalt. Im Vergleich zu herkömmlichen Computern weisen Smartphones jedoch auch einige hardwarebezogene Einschränkungen auf. Diese beziehen sich vor allem auf die begrenzte Energieversorgung durch den eingebauten Akku und die limitierte Bildschirmgröße. Bei der Entwicklung von mobilen Anwendungen ist es daher von zentraler Bedeutung, die spezifischen Restriktionen der Smartphones zu beachten. *Apps* sollten dementsprechend möglichst schonend mit den begrenzten Hardwareressourcen umgehen und zusätzlich aufgrund der großen Gerätevielfalt verschiedene Bildschirmgrößen und -auflösungen unterstützen können. Weitere Informationen werden dazu u. a. von Dunkel und Koschel in [12] gegeben.

Neben den Smartphones existiert die Gerätekategorie der Tablets. Diese unterscheiden sich vor allem anhand eines größeren Bildschirms von den typischen Smartphones. Aufgrund dessen kann gerade im geschäftlichen Umfeld die Arbeitsproduktivität durch die Verwendung solcher Geräte gesteigert werden. Die übrigen Hardwarespezifikationen stimmen größtenteils mit denen von Smartphones überein. Als Betriebssysteme kommen ebenfalls z. B. Android oder iOS zum Einsatz.

2.1.2 Mobile Plattformen

Wie von der Webseite der *International Data Corporation* (IDC) [22] entnommen werden kann, stellt das Android-Betriebssystem von *Google* die derzeit führende Plattform im Smartphone- und Tablet-Markt dar. Demnach wurden bereits im zweiten Quartal 2014 84,7% aller weltweit ausgelieferten Smartphones oder Tablets mit der Android-Plattform betrieben. Weitere Prognosen für das zweite Halbjahr 2014 weisen zudem darauf hin, dass sich diese dominierende Marktstellung noch weiter verstärken wird. Im Gegensatz dazu besitzt das iOS-Betriebssystem von *Apple* zurzeit einen Marktanteil von lediglich 11,7% mit anhaltend sinkender Tendenz, obwohl die Auslieferungsmenge von *iPhones* und *iPads* im ersten Halbjahr 2014 durch *Apple* sogar gesteigert werden konnte [22]. Somit teilen sich Android und iOS den Markt für mobile Plattformen mit zusammen über 96% Absatzanteil untereinander auf. Andere mobile Betriebssysteme wie z. B. Windows Phone von *Microsoft* oder BlackBerry OS von *RIM* besitzen eine entsprechend schwache Marktstellung im unteren einstelligen Prozentbereich. Abbildung 2.1 zeigt einen Überblick über die weltweiten Marktanteile mobiler Plattformen im zweiten Quartal 2014.

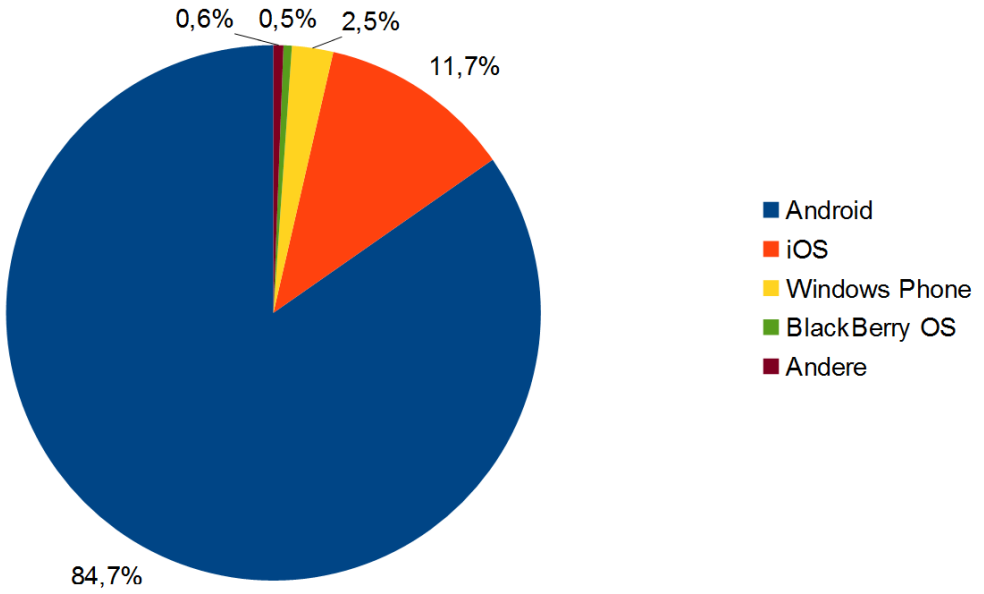

Abbildung 2.1: Weltweite Marktanteile mobiler Plattformen im zweiten Quartal 2014

Aufgrund der rasanten Weiterentwicklung von Smartphones und Tablets in den vergangenen Jahren hat sich auch der Verwendungskontext dieser Geräte stark gewandelt. Leistungstärkere Hardware ermöglicht inzwischen die Unterstützung für beinah sämtliche Anwendungskategorien bis hin zu aufwendigen Simulationen oder Spielen auf solchen mobilen Geräten. Die anfänglich verbreitete Nutzung mobiler Webanwendungen mit Hilfe des Smartphone-Webbrowsers wird heute in weiten Teilen durch die Verwendung entsprechender *Apps* ersetzt.

Im Gegensatz zu Webanwendungen werden *Apps* direkt auf den mobilen Geräten installiert. Dadurch können sie spezielle Betriebssystem- und Geräteeigenschaften (u. a. die Kamera, Sensoren, externe Speichermedien oder andere installierte Anwendungen) nutzen. Heutige Betriebssysteme besitzen bereits bei der Auslieferung eine ganze Fülle vorinstallierter *Apps* wie z. B. die Telefonanwendung, einen Kalender oder ein Kontaktbuch. Um die Funktionalität der Smartphones und Tablets an die eigenen Bedürfnisse anzupassen, können verschiedene weitere *Apps* von Drittanbietern über sogenannte *App-Stores* bezogen werden. Bis Mitte 2013 wurden im offiziellen Android *App-Store Google Play* weltweit bereits mehr als 50 Milliarden *App*-Downloads gezählt. *Google Play* umfasst inzwischen mehr als 50 verschiedene *App*-Kategorien und mehr als eine Million Anwendungen. Etwa 70% der *Apps* werden dabei kostenfrei angeboten. Andere Anwendungen können gegen ein üblicherweise geringes Entgelt erworben werden. Studien haben allerdings ergeben, dass fast jeder zweite Smartphone-Nutzer ausschließlich kostenfreie Anwendungen verwendet [12].

2.2 Android

Zu Beginn werden in diesem Abschnitt die grundlegenden Eigenschaften der Android-Plattform vorgestellt. Im Anschluss daran erfolgt eine Übersicht über die Betriebssystem-Architektur und die speziellen Komponenten einer Android-Anwendung. Bezogen auf die Motivation dieser Arbeit (siehe Abschnitt 1.1) werden die Sicherheitsmechanismen der Android-Plattform zum Schutz vor mobiler Malware erläutert. Weiterhin werden im Hinblick auf die Entwicklung eines IF-MAP Clients grundlegende Strategien zum Auslesen bestimmter Sensor-, System-, Kontext- und Anwendungsinformationen sowie im Umgang mit *Services* und *AsyncTasks* vorgestellt.

2.2.1 Überblick

Android ist ein Softwarestack, der für mobile Geräte wie Smartphones und Tablets entwickelt wurde. Dieser beinhaltet neben dem eigentlichen Betriebssystem auch eine Software-Plattform und das Android Software Development Kit (Android SDK) für die Entwicklung entsprechender mobiler Anwendungen.

Das Android-Betriebssystem basiert auf einem modifizierten Linux-Kernel, der u. a. für die Prozess- und Speicherverwaltung, die Netzwerkkommunikation und das Energie-Management zuständig ist. Frühere Versionen der Android-Plattform basierten auf dem Linux-Kernel 2.6.x. Aktuellere Android-Betriebssysteme ab Version 4.0 bauen hingegen auf einem neueren Kernel der 3.x-Serie auf. Die Software-Plattform stützt sich wiederum auf die Programmiersprache *Java*.

Zur Ausführung von komplexen *Java*-Anwendungen auf den begrenzten Hardware-ressourcen wurde die *Dalvik Virtual Machine* (DVM) entwickelt. Jede Anwendung unter Android läuft dabei in einem eigenen Betriebssystem-Prozess ab. Anwendungen werden in ein spezielles Bytecode-Format (*Dalvik-Executable*) übersetzt, wodurch im Vergleich zu herkömmlichen JAR-Archiven deutlich Speicherplatz eingespart wird. Ab der Android-Version 4.4 wird die DVM durch *Android Runtime* (ART) ersetzt. Weitere Informationen dazu sind in der Android-Entwicklerdokumentation [20] zu finden.

Android wird als Open-Source Software unter der Aufsicht von *Google* entwickelt. Der offizielle *App-Store* der Android-Plattform ist *Google Play* (früherer *Android Market*). Außerdem ist es möglich, Anwendungen aus anderen Quellen zu installieren. Am 21. Oktober 2008 wurde Android offiziell veröffentlicht. Seitdem erschienen zahlreiche Plattform-Updates mit diversen Erweiterungen und Verbesserungen. Mit der Version 3.0 (Honeycomb) unterstützte das Android-Betriebssystem auch erstmals explizit Tablets. Aktuell liegt Android in der Version 4.4.4 (KitKat) vor.

2.2.2 Architektur

Die grundlegende Architektur der Android-Plattform ist in fünf logischen Schichten organisiert, die im Folgenden überblicksartig vorgestellt werden. Abbildung 2.2 zeigt dazu den schematischen Aufbau der Android-Architektur mit den wichtigsten Komponenten.

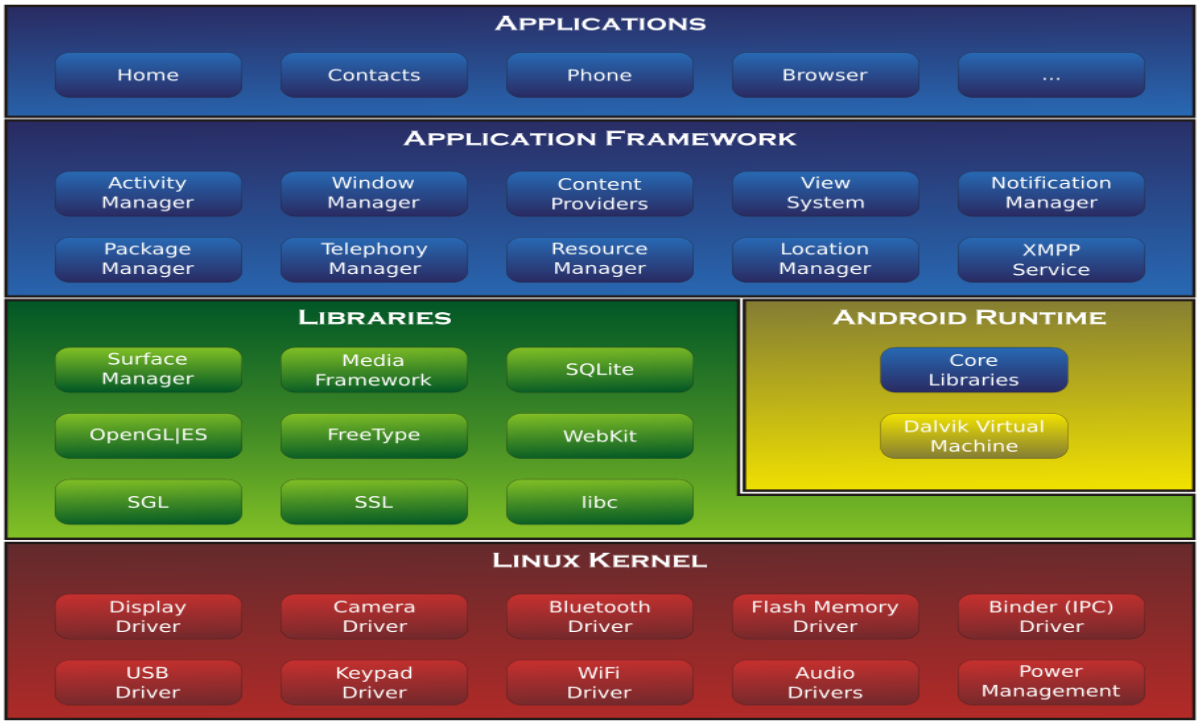

Abbildung 2.2: Logische Schichten der Android-Architektur, Quelle: [12]

Applications

Diese Schicht bildet alle auf dem Gerät installierten Applikationen ab. Dazu gehören einerseits Systemanwendungen (z. B. Webbrowser, Kalender oder Kontaktbuch), die als Grundausstattung auf jedem Android-Gerät vorinstalliert sind. Andererseits ordnen sich nachträglich installierte Anwendungen aus dem *App-Store* ebenfalls in diese Schicht ein.

Application Framework

Diese Schicht beinhaltet sämtliche Systemschnittstellen, die verwendet werden können, um Zugang zu bestimmten Systemdiensten zu erhalten. Dazu zählen z. B. der *Location Manager* zur Ermittlung der aktuellen Position oder das *View System*, um Elemente der Benutzeroberfläche wie beispielsweise *Buttons* anzusprechen.

Libraries

In dieser Schicht sind verschiedene Bibliotheken wie z. B. *SQLite*, *SSL* oder *libc* aufgeführt. Diese können wiederum über das *Application Framework* verwendet werden.

Android Runtime

Diese Schicht enthält die *Java Core Libraries* und die *Dalvik Virtual Machine* (DVM). Aus den *Java Core Libraries* wurden einige Komponenten für den Einsatz unter An-

droid entfernt, da sie entweder keine Verwendung fanden oder Performanz-Probleme verursachten. Zudem unterbindet der Compiler das Hinzufügen solcher Bibliotheken. Weitere Informationen zur DVM sind in Abschnitt 2.2.1 aufgeführt.

Linux Kernel

Wie ebenfalls in Abschnitt 2.2.1 beschrieben wird, basiert das Android-Betriebssystem auf einem Linux-Kernel. Durch diverse Treiber bildet er das Bindeglied zwischen der zugrundeliegenden Hardware und den übrigen Schichten der Android-Architektur.

Komponenten einer Android-Anwendung

Aufbauend auf die Android-Entwicklerdokumentation [20] werden nachfolgend die Bestandteile einer Android-Anwendung erläutert. Android-*Apps* werden in *Java* programmiert und bestehen dabei aus den folgenden Hauptkomponenten:

- *Activity*: Repräsentation eines einzelnen Fensters der Benutzeroberfläche; jede Applikation muss mindestens eine *Activity* als Einstiegspunkt besitzen
- *Service*: Hintergrundprozess ohne grafische Benutzeroberfläche für langlaufende Operationen (z. B. Netzwerkzugriffe)
- *Content Provider*: Informationsspeicher (z. B. das Dateisystem oder eine *SQLite*-Datenbank)
- *Broadcast Receiver*: Listener für systemweite Broadcast-Nachrichten des Betriebssystems oder von Drittanbieter-Anwendungen

Durch diese Komponenten sind Android-*Apps* in sich lose gekoppelt. Daher müssen sogenannte *Intents* als asynchrone Nachrichten verwendet werden, um mit anderen Anwendungskomponenten (*Activities*, *Services* oder *Broadcast Receiver*) zu kommunizieren. Dazu existieren einerseits explizite *Intents*, bei denen Nachrichten an eine spezifische, bekannte Komponente gesendet werden und andererseits implizite *Intents*, bei denen Nachrichten an einen bestimmten Komponententyp geschickt werden. Mittels *Intents* können auch zusätzliche Daten (*Payload*) an andere Komponenten übertragen oder Ergebnisse an die aufrufende Anwendung zurückgeliefert werden.

Neben dem *Java*-Quellcode besitzt eine Android-Anwendung weitere Ressourcen in Form von XML[10]-Dokumenten. Diese definieren dabei u. a. das Layout von *Activities* sowie Beschriftungen und Farben. Die Komponenten der Benutzeroberfläche (*Views*) werden deklarativ beschrieben und mit Hilfe von Layouts (z. B. *RelativeLayout*) entsprechend angeordnet. Durch die strikte Trennung zwischen der Deklaration der Benutzeroberfläche in Form von XML-Dokumenten und dem übrigen *Java*-Quellcode, weisen Android-Anwendungen diesbezüglich einen hohen Grad an Wartbarkeit und Änderbarkeit auf. Für jede definierte Ressource generiert das Android SDK

[10] Extensible Markup Language

automatisch eine eindeutige ID, über die die Ressource im *Java*-Quellcode verwendet werden kann.

Besondere Bedeutung kommt der Manifest-Datei als fester Bestandteil einer jeden Android-*App* zu. Sie liegt ebenfalls in Form eines XML-Dokuments vor und spezifiziert die gesamte Struktur der Anwendung. Dabei werden alle Komponenten der Applikation (*Activities*, *Services*, *Broadcast Receiver* und *Content Provider*) mit ihren Fähigkeiten (z. B. über *Intent-Filter*) deklariert. Zusätzlich muss das mindestens erforderliche Android API-Level angegeben werden. Weiterhin werden hier die benötigten Systemberechtigungen (*Permissions*), Hardware-Eigenschaften (z. B. NFC) und Bibliotheken (z. B. die *Google Maps Library*) deklariert.

Jede *Activity* durchläuft während ihrer Ausführung einen spezifischen Lebenszyklus. *Activities* können bestimmte *Hook*-Methoden implementieren, die während des Lebenszyklus vom Android-System aufgerufen werden. Detailierte Informationen zur Entwicklung von Android-Anwendungen sind u. a. auch von Dunkel und Koschel in [12] oder von Vogel in [35] zu finden.

2.2.3 Sicherheitsmechanismen

Die in Abschnitt 2.1.2 aufgezeigte Popularität von Smartphones und Tablets mit dem Android-Betriebssystem führt unweigerlich auch dazu, dass solche Geräte stärker in den Fokus von Cyber-Kriminellen rücken. Vor allem die Möglichkeit, zusätzliche Anwendungen von Drittanbietern zu installieren, erhöht die Gefahr von Bedrohungen einerseits für die mobilen Geräte selbst und andererseits für die zugrundeliegenden IT-Infrastrukturen. Aus diesem Grund existieren für die Android-Plattform einige Sicherheitsmechanismen, die im Folgenden vorgestellt werden.

Virenprüfungen im App-Store

Seit Anfang 2012 werden *Apps* im offiziellen *App-Store Google Play* automatisch auf Malware geprüft. Trotzdem wurde bereits Malware über den *App-Store* verbreitet.

Sandboxing

Unter Android werden Anwendungen von Drittanbietern in sogenannten *Sandboxes* isoliert verwaltet. Dadurch wird verhindert, dass solche *Apps* einerseits willkürlich auf andere Anwendungen zugreifen und andererseits unkontrolliert bestimmte Ressourcen des mobilen Geräts verwenden können. Jede Android-Anwendung läuft daher in einem eigenen Betriebssystem-Prozess bzw. in einer eigenen Instanz der DVM ab. Zusätzlich besitzt jede Applikation ihr eigenes, privates Verzeichnis im Dateisystem, auf das nur die entsprechende Anwendung zugreifen kann.

Systemberechtigungen

Android implementiert ein Modell von Systemberechtigungen (*Permissions*), um Anwendungen den Zugriff auf Ressourcen zu ermöglichen, die sich außerhalb ihrer *Sandbox*

befinden (siehe oben). Damit eine *App* z. B. bestimmte Sensoren nutzen, mit dem Internet kommunizieren oder auf Komponenten anderer Systemanwendungen zugreifen kann, benötigt sie die dafür entsprechenden Berechtigungen. Android definiert dazu zahlreiche standardisierte *Permissions*, die den Zugriff auf bestimmte Ressourcen steuern. Darüber hinaus können auch eigene *Permissions* definiert werden. Unter Android werden die Berechtigungen zum Zeitpunkt der Installation an eine Anwendung vergeben. Die Berechtigungen werden abhängig von ihrem potentiellen Risiko und der Art ihrer Vergabe in die folgenden vier Kategorien eingeteilt [5]:

- *Normal Permissions*: Berechtigungen mit geringem Risiko, die eine *App* ohne Zustimmung der Benutzer verwenden kann
- *Dangerous Permissions*: Berechtigungen, die Zugriff auf sensible Benutzerdaten erlauben oder Kosten verursachen können und daher die Zustimmung der Benutzer benötigen
- *Signature Permissions*: Berechtigungen, die nur dann vergeben werden, wenn die *App* mit dem Zertifikat der Anwendung signiert wurde, die die Berechtigung deklariert hat
- *SignatureOrSystem Permissions*: Berechtigungen, die ausschließlich an *Apps* vergeben werden, die standardmäßig im Betriebssystem vorinstalliert sind oder das gleiche Zertifikat dieser Systemanwendungen besitzen

Bei der Installation kann der Benutzer entweder allen *Permissions* einer *App* zustimmen oder die Installation verweigern. Weiterhin ist es bei der großen Menge an verschiedenen Systemberechtigungen schwierig, die genaue Bedeutung einer einzelnen Berechtigung oder einer Kombination unterschiedlicher Berechtigungen zu verstehen.

Sicherheits-Anwendungen

Inzwischen bieten zahlreiche Anbieter Sicherheits-*Apps* mit teilweise guten Resultaten für die Android-Plattform an. Diese Lösungen fokussieren sich jedoch oftmals nur auf das zugrundeliegende Gerät. Eine Integration in die Sicherheitssysteme vorhandener IT-Infrastrukturen ist demnach mit solchen Produkten nicht möglich. Eine detailierte Untersuchung vorhandener Sicherheitslösungen findet in Kapitel 3 dieser Arbeit statt.

2.2.4 Android als Basis für einen IF-MAP Client

Die Android-Plattform kann als technische Basis für einen mobilen IF-MAP Client auf den entsprechenden Smartphones oder Tablets fungieren. In einem ersten Schritt müsste ein Client dazu in der Lage sein, diverse Sensordaten der mobilen Geräte auslesen zu können. Android unterstützt dabei drei verschiedene Sensor-Kategorien:

- *Bewegungssensoren*: Messen die Beschleunigungs- und Rotationskräfte
- *Umgebungssensoren*: Erfassen die Eigenschaften der Umgebung des mobilen Geräts

- *Positionssensoren*: Ermitteln die physikalische Position des mobilen Geräts

Diese Sensoren sind über das sogenannte *Sensor Framework* zugänglich. Um über Veränderungen einzelner Sensorwerte informiert zu werden, können mit Hilfe der `SensorManager`-Klasse verschiedene `SensorEventListener` als *Sensor Observer* registriert werden, deren *Hook*-Methoden `onSensorChanged()` und `onAccuracyChanged()` vom Android-System aufgerufen werden. Die Methode `onSensorChanged()` erwartet dabei einen Parameter vom Typ `SensorEvent`, der die folgenden Informationen enthält: den oder die Messwerte, den Sensortyp, die Messgenauigkeit und einen Zeitstempel. Sobald der entsprechende Sensor einen neuen Messwert ermittelt hat, wird die Methode angestoßen. Zudem können mit der `Sensor`-Klasse einzelne Sensoren angesprochen werden, um deren Eigenschaften (z. B. den Hersteller) auszulesen. Beim Registrieren eines `SensorEventListeners` als *Sensor Observer* sollte ein angemessenes Abfrageintervall gewählt werden, um das System oder den Akku nicht unnötig zu belasten [20].

Weitere System- und Anwendungsinformationen können über verschiedene Dienste und Schnittstellen des *Application Frameworks* der Android-Architektur (siehe Abbildung 2.2) bezogen werden. So sind beispielsweise diverse Telefoninformationen wie die IMEI[11] oder die IMSI[12] über den `TelephonyManager` verfügbar. Anwendungsinformationen wie z. B. installierte *Apps* oder verwendete *Permissions* werden hingegen mit Hilfe des `PackageManagers` ermittelt. Viele Statusinformationen können durch *Broadcast Receiver* für bestimmte System-Ereignisse (z. B. ein eingehender Anruf) gewonnen werden, die durch das Android-System angestoßen werden. Andere dynamische Daten sollten periodisch z. B. mit Hilfe eines *TimerTasks* abgefragt werden. Auch hier muss ein angemessenes Zeitintervall entsprechend des Anwendungsfalls gewählt werden.

Ein jeweiliger IF-MAP Client sollte auf den mobilen Geräten betrieben werden können, während diese für die eigentlichen geschäftlichen Tätigkeiten genutzt werden. Dementsprechend würde eine Client-Anwendung die meiste Zeit im Hintergrund arbeiten, wodurch der Einsatz eines *Services* für die *Datensammlung*, *-verarbeitung* und *-übertragung* nötig wird. Somit kann der Client auch dann weiterlaufen, wenn die zugrundeliegenden Smartphones und Tablets für andere Zwecke eingesetzt werden und keine Benutzerinteraktion mit der Client-Anwendung besteht. Dazu besitzen *Services* einen eigenen Lebenszyklus, der nicht an den der aufrufenden Komponente gebunden ist. *Services* können aus anderen Anwendungskomponenten (z. B. *Activities* oder weiteren *Services*) mit Hilfe von *Intents* gestartet oder gestoppt werden (siehe Abschnitt 2.2.2). Ähnlich zu den *Activities* besitzen *Services* mehrere *Hook*-Methoden, die während des Lebenszyklus vom Android-System aufgerufen werden. Für die Kommunikation mit dem MAP-Server sind Netzwerkzugriffe innerhalb eines mobilen IF-MAP Clients erforderlich, die nicht im *Main-Thread* ausgeführt werden dürfen, um das Blockieren der Benutzeroberfläche durch potentiell langlaufende Netzwerkoperationen zu verhindern. Da ein *Service* selbst im *Main-Thread* einer Anwendung läuft, sollten Netzwerkzugriffe in *AsyncTasks* ausgelagert werden, die die Netzwerkkommunikation schließlich in gesonderten *Threads* durchführen.

[11] International Mobile Equipment Identity
[12] International Mobile Subscriber Identity

2.3 CEP

Dieser Abschnitt bietet zunächst eine Einführung in die Grundkonzepte einer *Event-Driven Architecture* (EDA) und in das *Complex Event Processing* (CEP). Im Anschluss daran werden *Ereignismodelle* erläutert und die Verarbeitung sowie die Behandlung von Ereignissen analysiert. Abschließend erfolgt eine Übersicht über die im entwickelten IF-MAP Client verwendete, mobile *Event Processing Engine Asper*.

2.3.1 Grundideen einer EDA

Wie Bruns und Dunkel in [8] erläutern, bildet die *Event-Driven Architecture* (EDA) einen Architekturstil, um Anwendungssysteme durch das zentrale Konzept der *Ereignisverarbeitung* effizienter und agiler zu gestalten. Prinzipiell könnten dabei sämtliche Vorkommnisse und Aktivitäten zur Änderung eines Zustands innerhalb einer Anwendungsdomäne als ein Ereignis aufgefasst werden. Abhängig vom fachlichen Wissensgehalt (z. B. ein einfacher Messwert oder ein komplexer Sachverhalt) liegen die Ereignisse auf verschiedenen *Abstraktionsstufen* vor. In einer EDA wird der Programmablauf letztlich durch das Auftreten von Ereignissen gesteuert. Die Kommunikation zwischen den Komponenten einer EDA erfolgt durch den Austausch von *Ereignisobjekten*. Ereignisgesteuerte Anwendungssysteme umfassen dabei die folgenden drei logischen Schichten und Basisaufgaben:

- *Ereignisquellen*: Relevante Ereignisse der Anwendungsdomäne unmittelbar nach dem Auftreten erkennen und passende *Ereignisobjekte* generieren (Abschnitt 2.3.3)
- *Ereignisverarbeitung*: Ereignisse mehrerer Quellen mittels CEP *filtern*, *anreichern*, *aggregieren*, *korrelieren* und auf *Ereignismuster* abgleichen (Abschnitt 2.3.4)
- *Ereignisbehandlung*: Zeitnahe Reaktionen auf gefundene *Ereignismuster*, indem z. B. Dienste der nachgelagerten Anwendungen angestoßen werden (Abschnitt 2.3.5)

Unmittelbar nach dem Auftreten eines Ereignisses senden die *Ereignisquellen* ein entsprechendes *Ereignisobjekt* an einen Mediator (z. B. eine Message-oriented Middleware). Dieser übernimmt die Weiterleitung des *Ereignisobjekts* an bestimmte *Ereignissenken* zur weiteren Verarbeitung. Die *Ereignisobjekte* selbst besitzen dabei keine Verarbeitungslogik.

Durch die asynchrone Kommunikation der beteiligten Komponenten werden die *Ereignisquellen* nicht durch die weitere Verarbeitung der *Ereignisobjekte* blockiert. Dazu liegt ereignisgesteuerten Systemen typischerweise eine *Publish/Subscribe*-Kommunikation zugrunde. Die *Ereignisquellen* publizieren ihre *Ereignisobjekte* an den Mediator, wobei die *Ereignissenken* wiederum bestimmte *Ereignistypen* vom Mediator abonnieren können. Grundsätzlich bestimmen die *Ereignisquellen* den Zeitpunkt, an dem ein *Ereignisobjekt* an den Mediator geschickt werden soll (*Push-Modus*).

Durch den Ereignisaustausch über einen Mediator sind sämtliche Komponenten einer EDA sehr lose gekoppelt. Dies führt zu einem hohen Grad an Wartbarkeit, Skalierbarkeit und Robustheit gegenüber Änderungen. Die *Ereignisverarbeitung* mittels einer

Event Processing Engine (siehe Abschnitt 2.3.4) erhöht zudem die Effizienz, Agilität und Aktualität des Anwendungssystems. Die asynchrone Kommunikation sowie die *Ereignisverarbeitung* durch eine CEP-Komponente bilden jedoch einen teilweise undurchsichtigen Kontrollfluss, der eine Fehlersuche unter Umständen erheblich erschweren könnte.

2.3.2 Grundkonzepte zu CEP

Seeger zeigt in [24], dass *Complex Event Processing* (CEP) eine innovative Technologie ist, mit der eine große Anzahl eintreffender, komplexer Ereignisse gleichzeitig und dynamisch verarbeitet werden kann. Die Ereignisse werden dabei in *kausale*, *temporale* oder *räumliche* Beziehungen gesetzt, aus denen wiederum spezifische *Ereignismuster* abgeleitet werden können, die eine fachliche Relevanz für die Anwendungsdomäne darstellen. Durch entsprechende *Event Processing Engines* können die eintreffenden Ereignisse in Echtzeit auf solche *Muster* untersucht werden (*Event Pattern Matching*), um den Programmablauf anhand fachlicher Daten der Ereignisse zu beeinflussen. Beim CEP (bzw. ESP[13]) treffen die Ereignisse als kontinuierlicher Strom ein. Die Regeln zur *Mustererkennung* für diese Ereignisströme werden hingegen persistent gespeichert. Viele EDAs enthalten eine zentrale CEP-Komponente (z. B. einen *Event Processing Agent* (EPA)), die die *Ereignisverarbeitungsschicht* der EDA bildet (siehe Abschnitt 2.3.1). Nach Bruns und Dunkel [8] bietet ein EPA dabei folgende Funktionen und Eigenschaften:

- Identifikation, Verarbeitung und Auswertung von Ereignissen in Echtzeit
- Suche nach einfachen und komplexen *Ereignismustern* über lange Zeiträume
- *Anreicherung* der Ereignisdaten mit *Kontextwissen* der Anwendungsdomäne
- Abstraktion von vielen einfachen Einzelereignissen zu einem komplexen Ereignis
- Verarbeitung von kontinuierlichen Ereignisströmen mehrerer Quellen
- Fachliches Wissen ist deklarativ in Form von *Wenn-Dann-Regeln* formuliert

Ein EPA basiert, wie in Abbildung 2.3 dargestellt ist, auf einem *Ereignismodell*, *Ereignisregeln* und einer *Event Processing Engine*. Das *Ereignismodell* spezifiziert alle relevanten *Ereignistypen* der Anwendungsdomäne mit ihren Attributen und Beziehungen untereinander (siehe Abschnitt 2.3.3). Basierend auf dem *Ereignismodell* werden *Ereignisregeln* anhand der fachlichen Anwendungslogik zur Verarbeitung der Ereignisse erstellt. Dadurch können Ereignisse *transformiert*, *gefiltert*, *aggregiert* oder *korreliert* werden. Weiterhin ist es möglich, Ereignisse mit *Kontextwissen anzureichern*, neue *Ereignisobjekte* zu generieren (*Kaskadieren von Ereignissen*) und die eintreffenden Ereignisströme auf bestimmte *Muster* zu untersuchen (siehe Abschnitt 2.3.4). Eckert und Bry zeigen dazu in [13] auf, dass zur Definition dieser *Ereignisregeln* spezielle *Event Pattern Languages* (EPLs) bzw. *Continuous Query Languages* (CQLs) existieren. Die

[13] *Event Stream Processing*; Legt den Fokus auf die Verarbeitung kontinuierlicher Ereignisströme

zentrale Komponente eines EPAs bildet die *Event Processing Engine* in Form eines Regelinterpreters. Die kontinuierlichen Ereignisströme werden dabei mit dem Bedingungsteil der *Ereignisregeln* abgeglichen. Bei einer Übereinstimmung wird der Aktionsteil der entsprechenden *Ereignisregel* ausgeführt.

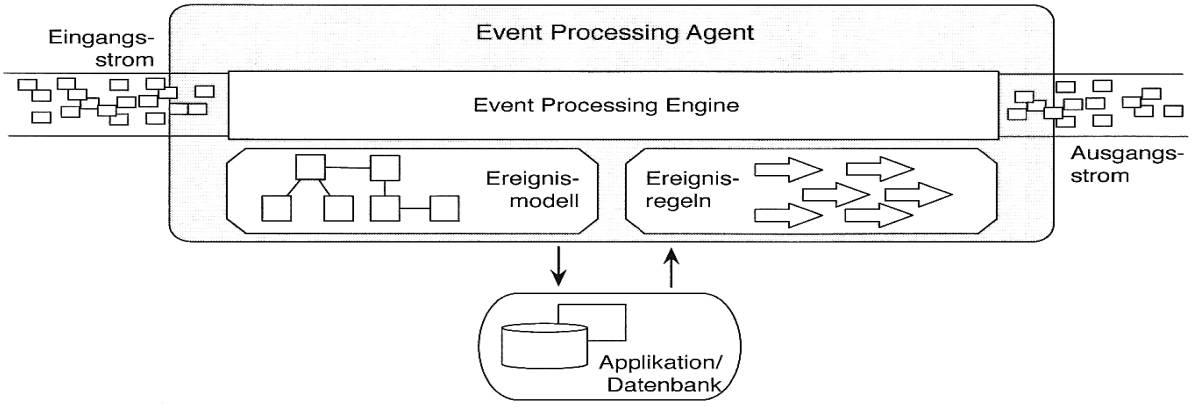

Abbildung 2.3: Elemente eines *Event Processing Agents*, Quelle: [8]

Um die Komplexität der *Ereignisverarbeitung* zu reduzieren, ist der Aufbau eines *Event Processing Networks* (EPN) sinnvoll. Die Verarbeitung der Ereignisse wird dabei in mehrere klar umrissene Teilschritte mit jeweils wenigen *Ereignisregeln* und einfachen *Ereignismustern* zerlegt. Jeder Verarbeitungsschritt wird durch einen eigenen, leichtgewichtigen EPA realisiert. Schließlich werden alle EPAs in einem EPN miteinander verknüpft, wobei die Kommunikation zwischen den EPAs durch den Austausch von *Ereignisobjekten* über sogenannte *Ereigniskanäle* realisiert wird (z. B. mit einer Message-oriented Middleware). EPAs besitzen dazu *Eingangs-* und *Ausgangskanäle* zum Empfangen bzw. Senden von *Ereignisobjekten*. In einem typischen EPN werden zunächst *Filter*-EPAs eingesetzt, die alle unbedeutsamen Ereignisse aus den Ereignisströmen herausfiltern. Erst danach erfolgt die weitere, schrittweise Verarbeitung in *Anreicherungs-*, *Aggregations-* und *Korrelations*-EPAs. Dadurch entsteht ein teilweise sequenzieller, transparenter Kontrollfluss und die physikalische Verteilung einzelner CEP-Komponenten wird ermöglicht. Der wesentliche Vorteil im Aufbau eines EPNs besteht darin, die fachliche Komplexität in Form von *Ereignisregeln* auf mehrere EPAs zu verteilen, sodass jeder EPA nur wenige, fachlich eng verknüpfte *Ereignisregeln* umfasst. Dadurch besitzt jeder EPA eine verständliche und klar definierte Teilaufgabe im Verarbeitungsprozess (starke Kohäsion) und ist aufgrund der Kommunikation über *Ereigniskanäle* mit den übrigen EPAs lose gekoppelt. Für detailierte Informationen zu EPNs sei hier abermals auf Bruns und Dunkel [8] hingewiesen.

2.3.3 Ereignismodelle

Ereignisse bilden die datentragenden Elemente in einer EDA. Abhängig vom fachlichen Wissensgehalt liegen sie auf verschiedenen *Abstraktionsstufen* vor. Mittels CEP

können einfache Einzelereignisse mit einem begrenzten Informationswert *gefiltert*, *angereichert*,*aggregiert* oder *korreliert* werden, um komplexe Ereignisse eines höheren *Abstraktionsniveaus* zu generieren, die die fachliche Bedeutung der Kombination dieser zuvor aufgetretenen Einzelereignisse repräsentieren. Abbildung 2.4 zeigt, wie dadurch die Menge der Ereignisse in jeder *Abstraktionsstufe* reduziert wird.

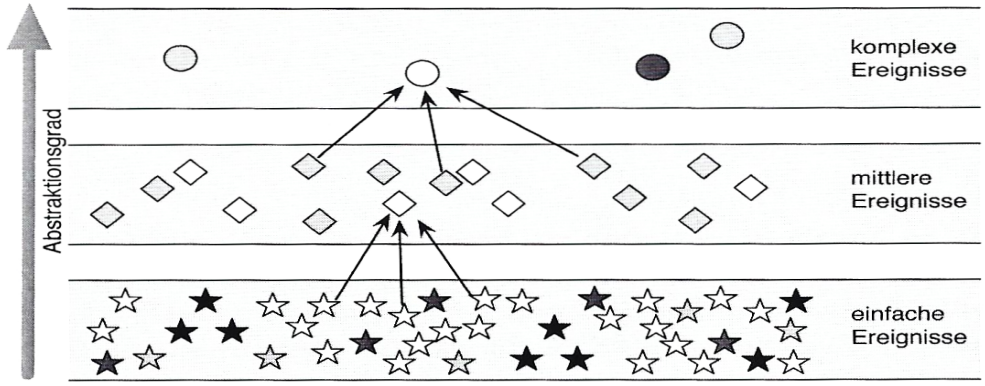

Abbildung 2.4: Abstraktion von Ereignissen über mehrere Stufen, Quelle: [8]

Ereignisse spezifizieren sämtliche Informationen, die für die Verarbeitung durch CEP notwendig sind. Einerseits enthalten sie dazu allgemeine, von der jeweiligen Anwendungsdomäne unabhängige Metainformationen wie z. B. eine ID, den *Ereignistyp*, die *Ereignisquelle* und einen Zeitstempel. Andererseits beinhalten sie spezifische Daten über den Ereigniskontext innerhalb der fachlichen Anwendungsdomäne wie z. B. bestimmte Messwerte. Ereignisse besitzen dabei keine zeitliche Dauer, sondern werden durch den Zeitpunkt ihres Auftretens definiert.

Ereignistypen definieren Klassen von Ereignissen mit gleichen Eigenschaften. *Ereignisobjekte* bilden wiederum konkrete Instanzen dieser *Ereignistypen* [8]. Dies ist vergleichbar mit der Beziehung zwischen einer Klasse und einem Objekt dieser Klasse in *Java*. Ähnlich zum Konzept der Vererbung in *Java*, können *Ereignistypen* in Hierarchien organisiert werden. Die Struktur eines *Ereignismodells* kann dabei u. a. mit Hilfe eines UML[14]-Klassenmodells entsprechend beschrieben werden. *Ereignisobjekte* können von diversen *Ereignisquellen* innerhalb eines ereignisgesteuerten Systems erzeugt werden und bilden einen potentiell unendlichen Ereignisstrom für die *Event Processing Engine*. Bei der Verarbeitung werden entsprechende Ausschnitte (*Sliding Windows*) dieser Ereignisströme betrachtet.

Für einen hohen Informationsgehalt müssen die Ereignisse in *kausale*, *zeitliche* oder *räumliche* Beziehungen zueinander gesetzt werden. Fachlich zusammenhängende Ereignisse können *aggregiert* oder *korreliert* werden, um komplexe Ereignisse anhand bestimmter *Ereignismuster* zu generieren. Ein komplexes Ereignis beschreibt dabei Erkenntnisse und Rückschlüsse, die durch das Auftreten der einzelnen Ereignisse gewonnen

[14]Unified Modeling Language

wurden. Dazu werden die einfachen Einzelereignisse anhand der fachlichen Anwendungslogik zu einem komplexen Ereignis abstrahiert (siehe Abschnitt 2.3.4).

Das *Ereignismodell* definiert alle fachlich relevanten *Ereignistypen* mit ihren Attributen, Beziehungen und Beschränkungen. Diese Einschränkungen beziehen sich dabei z. B. auf zulässige Wertebereiche für bestimmte Attribute. Sie werden in Form von *Filter*- oder *Korrektur*-Regeln umgesetzt, damit in den eigentlichen Verarbeitungsschritten ausschließlich konsistente *Ereignisse* betrachtet werden. Das *Ereignismodell* selbst besitzt keine Verarbeitungslogik, wodurch eine lose Kopplung zwischen den *Ereignisstrukturen* und der *Ereignisverarbeitung* erreicht wird.

2.3.4 Ereignisverarbeitung

Nach Bruns und Dunkel [8] besteht die Hauptaufgabe des CEP in der Verarbeitung von Ereignissen durch die Suche nach bestimmten *Mustern* in den Ereignisströmen. Dazu werden Ereignisse aus verschiedenen Quellen und unterschiedlichen *Abstraktionsstufen* analysiert. Eine CEP-Komponente stellt letztlich als *Ereignisverarbeitungsschicht* die entscheidende Einheit einer EDA dar (siehe Abschnitt 2.3.1). In vielen Fällen bildet erst ein *Muster* aus verschiedenen einfachen Einzelereignissen eine relevante Situation in der fachlichen Anwendungsdomäne ab. Einfache Ereignisse werden durch *Filterung*, *Anreicherung*, *Aggregation* und *Korrelation* in komplexe Ereignisse mit einem hohen Informationswert transformiert. Gleichzeitig reduziert sich dabei die Ereignismenge, wodurch insgesamt eine Datenverdichtung stattfindet (siehe Abbildung 2.4).

Zur Erkennung bestimmter Situationen werden *Muster* deklarativ durch spezielle Regelsprachen beschrieben [13]. Eine *Ereignisregel* setzt sich dabei aus einem Bedingungsteil und einem Aktionsteil zusammen. Im Bedingungsteil wird das *Muster* spezifiziert, nach welchem der Ereignisstrom durchsucht werden soll. Der Aktionsteil beschreibt, wie auf ein gefundenes *Muster* reagiert wird. Dabei können entweder neue Ereignisse erzeugt oder Dienste der nachgelagerten Anwendungssysteme angestoßen werden. Die *Event Processing Engine* dient dazu, die spezifizierten *Ereignisregeln* mit dem eintreffenden Ereignisstrom abzugleichen. Alle in den *Ereignismustern* verwendeten *Ereignistypen* müssen entsprechend im *Ereignismodell* definiert sein (siehe Abschnitt 2.3.3).

Ereignismuster spezifizieren eine bestimmte Sequenz von *Ereignisobjekten*, die im Ereignisstrom auftreten muss. Dazu existieren verschiedene Sprachkonstrukte, um *kausale*, *zeitliche* und *fachliche* Zusammenhänge zwischen den *Ereignisobjekten* zu definieren. Diese werden im Folgenden vorgestellt:

- *Ereignistypbasierte Muster*: Spezifikation der gesuchten *Ereignistypen* mit Hilfe von Sequenzoperatoren, Simultanoperatoren und booleschen Operatoren
- *Kontextbedingungen*: Bedingungen an die Attribute eines *Ereignisobjekts*
- Auswahl des *Ereignisobjekts*: Erfüllen mehrere *Ereignisobjekte* ein *Muster*, wird ein bestimmtes *Ereignisobjekt* für die *Ereignisregel* selektiert
- Verbrauch von *Ereignisobjekten*: Spezifikation, ob ein einzelnes *Ereignisobjekt* mehrere *Ereignismuster* auslösen kann

- *Sliding Windows*: Potentiell unendliche Ereignisströme werden durch Längen- und Zeitfenster limitiert, wobei Längenfenster die Anzahl der Ereignisse und Zeitfenster einen Zeitraum spezifizieren

- *Aggregatfunktionen*: Zusammenfassung von Ereignisattributen innerhalb eines *Sliding Windows* (z. B. Summe, Durchschnitt, Minimum, Maximum)

Sobald ein *Muster* erkannt wurde, wird der Aktionsteil der entsprechenden *Ereignisregel* ausgeführt. Einerseits können dabei neue Ereignisse generiert werden, die wiederum in den Ereignisstrom eingefügt werden (*Kaskadieren von Ereignissen*). Andererseits können aber auch Dienste der Anwendungssysteme aufgerufen werden (siehe Abschnitt 2.3.5). Neue Ereignisse entstehen durch *Translation*, *Filterung*, *Aggregation* oder *Splitting*, ohne dass weitere Informationen ergänzt werden müssen. Weiterhin ermöglichen die *Anreicherung* mit *Kontextwissen* und die *Synthese* komplexer Ereignisse die Generierung neuer Ereignisse, die einen höheren Informationsgehalt als die zugrundeliegenden Ereignisse besitzen.

Ereignisse werden durch die *Translation* in ein anderes Format übersetzt. Die *Filterung* selektiert die für die fachliche Anwendungsdomäne relevanten Ereignisse und reduziert somit die Anzahl der verbleibenden Ereignisse. Auch die *Aggregation* verringert die Ereignismenge, indem mehrere einfache Einzelereignisse zu einem bedeutsameren *Ereignisobjekt* zusammengefasst werden. Durch *Splitting* entstehen aus einem *Ereignisobjekt* mehrere neue Einzelereignisse. Bei der *Anreicherung* mit *Kontextwissen* beziehen die *Ereignisobjekte* zusätzliche Informationen aus den Anwendungssystemen. Die *Synthese* komplexer Ereignisse generiert neue, komplexe Ereignisse als eine Abstraktion von bestimmten in Beziehung stehenden, einfachen Ereignissen, die einen Erkenntnisgewinn repräsentieren. Zu sämtlichen in den *Ereignisregeln* erzeugten *Ereignisobjekten* müssen die entsprechenden *Ereignistypen* im *Ereignismodell* definiert sein (siehe Abschnitt 2.3.3). Zur Behandlung eines bestimmten *Ereignismusters* werden z. B. Dienste der nachgelagerten Anwendungssysteme angestoßen (siehe Abschnitt 2.3.5).

2.3.5 Ereignisbehandlung

Die *Ereignisbehandlung* ist klar von der *Ereignisverarbeitung* (siehe Abschnitt 2.3.4) abgegrenzt. Die Verarbeitung von Ereignisströmen durch entsprechende *Ereignisregeln* und die Erkennung von spezifischen *Mustern* erfolgt mit Hilfe einer CEP-Komponente. Die Reaktion auf gefundene *Ereignismuster* wird hingegen durch den Aktionsteil der *Ereignisregeln* in die nachgelagerten Anwendungssysteme delegiert. Dort können wiederum komplexe Algorithmen durchgeführt, mit Nachbarsystemen interagiert oder ablauforientierte Geschäftsprozesse angestoßen werden. Zur Integration einer *Event Processing Engine* existieren sogenannte *In-* und *Out-Adapter* als schmale Schnittstellen zwischen der *Ereignisverarbeitungsschicht* und den *Ereignisquellen* sowie der *Ereignisbehandlung*. Dadurch wird das interne Format der CEP-Komponente gekapselt und es besteht eine lose Kopplung zwischen den Schichten einer EDA (siehe Abschnitt 2.3.1).

2.3.6 CEP auf mobilen Geräten

In den vergangenen Jahren kam CEP hauptsächlich im Bereich des automatisierten Handelns in Finanzmarktsystemen auf leistungsstarken Servern zum Einsatz. Erst moderne Smartphones und Tablets mit ausgereifter Hard- und Software ermöglichen auch *Mobile Event Processing*. Inzwischen existiert mit *Asper* [3] eine mobile CEP-Komponente, die direkt auf Android-Smartphones und -Tablets u. a. zur ereignisbasierten Verarbeitung und Verdichtung von Sensor-, System-, Kontext- und Anwendungsinformationen der Android-Plattform innerhalb eines IF-MAP Clients eingesetzt werden könnte. *Asper* ist eine Portierung der leistungsfähigen und weit verbreiteten Open-Source CEP-Komponente *Esper* [16] für Android, wobei die Abhängigkeiten zu externen Bibliotheken, die unter Android nicht zur Verfügung stehen, eliminiert wurden. Trotzdem bietet *Asper* nahezu den gesamten Funktionsumfang von *Esper* an. Die EPL von *Asper* besitzt eine SQL[15]-ähnliche Syntax zur *Ereignisverarbeitung* durch die Definition von *Wenn-Dann-Ereignisregeln* mit entsprechenden *Ereignismustern*. Diese Regeln werden kontinuierlich gegen die eintreffenden Ereignisse ausgeführt (siehe Abschnitt 2.3.2).

Durch die Integration in *Java* werden *Ereignistypen* in *Asper* typischerweise als *Java*-Beans repräsentiert. Für einen *Asper*-EPA kann ein `EPRuntime`-Objekt generiert werden, das die `sendEvent()`-Methode anbietet, um dem EPA *Ereignisobjekte* zu übergeben.

Die EPL dient ausschließlich zur Definition von Regeln zur *Ereignisverarbeitung*. *Asper*-Regeln besitzen typische SQL-Elemente: Durch die *From*-Klausel können die zu betrachtenden *Ereignistypen* spezifiziert werden, die *Select*-Klausel selektiert bestimmte Ereignisattribute und in der *Where*-Klausel können spezifizierte *Ereignistypen* durch bestimmte Bedingungen weiter eingeschränkt werden. Zur Registrierung von *Asper*-Regeln wird für den jeweiligen EPA ein `EPAdministrator`-Objekt generiert und dessen `createEPL()`-Methode aufgerufen. Listener stellen wiederum den Aktionsteil einer *Ereignisregel* dar. Diese können durch die `addListener()`-Methode mit der *Asper*-Regel verknüpft werden. Listener implementieren eine `update()`-Methode, die jedes Mal aufgerufen wird, wenn das *Muster* des Bedingungsteils der verknüpften *Ereignisregel* im Ereignisstrom gefunden wird. Als Parameter wird dabei ein generisches `EventBean`-Array übergeben, dessen Daten entsprechend extrahiert und transformiert werden müssen.

Durch das Sprachkonstrukt `pattern[]` werden *ereignistypbasierte Muster* definiert. Dabei können Aliasnamen, der Sequenzoperator, *Join*-Operationen, *Kontextbedingungen* oder boolesche Operatoren verwendet werden (siehe Abschnitt 2.3.4). Durch den Bezeichner `every` kann der Verbrauch von *Ereignisobjekten* gesteuert werden. Weiterhin ist es in *Asper* möglich, *Sliding Windows* in Form von Längen- und Zeitfenstern zu spezifizieren. Dabei können diese Ausschnitte auch als *Batch-Windows* definiert werden, sodass eine Verarbeitung erst nach Ablauf der angegebenen Zeit (Zeitfenster) oder Eintreffen der entsprechenden Anzahl von Ereignissen (Längenfenster) durchgeführt wird. Durch sogenannte *Pattern Guards* sind *Sliding Windows* auch innerhalb des `pattern[]`-Konstrukts möglich. *Asper* bietet u. a. die *Aggregationsfunktionen* `sum()`, `avg()`, `min()` und `max()` an. Neben der *Ereignisbehandlung* in Form des Aufrufs von Listenern, können in *Asper*-Regeln auch neue Ereignisse über das Sprachkonstrukt `insert into` generiert werden.

[15] Structured Query Language; Anfrage-Sprache für relationale DBS

EPAs werden über die `getProvider()`-Methode des `EPServiceProviderManagers` erzeugt. EPNs lassen sich einrichten, indem mehrere EPAs verknüpft werden.

Durch diese ausgereifte EPL bietet *Asper* ein mächtiges Werkzeug, um diverse Sensor-, System-, Kontext- und Anwendungsinformationen von Android in Form von Ereignissen zu verarbeiten und komplexe Ereignisse mit einem hohen fachlichen Wissensgehalt zu erzeugen. Details zur vorgestellten EPL sind in der *Esper*-Dokumentation [17] zu finden.

2.4 IF-MAP

In diesem Abschnitt werden zunächst die Ziele der *Trusted Computing Group* (TCG) zusammengefasst und die Eigenschaften der *Trusted Network Connect* (TNC)-Architektur aufgezeigt. Im Anschluss daran erfolgt eine Einführung in das IF-MAP Kommunikationsprotokoll. Der Schwerpunkt liegt dabei auf der Analyse des Daten- und Kommunikationsmodells. Abschließend wird IF-MAP im Kontext des entwickelten Clients betrachtet. Dazu werden auch die von der *Trust@HsH*-Forschungsgruppe entwickelten Softwarekomponenten für das IF-MAP Umfeld vorgestellt.

2.4.1 Trusted Computing

Die vorliegende Arbeit gliedert sich fachlich primär in das Themengebiet der IT-Sicherheit ein. Die in dieser Arbeit entwickelte Lösung verwendet Softwarekomponenten aus dem IF-MAP Umfeld, die auf spezifizierten Standards der *Trusted Computing Group* (TCG) basieren. Im Folgenden werden daher zunächst die Ziele der TCG zusammengefasst.

In [30] beschreibt sich die TCG selbst als eine gemeinnützige Standardisierungsorganisation der Industrie. Dabei verfolgt sie das Ziel, einen offenen und herstellerunabhängigen Standard zum *Trusted Computing* zu entwickeln, um die Sicherheit in IT-Infrastrukturen auf verschiedenen Plattformen zu erhöhen. Seit dem Jahr 2003 setzt sie dazu die Standardisierungsarbeiten der ehemaligen *Trusted Computing Platform Alliance* (TCPA) fort. Als Ergebnis der Standardisierungsarbeiten entwickelt und veröffentlicht die TCG Spezifikationen zur Definition von Architekturen, Funktionen und Schnittstellen im Bereich des *Trusted Computing*. Auf dieser Basis können Entwickler Anwendungskomponenten für verschiedene Plattformen implementieren, um die Sicherheit von Computernetzwerken zu verbessern.

Aufgrund der vielfältigen Anwendungsgebiete im Bereich des *Trusted Computing* gliedert sich die TCG in verschiedene Untergruppen, die sich jeweils auf einen bestimmten Teilaspekt spezialisiert haben. Dazu zählt u. a. die *Trusted Network Connect Working Group* (TNC-WG), die Mechanismen entwickelt, um die Integrität von Endpunkten in Computernetzwerken zu kontrollieren. Daher wurde eine Architektur zur Integritätskontrolle in heterogenen Netzwerken mit dem Namen *Trusted Network Connect* (TNC) eingeführt, die in Abschnitt 2.4.2 näher erläutert wird. Für den entwickelten Client sind dabei vor allem die darin spezifizierten Komponenten *Metadata Access Point Client* und *Metadata Access Point Server* sowie das zum Austausch von *Metadaten* verwendete IF-MAP Protokoll von zentraler Bedeutung.

2.4.2 TNC-Architektur

Die TNC-WG hat zur Überprüfung der Integrität von Endpunkten in heterogenen Computernetzwerken die TNC-Architektur entwickelt. Die Spezifikation dazu ist in [31] frei verfügbar. Durch die TNC-Architektur können Unternehmensnetzwerke mit Hilfe spezifischer Richtlinien (sogenannten *Policies*) vor dem Zugriff von nicht-autorisierten Geräten oder Benutzern geschützt werden. Die TNC-Architektur bildet somit einen erweiterten Sicherheitsmechanismus gegenüber dem generellen IEEE 802.1X Standard, der lediglich zur Authentifizierung eines Endpunktes im Netzwerk dient [31]. Die erweiterte Funktionalität der TNC-Architektur ist immer dann von Bedeutung, wenn die reine Authentifizierung in einem sicherheitskritischen Unternehmensbereich unzureichend wäre und weitere Richtlinien wie z. B. die Aktualität des Betriebssystems oder des Virenscanners überprüft werden müssen. Nur wenn ein Endpunkt neben der reinen Authentifizierung auch diese zusätzlichen Kriterien erfüllt, wird ihm der Zugang zum Netzwerk gewährt. Damit wird das zugrundeliegende Netzwerk dynamisch vor Bedrohungen und Gefahren durch Endpunkte mit Sicherheitsproblemen geschützt. Wie in Abbildung 2.5 dargestellt ist, umfasst die TNC-Architektur mehrere Komponenten, die über verschiedene Kommunikationsprotokolle miteinander interagieren.

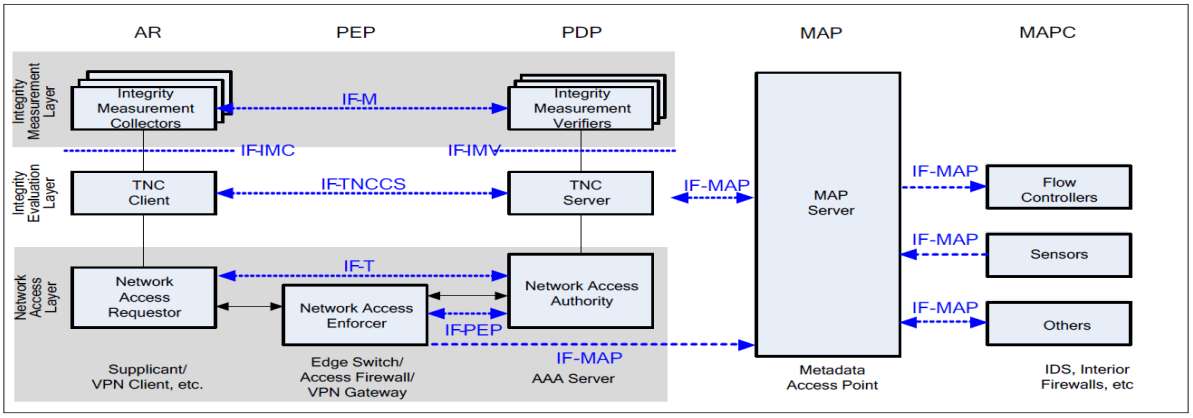

Abbildung 2.5: Aufbau der *Trusted Network Connect* Architektur, Quelle: [31]

Die TNC-Architektur gliedert sich in fünf vertikale Schichten, die die verschiedenen Verantwortlichkeiten der Architektur repräsentieren. Abhängig von ihrer Funktionalität kann eine Netzwerkkomponente dabei eine der folgenden Rollen innerhalb der TNC-Architektur besitzen [31]:

- *Access Requestor* (AR): Ein bestimmter Endpunkt, der Zugang zu einem durch TNC-Komponenten überwachten Netzwerk erfragt (z. B. ein Smartphone)
- *Policy Decision Point* (PDP): Eine Komponente innerhalb des geschützten Netzwerks, die anhand von *Policies* (z. B. Restriktionen für bestimmte Benutzer oder Softwarekonfigurationen) überprüft, ob dem AR der Zugang zum Netzwerk gestattet wird oder nicht

- *Policy Enforcement Point* (PEP): Eine Komponente, die die Entscheidungen des PDP umsetzt, indem sie den Netzwerkzugang für den AR freigibt oder sperrt
- *Metadata Access Point Server* (MAPS, MAP-Server): Eine zentrale Netzwerkdatenbank zur Speicherung diverser Statusinformationen in Form von *Metadaten*
- *Metadata Access Point Client* (MAPC, MAP-Client): Eine Komponente, die *Metadaten* an den MAP-Server publiziert, bestimmte *Metadaten* auf dem MAP-Server sucht oder diese von dem MAP-Server abonniert

Weiterhin unterteilt sich die TNC-Architektur in drei horizontale Schichten [31]. Im *Integrity Measurement Layer* (IML) werden Nachrichten zwischen dem AR und einem PDP ausgetauscht, um Integritätsinformationen über den AR zu sammeln. Der *Integrity Evaluation Layer* (IEL) dient zur Überprüfung der Integrität des AR anhand der Ergebnisse der IML-Schicht. Der *Network Access Layer* (NAL) umfasst gewöhnliche Sicherheits- und Authentifizierungsmechanismen wie z. B. den IEEE 802.1X Standard.

Im Kontext des entwickelten Clients bilden die MAP-Clients und der MAP-Server die entscheidenden Systeme innerhalb der TNC-Architektur. Von zentraler Bedeutung für die Entwicklung eines eigenen Clients ist demnach ein grundlegendes Verständnis über die Kommunikation zwischen diesen beiden Systemen mit Hilfe des IF-MAP Protokolls. Ein MAP-Client ist eine Netzwerkkomponente, die das IF-MAP Protokoll nutzt, um mit einem MAP-Server verschiedene Statusinformationen in Form von sogenannten *Metadaten* auszutauschen. Dabei kann ein MAP-Client für unterschiedliche Anwendungsfälle innerhalb der TNC-Architektur verwendet werden. Der MAP-Server bildet den zentralen Zugriffspunkt für alle MAP-Clients in der TNC-Architektur, die das IF-MAP Kommunikationsprotokoll zum Datenaustausch verwenden. Sämtliche *Metadaten*, die von den MAP-Clients publiziert werden, speichert der MAP-Server in einem speziellen Datenmodell. Detailierte Informationen hierzu werden in den Abschnitten 2.4.3, 2.4.4 und 2.4.5 erläutert.

2.4.3 IF-MAP Grundkonzepte

Das Grundkonzept des IF-MAP[16] Kommunikationsprotokolls besteht darin, Statusinformationen in Form von *Metadaten* zwischen verschiedenen Netzwerkkomponenten auszutauschen. Diese *Metadaten* können diverse Informationen, wie z. B. den Authentifizierungsstatus eines Geräts, repräsentieren. Alle *Metadaten* werden von den MAP-Clients an einen zentralen MAP-Server publiziert, der daraus wiederum ein spezielles Datenmodell in Form einer ungerichteten und ungewichteten Graphenstruktur aufbaut. Dabei setzt der MAP-Server die empfangenen *Metadaten* in semantische Beziehungen zueinander. Weitere Informationen zum IF-MAP Datenmodell sind in Abschnitt 2.4.4 zu finden.

Angelehnt an [33] wird nachfolgend ein Beispiel zur Funktionsweise des IF-MAP Protokolls aufgezeigt. Zunächst wird dabei die Statusinformation publiziert, dass sich der

[16] Interface for Metadata Access Points

Benutzer *XY* mit einem Gerät (MAC-Adresse: *AA:BB:CC:DD:EE:FF*) an einem IEEE 802.1X Switch authentifiziert hat. Ein DHCP-Server meldet daraufhin, dass dem entsprechenden Gerät die IP-Adresse *192.168.0.10* zugeteilt wurde. Der MAP-Server speichert diese *Metadaten* und setzt sie in seinem Datenmodell in Beziehung zueinander. Die Graphenstruktur des MAP-Servers repräsentiert schließlich die Situation, dass sich der Benutzer *XY* authentifiziert hat und mit ihm die MAC-Adresse *AA:BB:CC:DD:EE:FF* sowie die IP-Adresse *192.168.0.10* verknüpft sind. Alle Informationen, die das Datenmodell des MAP-Servers verwaltet, werden von entsprechenden MAP-Clients publiziert. Im aufgezeigten Beispiel stellen sowohl der IEEE 802.1X Switch als auch der DHCP-Server jeweils einen solchen MAP-Client dar.

MAP-Clients können auf unterschiedliche Weise mit dem MAP-Server interagieren. Einerseits können sie, wie im obigen Beispiel beschrieben, verschiedene *Metadaten* an den MAP-Server publizieren. Andererseits können MAP-Clients aber auch bestimmte *Metadaten* vom MAP-Server abonnieren oder explizit nach *Metadaten* im Datenmodell des MAP-Servers suchen. Abonniert ein MAP-Client bestimmte *Metadaten*, so werden die angeforderten Informationen erst dann vom MAP-Server übertragen, sobald diese zur Verfügung stehen. Insgesamt kann das Konzept des IF-MAP Protokolls als eine Plattform zum Austausch diverser Statusinformationen betrachtet werden. Der Datenaustausch zwischen den einzelnen MAP-Clients erfolgt dabei grundsätzlich über den MAP-Server als zentrale Netzwerkdatenbank zur Verwaltung sämtlicher Informationen. Dazu bietet der MAP-Server Funktionen zum Veröffentlichen, Abonnieren und Suchen von *Metadaten* an (siehe Abschnitt 2.4.5). Die Art und Anzahl der publizierten *Metadaten* hängt von der Implementierung des jeweiligen MAP-Clients ab. Der MAP-Server besitzt keine Möglichkeit, empfangene *Metadaten* auf Korrektheit, Aktualität und Konsistenz zu prüfen. Daher speichert er grundsätzlich alle Informationen in seinem Datenmodell. Die Gewährleistung, dass ausschließlich korrekte, aktuelle und konsistente *Metadaten* an den MAP-Server übermittelt werden, liegt somit vollständig auf der Seite der MAP-Clients. Detailierte Informationen dazu werden u. a. in [33] gegeben.

Durch das IF-MAP Protokoll werden *Metadaten* zwischen MAP-Clients und einem MAP-Server ausgetauscht. Die Hauptaufgaben eines MAP-Clients beziehen sich dabei, wie oben beschrieben, auf das Publizieren, Abonnieren und Suchen von *Metadaten* auf dem MAP-Server. Innerhalb der TNC-Architektur repräsentieren verschiedene Geräte, wie z. B. Firewalls oder DHCP-Server, einen MAP-Client (siehe Abschnitt 2.4.2). Auch Smartphones und Tablets können die Rolle eines MAP-Clients in der TNC-Architektur bilden, indem sie z. B. verschiedene Statusinformationen über das zugrundeliegende Gerät auf dem MAP-Server veröffentlichen. Für die Rolle eines MAP-Clients ergeben sich somit unterschiedliche Anwendungsfälle:

- Publizieren von *Metadaten*: Ein Sensor veröffentlicht einen neuen Messwert
- Abonnieren von *Metadaten*: Eine Firewall abonniert Sicherheitsdaten über ein bestimmtes Netzwerkgerät vom MAP-Server
- Suchen von *Metadaten*: Ein Smartphone sucht nach dem Standort eines anderen Smartphones auf dem MAP-Server

2.4.4 IF-MAP Datenmodell

Der MAP-Server speichert und verwaltet alle *Metadaten*, die von MAP-Clients publiziert werden, in einem speziellen Datenmodell, das in [33] von der TCG spezifiziert wurde. Dieses bildet die Form einer ungerichteten und ungewichteten Graphenstruktur, bestehend aus *Identifiern*, *Metadaten* und *Links*. Innerhalb dieser Graphenstruktur werden einzelne, von unterschiedlichen MAP-Clients veröffentlichte Statusinformationen in semantische Beziehungen zueinander gesetzt. Ein Beispiel für ein solches Datenmodell ist in Abbildung 2.6 dargestellt. Dabei sind die *Identifier* durch Ovale und die *Metadaten* durch Vierecke gekennzeichnet. *Links* bilden eine Verbindung zwischen je zwei *Identifiern*. Nachfolgend werden die einzelnen Bestandteile des Datenmodells näher erläutert.

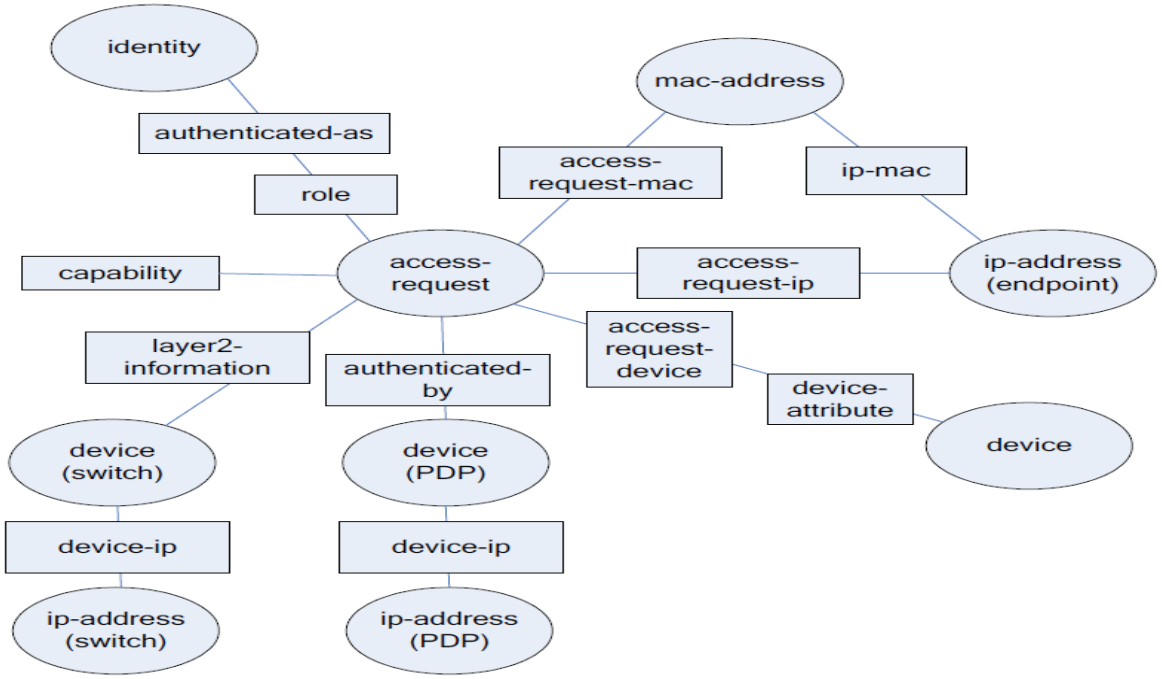

Abbildung 2.6: Beispiel für ein IF-MAP Datenmodell, Quelle: [33]

Identifier

Ein *Identifier* repräsentiert einen Knoten innerhalb der Graphenstruktur. In [32] wurden von der TCG fünf *Identifier* standardisiert: *ip-address*, *mac-address*, *identity*, *device* und *access-request*.

Metadaten

Die von den MAP-Clients publizierten *Metadaten* können beliebige Informationen enthalten. In Bezug auf die TNC-Architektur wurde von der TCG eine Spezifikation [32] veröffentlicht, die einige *Metadaten* zur Netzwerksicherheit standardisiert. Dazu gehören

u. a. die in Abbildung 2.6 dargestellten *Metadaten* wie beispielsweise *authenticated-as* oder *access-request-device*. Darüber hinaus können auch eigene *Metadaten*, z. B. für die speziellen Eigenschaften von Smartphones und Tablets, definiert werden. Die *Metadaten* können im Datenmodell abhängig von ihrem Typ entweder direkt mit einem *Identifier* oder mit einem *Link* zwischen zwei *Identifiern* verknüpft werden.

Links

Ein *Link* beschreibt eine Beziehung zwischen zwei *Identifiern* als ungerichtete und ungewichtete Verbindung. Die *Links* werden dabei grundsätzlich durch MAP-Clients erzeugt, sobald diese *Metadaten* veröffentlichen, die zwei bestimmte *Identifier* in Beziehung zueinander setzen. Die entsprechend publizierten *Metadaten* werden dann wiederum mit dem *Link* zwischen diesen beiden *Identifiern* verbunden. Ein DHCP-Server publiziert z. B. *Metadaten*, die IP- und MAC-Adressen in Beziehung zueinander setzen. Diese veröffentlichten *ip-mac-Metadaten* werden dann an den entsprechenden *Link* zwischen der jeweiligen IP- und MAC-Adresse angehängt.

2.4.5 IF-MAP Kommunikationsmodell

IF-MAP ist ein *Publish/Subscribe*-Kommunikationsprotokoll. *Metadaten* werden dabei in Form von XML[17]-Dokumenten als Nutzdaten im *SOAP*[18]*-Body* über HTTPS[19] ausgetauscht [33]. MAP-Clients und MAP-Server müssen sich bei dem Verbindungsaufbau gegenseitig authentifizieren. Die Kommunikation kann dabei über bis zu zwei Kanäle realisiert werden. Für den Anwendungsfall, dass ein MAP-Client lediglich eine Verbindung zum MAP-Server aufbaut, *Metadaten* publiziert oder nach Informationen im Datenmodell des MAP-Servers sucht, wird ausschließlich der *Synchronous Send Receive Channel* (SSRC) benötigt. Der *Asynchronous Receive Channel* (ARC) ist nur dann notwendig, wenn ein MAP-Client bestimmte *Metadaten* vom MAP-Server abonniert, da die Statusinformationen je nach Verfügbarkeit durch den MAP-Server an den MAP-Client übertragen werden. Für den Aufbau der IF-MAP Operationen hat die TCG in [33] ein *XML-Schema* spezifiziert. Wie aus [32] entnommen werden kann, existiert zu den von der TCG standardisierten *Metadaten* zur Netzwerksicherheit ebenfalls ein *XML-Schema*. Nachfolgend werden die Operationen des IF-MAP Protokolls im Einzelnen vorgestellt.

Operationen zum Verbindungsmanagement

Für den Verbindungsaufbau und den Verbindungsabbau existieren die Operationen *newSession* und *endSession*. Der Verbindungsaufbau wird grundsätzlich vom MAP-Client angestoßen. Je nach Anwendungsfall werden dann der SSRC oder der ARC benötigt. Mit *newSession* richtet der MAP-Client eine neue Sitzung mit dem MAP-Server ein.

[17] Extensible Markup Language
[18] Netzwerkprotokoll zum Datenaustausch zwischen Computersystemen
[19] HyperText Transfer Protocol Secure

Der MAP-Server antwortet dabei mit den beiden eindeutigen IDs *publisherID* und *sessionID*. Die *publisherID* kennzeichnet den eigentlichen MAP-Client, sodass dieser bei jedem Verbindungsaufbau die gleiche *publisherID* zugewiesen bekommt. Im Gegensatz dazu wird die *sessionID* für jede Sitzung neu generiert, sodass diese lediglich die aktuell verwendete Sitzung mit dem MAP-Server repräsentiert. Mit der Operation *endSession* wird eine bestehende Verbindung mit dem MAP-Server beendet, wobei der MAP-Server mit einem *endSessionResult* antwortet [33].

Operationen zum Datenaustausch

Zum Austausch von *Metadaten* bietet das IF-MAP Protokoll die folgenden Operationen an: *publish*, *subscribe*, *search*, *poll* und *purgePublish*. Bei einem Fehler während der Kommunikation wird vom MAP-Server ein *errorResult* erzeugt. Lediglich die *poll*-Operation benötigt den ARC, da der MAP-Server je nach Verfügbarkeit der *Metadaten* eine verzögerte Antwort an den MAP-Client senden könnte. Bei allen anderen Operationen antwortet der MAP-Server direkt auf eine Anfrage des MAP-Clients.

Die *publish*-Operation dient zum Veröffentlichen von *Metadaten*. Diese Informationen können auf unterschiedliche Weise publiziert werden. Ein *publishUpdate* veröffentlicht die Informationen an den MAP-Server. Die *Metadaten* werden dabei im Datenmodell des MAP-Servers gespeichert und bleiben solange gültig, bis sie explizit gelöscht werden. Beim *publishNotify* werden die Informationen nicht im Datenmodell des MAP-Servers gespeichert, sondern lediglich an die MAP-Clients weitergeleitet, die zum Zeitpunkt der Veröffentlichung ein Abonnement für diese *Metadaten* besitzen. Über die *publishDelete*-Operationen können hingegen zuvor veröffentlichte *Metadaten* im Datenmodell des MAP-Servers explizit gelöscht werden.

Mit der *subscribe*-Operation können *Metadaten*, die mit bestimmten *Identifiern* verknüpft sind, abonniert werden. Zusätzlich ist dabei der Einsatz verschiedener Filter möglich, um die gesuchten *Metadaten* je nach Anwendungsfall weiter einzuschränken. Bei der Veröffentlichung entsprechender Informationen wird der MAP-Client schließlich durch den MAP-Server asynchron benachrichtigt. Ein einzelner MAP-Client kann dabei Abonnements für verschiedene *Metadaten* besitzen. Ein Abonnement kann hierzu gewissermaßen als dauerhafte *search*-Operation angesehen werden. Durch die besagte *search*-Operation kann ein MAP-Client explizit nach *Metadaten* im Datenmodell des MAP-Servers suchen. Dazu muss er einen bestimmten *Identifier* als Startpunkt der Suche definieren. Im Gegensatz zum Abonnement wird die *search*-Anfrage einmalig ausgeführt und vom MAP-Server direkt beantwortet. Durch einen *poll* kann der MAP-Client explizit nachfragen, ob *Metadaten* für ein zuvor spezifiziertes Abonnement im MAP-Server zur Verfügung stehen. Der MAP-Server antwortet dem MAP-Client sobald die abonnierten *Metadaten* durch einen anderen MAP-Client publiziert werden. Mit der *purgePublish*-Operation kann ein MAP-Client eine Anfrage an den MAP-Server schicken, sodass alle von diesem MAP-Client zuvor publizierten *Metadaten* im Datenmodell des MAP-Servers gelöscht werden sollen. Der MAP-Server entscheidet dann, ob die Informationen gelöscht werden oder nicht. Details zu den IF-MAP Operationen sind in [33] zu finden.

2.4.6 IF-MAP im Kontext des entwickelten Clients

Wie in Abschnitt 2.4.3 beschrieben wurde, kann ein MAP-Client je nach Anwendungsfall auf unterschiedliche Weise mit dem MAP-Server kommunizieren bzw. *Metadaten* austauschen. So existieren MAP-Clients, die ausschließlich *Metadaten* publizieren und andere, die wiederum nur bestimmte Informationen vom MAP-Server abonnieren. Für einen MAP-Client auf der Android-Plattform sind dementsprechend die folgenden Anwendungsszenarien denkbar:

- Publizieren von *Metadaten* bzgl. des mobilen Geräts (z. B. Sensor-, System-, Kontext- und Anwendungsinformationen)
- Abonnement für bestimmte *Metadaten* beim MAP-Server oder explizite Suche nach bestimmten Informationen im MAP-Server

In Bezug auf die Veröffentlichung von *Metadaten* können diverse Informationen des mobilen Geräts mit Hilfe der Android-Plattform gesammelt werden. Details können dazu aus den Abschnitten 2.2 und 4.1 entnommen werden. Weiterhin wäre es denkbar, dass der Client ein Abonnement für *request-for-investigation-Metadaten* besitzt, mit welchem z. B. das Auslesen bestimmter Smartphone-Daten durch einen anderen MAP-Client angestoßen werden könnte.

Im Kontext des entwickelten IF-MAP Clients existieren u. a. einige Softwarekomponenten der *Trust@HsH*-Forschungsgruppe, in die sich die Client-Anwendung entsprechend eingliedert. Dazu stellt zunächst die Bibliothek *ifmapj* einen wichtigen Bestandteil des entworfenen Clients dar. Diese vereinfacht die Entwicklung eigener MAP-Clients in der Programmiersprache *Java*, indem sie die eigentliche Kommunikation mit dem MAP-Server über das IF-MAP Protokoll durch einige *Java*-Methoden abstrahiert. Der Entwickler eines MAP-Clients muss sich daher nicht mehr mit den Details des IF-MAP Kommunikationsprotokolls auseinandersetzen. Um *Metadaten* zu publizieren, werden diese einfach als Parameter in Form von XML-Dokumenten an die entsprechende Methode der *ifmapj*-Bibliothek übergeben. Dabei spielt es keine Rolle, ob es sich um standardisierte *Metadaten* der TCG oder um selbstdefinierte *Metadaten* handelt. Die *ifmapj*-Bibliothek wurde so entworfen, dass sie möglichst wenige Abhängigkeiten zu anderen Bibliotheken besitzt. Deshalb ist es möglich, *ifmapj* auf jeder Plattform zu verwenden, die eine *Java*-Laufzeitumgebung unterstützt (also z. B. auch auf der Android-Plattform).

Neben der *ifmapj*-Bibliothek bietet die *Trust@HsH*-Forschungsgruppe zusätzlich *irond* als MAP-Server an. Dieser ist ebenfalls in *Java* implementiert und besitzt minimale Abhängigkeiten zu anderen Bibliotheken. In der aktuellen Version basiert *irond* auf der IF-MAP Spezifikation 2.2 [33]. Außerdem existiert mit *irondetect* ein weiterer MAP-Client, der *Metadaten* anhand von unterschiedlichen Bedingungen und Richtlinien korreliert und analysiert. Auf gefundene Anomalien reagiert *irondetect* dadurch, dass wiederum spezifische *alert-Metadaten* an den MAP-Server publiziert werden. Zusätzlich kann die Softwarekomponente *VisITMeta* verwendet werden, um das im MAP-Server verwaltete Datenmodell in Form einer Graphenstruktur zu visualisieren. Für weitere Informationen zu den von der *Trust@HsH*-Forschungsgruppe veröffentlichten Softwareprodukten im IF-MAP Umfeld sei hier auf deren offizielle Webseite [34] verwiesen.

3 Bestehende Lösungen im fachlichen und technischen Umfeld

Nachdem im vorherigen Kapitel die Basistechnologien zur Entwicklung des IF-MAP Clients für Android mit Hilfe einer *Event Processing Engine* zur Datenverdichtung vorgestellt wurden, werden in diesem Kapitel die bereits bestehenden Lösungen im Kontext des entwickelten Clients analysiert und bewertet. In Abschnitt 3.1 werden dazu mehrere fachliche Lösungsansätze vorgestellt, die verschiedene Sicherheitsprobleme auf den Smartphones und Tablets erkennen sowie diese Geräte sicher in ein Unternehmensnetz integrieren können. Im Anschluss daran erfolgt in Abschnitt 3.2 die Analyse mehrerer technischer Lösungsansätze für den Einsatz von *Complex Event Processing* (CEP) im Kontext von solchen mobilen Geräten. Abschließend werden in Abschnitt 3.3 die aufgezeigten Lösungsstrategien bewertet und gegen den eigenen Client abgegrenzt.

3.1 Fachliche Lösungen zur mobilen Sicherheitsanalyse

3.1.1 Hostbasierte Sicherheitskonzepte

Neben den grundlegenden Sicherheitsmechanismen der Android-Plattform wie z. B. *Sandboxing* und *Permissions* (siehe Abschnitt 2.2.3), bieten hostbasierte Sicherheitskonzepte in Form von Betriebssystem-Erweiterungen zusätzlichen Schutz vor mobiler Malware [5]. Bei der Sicherheitsanalyse durchsuchen diese Produkte die auf den mobilen Geräten installierten *Apps* nach Schadsoftware und anderen Schwachstellen. Dafür benötigen sie allerdings entsprechende Modifikationen an der zugrundeliegenden Android-Plattform (z. B. *Root-Zugriff* o. Ä.). Der Vorteil hostbasierter Sicherheitskonzepte liegt darin, dass aufgrund dieser Betriebssystem-Modifikationen zusätzliche Schutzmaßnahmen auf allen Ebenen der Android-Plattform, wie z. B. direkt im Linux-Kernel oder in anderen Schichten der Android-Architektur (siehe Abschnitt 2.2.2), realisiert werden können. Beispielsweise können somit einzelne *Apps* besonders stark voneinander isoliert werden. Hostbasierte Sicherheitskonzepte besitzen jedoch zwei gravierende Nachteile: Zum einen fokussieren sie sich ausschließlich auf die Überwachung und den Schutz der zugrundeliegenden mobilen Geräte bzw. deren Betriebssysteme. Die Umgebung der Smartphones und Tablets, z. B. die Unternehmensinfrastruktur, wird dabei nicht beachtet. Demnach ist eine Integration dieser Produkte in die Sicherheitssysteme vorhandener IT-Infrastrukturen oftmals nicht möglich. Zum anderen stellen hostbasierte Sicherheitskonzepte schwergewichtige Schutzmechanismen dar, die für die Durchführung ihrer Analyseaufgaben teilweise tiefgreifende Modifikationen am zugrundeliegenden Betriebssystem erfordern.

3.1.2 CADS als netzwerkbasiertes Sicherheitskonzept

Netzwerkbasierte Sicherheitskonzepte, wie *CADS*[1] von Bente [5], ermöglichen die sichere Integration mobiler Geräte in bestehende Unternehmensinfrastrukturen. Dabei untersucht *CADS* drei Aspekte der Smartphones und Tablets, um deren allgemeinen Sicherheitszustand zu ermitteln: die installierten *Apps*, das Verhalten und den Verwendungskontext. Anhand der Ergebnisse dieser Sicherheitsanalyse wird schließlich der Zugang zum Unternehmensnetz für die entsprechenden mobilen Geräte gesteuert (also z. B. freigegeben oder gesperrt). Im Kontext von *CADS* werden sicherheitsrelevante Informationen verteilt auf den Smartphones und Tablets durch eine dezentrale *App* gesammelt, während der eigentliche Sicherheitszustand anschließend mit Hilfe der zentralen *Correlation-Engine irondetect* in Echtzeit analysiert wird. Im Gegensatz zu den hostbasierten Sicherheitskonzepten aus Abschnitt 3.1.1 ist *CADS* ein leichtgewichtiges System zur Integration mobiler Geräte in bestehende IT-Infrastrukturen, um deren Sicherheitszustand zu analysieren und daraus Entscheidungen für oder gegen den Zugang zum Unternehmensnetz bzw. zu unternehmenskritischen Ressourcen abzuleiten. Dazu legt *CADS* den Fokus auf ein Sicherheitskonzept aus der Sicht der zugrundeliegenden IT-Infrastrukturen. Zur Sammlung der sicherheitsrelevanten Smartphone-Daten wird eine herkömmliche *App* verwendet, die keine weiteren Modifikationen an der Android-Plattform erfordert.

Das *CADS*-System erkennt, ob es sich bei einem bestimmten Gerät um ein Smartphone oder Tablet handelt und in welchem Kontext (z. B. die Position) es sich derzeit befindet. Anhand unternehmensspezifischer Richtlinien werden die mobilen Geräte auf schädliche oder unerwünschte *Apps* untersucht. Mit Hilfe der Erkenntnisse aus dieser Analyse kann der Zugang zum Unternehmensnetz durch zuvor spezifizierte Regeln (*Policies*) gesteuert werden. Neben dieser Grundfunktion erfüllt *CADS* weitere Anforderungen. Dazu zählen einerseits das Erkennen eines abnormalen Nutzungsverhaltens und die dynamische Sicherheitsanalyse in Echtzeit. Andererseits kann *CADS* entsprechend der Anwendungsdomäne beliebig angepasst oder erweitert werden. Dazu ist z. B. die Sammlung von sicherheitsrelevanten Informationen auf den Smartphones je nach Anwendungsfall dynamisch veränderbar. Zudem ermöglicht *CADS* auch die Definition weiterer unternehmensspezifischer Richtlinien u. a. für die Sicherheitsanalyse oder zur Steuerung des Zugangs zum Unternehmensnetz. Durch diese Flexibilität bzgl. der Anwendungsdomäne kann *CADS* problemlos in bereits bestehende IT-Infrastrukturen integriert werden. Schließlich ermöglicht *CADS* auch explizit die Nutzung privater mobiler Geräte der Mitarbeiter im Sicherheitskonzept vorhandener Unternehmensinfrastrukturen, da auf den Smartphones und Tablets lediglich eine herkömmliche *App* ohne weitere Betriebssystem-Modifikationen installiert werden muss [5].

Zum Datenaustausch zwischen den Smartphones und der zentralen Analysekomponente *irondetect* verwendet *CADS* das IF-MAP Protokoll sowie den MAP-Server *irond* (siehe Abschnitt 2.4). Ein leichtgewichtiger IF-MAP Client für Android, der die sicherheitsrelevanten Daten auf den Smartphones sammelt und an *irond* zur weiteren Analyse durch *irondetect* überträgt, bildet eine weitere Hauptkomponente des *CADS*-Systems. Da

[1] Context-related Signature and Anomaly Detection for Smartphones

die TCG keine speziellen Smartphone-*Metadaten* standardisiert hat, spezifiziert *CADS* ein eigenes Datenmodell zum Austausch sicherheitsrelevanter Informationen der mobilen Geräte, das im Hinblick auf verschiedene Anwendungsdomänen beliebig erweiterbar bzw. veränderbar ist [5]. Sogenannte *Features* bilden dabei *Metadaten*, die beliebige Eigenschaften eines Smartphones oder Tablets repräsentieren können. Sie enthalten eine eindeutige ID, einen Wert, einen Typ (*qualified*, *quantitative*, *arbitrary*), Kontextparameter (z. B. die Position und die Uhrzeit) und eine Beschreibung. Zudem werden diese *Features* durch sogenannte *Category-Identifier* strukturiert. *Categories* können dabei durch *subcategory-of-Metadaten* in Hierarchien angeordnet werden. Insgesamt bildet das *Feature-Metadatenmodell* ein Teilgraph mit sämtlichen Smartphone-Daten, der an den entsprechenden *device-Identifier* des ursprünglichen und standardisierten IF-MAP Datenmodells (siehe Abschnitt 2.4.4) angehängt wird. *CADS* bietet somit ein beliebig erweiterbares Konzept zur Transformation von Smartphone-Daten in ein angemessenes IF-MAP Vokabular. Weiterhin spezifiziert das konzeptionelle Modell von *CADS* u. a. spezielle Komponenten zur Repräsentation von Kontextparametern, Signaturen, Anomalien und Richtlinien, die allerdings außerhalb des Fokus dieser Arbeit liegen.

Die *CADS*-Architektur gliedert sich in vier logische Schichten [5]: *Feature Collectors*, *Feature Provider*, *Correlation-Engine* und *Feature Consumer*. Der oben beschriebene IF-MAP Client bildet hierbei einen *Feature Collector*, *irond* stellt den *Feature Provider* dar und die *Correlation-Engine* wird durch *irondetect* abgebildet. *Feature Consumer* können wiederum *Metadaten* von *irond* abonnieren. Bezogen auf die Zielsetzung dieser Arbeit aus Abschnitt 1.2, sind vor allem *Feature Collectors* von zentraler Bedeutung. Diese sammeln sicherheitsrelevante Informationen direkt auf den Smartphones (*Feature Measurement*), transformieren sie mit Hilfe des oben erläuterten *Feature-Metadatenmodells* in IF-MAP Vokabular und übertragen die so generierten *Metadaten* schließlich an *irond* (*Feature Transmission*). Bente hat in [5] sowohl eine Abbildung diverser Smartphone-Daten auf das *Feature-Metadatenmodell* als auch eine Abbildung der logischen Schichten der *CADS*-Architektur auf physikalische Netzwerkkomponenten vorgestellt.

Im *CADS*-Prototyp kommt ein IF-MAP Client für Android der *DECOIT GmbH* als *Feature Collector* zum Einsatz [11]. Für die Kommunikation mit *irond* über das IF-MAP Protokoll verwendet er die von der *Trust@HsH*-Forschungsgruppe entwickelte Bibliothek *ifmapj* (siehe Abschnitt 2.4.6). Dieser Client enthält eine logische Schicht, um verschiedene Informationen der Android-Plattform bzw. der Smartphones und Tablets auszulesen. Dazu werden mehrere Dienste und Schnittstellen des *Application Frameworks* (siehe Abschnitt 2.2.2) z. B. zum Sammeln von System-, Telefon- und Anwendungsinformationen verwendet. Des Weiteren enthält dieser Client einige *Observer*-Komponenten, die u. a. den Ladezustand des Akkus, die aktuelle Position und die Verwendung der Kamera überwachen. Getrennt von der Informationsgewinnung findet die Transformation der gesammelten Daten in das *Feature-Metadatenmodell* statt. Das Publizieren der *Metadaten* erfolgt schließlich durch den Aufruf der entsprechenden Methode der *ifmapj*-Bibliothek. Sämtliche Konfigurationen der Client-Anwendung sowie der Verbindung zum MAP-Server werden über ein XML-Dokument verwaltet. Durch ein Setup können alle zu sammelnden Daten explizit spezifiziert werden. Der Client ist so programmiert, dass die *Datengewinnung* und -*übertragung* im Hintergrund ausgeführt werden, während

die Smartphones und Tablets für andere Zwecke verwendet werden können.

CADS bietet somit ein vollständiges Konzept zur sicheren Integration von mobilen Geräten in vorhandene Unternehmensnetze. Der Vorteil dieses Systems besteht darin, die logischen Schichten der *CADS*-Architektur physikalisch auf mehreren Netzwerkkomponenten zu verteilen. Einerseits wird dadurch auf den Smartphones und Tablets lediglich eine *App* benötigt, die ohne weitere Betriebssystem-Modifikationen Sicherheitsdaten sammelt und diese an andere Komponenten bereitstellt. Andererseits können durch den netzwerkbasierten Ansatz mobile Geräte in bereits bestehende Sicherheitskonzepte einer Unternehmensinfrastruktur eingebunden werden. Durch das IF-MAP Protokoll und *irondetect* ist *CADS* echtzeitfähig und erweiterbar. Allerdings kann sich die Integration von *CADS* in eine IT-Infrastruktur aufgrund der vier logischen Architekturschichten sehr komplex gestalten. Zudem erfordern die *Datengewinnung* im IF-MAP Client und die Sicherheitsanalyse durch *irondetect* fachliches Wissen über die Anwendungsdomäne. Ein Nachteil von *CADS* findet sich in der Konzeption des IF-MAP Clients als *Feature Collector*. Dieser sammelt sämtliche sicherheitsrelevanten Informationen und sendet diese direkt an *irond*. Je nach Anwendungsfall sind die auf den mobilen Geräten gesammelten Informationen sehr feingranular und kommen in riesiger Anzahl vor, sodass eine enorme Menge an *Feature-Metadaten* generiert und an *irond* übertragen werden muss. Einerseits führt dies zu einer erhöhten Belastung der Smartphones und Tablets selbst, wodurch u. a. deren Energieverbrauch ansteigen kann. Andererseits können aber auch *irond* sowie die gesamte Unternehmensinfrastruktur durch die von der großen Menge an übertragenen *Feature-Metadaten* verursachte Netzwerklast stark beansprucht werden. Des Weiteren sammelt der IF-MAP Client auch personenbezogene Daten wie z. B. die aktuelle Position und überträgt diese an *irond*, ohne sie zuvor anonymisiert zu haben.

3.1.3 Analyse des Smartphone-Nutzungsverhaltens

Ruhe hat in [23] mit dem *Android Usage Study System* ein Konzept entwickelt, das die Nutzung von Smartphones und Tablets über längere Zeiträume analysiert, um daraus Grundlagen für eine Anomalienerkennung auf diesen Geräten abzuleiten. Die zentrale Komponente dieses Systems bildet eine Android-*App*, die diverse Sensor-, System-, Kontext- und Anwendungsinformationen der zugrundeliegenden Geräte sammelt und in einer lokalen Datenbank zwischenspeichert. In regelmäßigen Abständen werden die so gesammelten Daten schließlich an einen zentralen Server zur weiteren Analyse übermittelt.

Das System umfasst somit vier Hauptaufgaben: *Datensammlung*, *Datenspeicherung*, *Datenübertragung* und *Datenanalyse*. Die *Datensammlung* findet direkt auf den mobilen Geräten statt, wobei eine Vielzahl verschiedener Informationen mit Hilfe der Android-Plattform einmalig, periodisch oder ereignisbasiert durch die *App* im Hintergrund gesammelt wird. Alle gesammelten Informationen werden zunächst auf den mobilen Geräten in ein spezielles Datenmodell übersetzt und anschließend in einer lokalen Datenbank zwischengespeichert. In regelmäßigen Abständen erfolgt schließlich die automatische *Datenübertragung* der gespeicherten Informationen an einen zentralen Server zur weiteren *Datenanalyse*. Bei der *Datenübertragung* zwischen der *App* und dem Server wird das HTTP Protokoll in Kombination mit SSL verwendet. Weiterhin beachtet das System

Datenschutzbestimmungen für personenbezogene Daten, sodass z. B. die Position der Smartphones oder Tablets bereits bei der *Datensammlung* anonymisiert wird. Zudem können über die *App* die zu sammelnden Daten individuell ausgewählt werden.

Die Architektur der *App* gliedert sich in drei logische Schichten: *Datensammlung*, *Datentransformation* und *Datenverarbeitung*. Abhängig von der verwendeten Android-Version sammelt die *App* Daten aus den folgenden Kategorien [23]: Allgemeine Geräte- und Systeminformationen, Netzwerk-, Telefon- und SMS-Statistiken, Anwendungs- und Sensorinformationen, Speicher- und Prozessorauslastungen und Statistiken über den Energieverbrauch der Smartphones. Zur Speicherung dieser Daten wird das *Feature-Metadatenmodell* aus Abschnitt 3.1.2 verwendet. Dabei stellen *Features* eine beliebige Eigenschaft der Smartphones dar. Durch *Categories* können diese *Features* wiederum organisiert und in semantische Beziehungen zueinander gesetzt werden. Weiterhin werden allgemeine Kontextparameter (z. B. die Position) zu den *Features* hinzugefügt. Dieses Datenmodell ist unabhängig von einer konkreten Anwendungsdomäne und beliebig erweiterbar bzw. anpassbar. Für die Speicherung des Datenmodells wird ein objektorientiertes DBMS verwendet. Eine detailierte Abbildung diverser domänenspezifischer Smartphone-Daten auf das oben beschriebene *Feature-Metadatenmodell* sowie eine exemplarische Analyse dieser Informationen werden von Ruhe in [23] vorgestellt.

Der Vorteil des *Android Usage Study Systems* liegt darin, dass nahezu sämtliche verfügbaren Informationen der Smartphones gesammelt und ausgewertet werden. Besonders aufgrund der Verwendung des *Feature-Metadatenmodells* ist das System auch auf andere Anwendungsdomänen anpassbar. Weiterhin werden bestimmte Datenschutzaspekte im Umgang mit personenbezogenen Daten beachtet, sodass kritische Informationen bereits vor ihrer Speicherung und Übertragung anonymisiert werden. Das System besitzt jedoch den Nachteil, dass die *Datenanalyse* nicht in Echtzeit stattfindet, da die Informationen zunächst auf den Smartphones und Tablets zwischengespeichert werden. Zudem wird die Integration des Systems in vorhandene Sicherheitskonzepte erschwert, da es kein standardisiertes Protokoll wie IF-MAP verwendet, das zum Austausch der Smartphone-Daten über einen zentralen MAP-Server dienen könnte.

3.1.4 Weitere Arbeiten im Kontext der Smartphone-Sicherheit

Dieser Abschnitt bietet einen Überblick über weitere Arbeiten zu host- und netzwerkbasierten Sicherheitskonzepten für Android-Smartphones und -Tablets. Dies verdeutlicht den Umfang der Forschungen zu diesem Thema in den letzten Jahren. Eine detailierte Analyse weiterer bestehender Sicherheitskonzepte für mobile Geräte ist u. a. in [5] zu finden.

Das hostbasierte Sicherheitskonzept *Kirin* von Enck et al. [15] fokussiert sich auf den Schutz vor potentiell gefährlichen Drittanbieter-*Apps* unter Android. Dazu wird während der Installation einer *App* deren Sicherheitskonfiguration (z. B. verwendete *Permissions*) anhand vordefinierter Regeln untersucht. Über diese Regeln können entsprechend unerwünschte Sicherheitskonfigurationen für Drittanbieter-*Apps* flexibel festgelegt werden. Trifft bei der Installation einer *App* eine dieser Regeln zu, so wird diese Installation verhindert und der Benutzer entsprechend informiert. Die Regeln werden dazu in

der speziellen Regelsprache *Kirin Security Language* geschrieben. Diese Lösung zeigt, dass die Prüfung der Sicherheitskonfigurationen der installierten *Apps* einen zentralen Bestandteil eines wirkungsvollen Sicherheitskonzepts für die Android-Plattform bildet.

Enck et al. präsentieren in [14] ein weiteres hostbasiertes Sicherheitskonzept namens *TaintDroid*. Dieses System dient der Erkennung potentiell böswilliger Drittanbieter-*Apps* anhand der Überwachung von Informationsflüssen unter Android. Dabei wird die Android-Plattform so modifiziert, dass *TaintDroid* den Informationsfluss diverser personenbezogener Daten (z. B. die Geräteposition) durch beliebige Drittanbieter-*Apps* nachverfolgen kann. Schließlich informiert *TaintDroid* den Benutzer, sobald personenbezogene Daten über eine nicht-vertrauenswürdige Drittanbieter-*App* das zugrundeliegende mobile Gerät verlassen. Die Ergebnisse während der Testphase von *TaintDroid* zeigten dabei, dass diese unerwünschten Informationsflüsse personenbezogener Daten durch solche Drittanbieter-*Apps* sehr oft stattfinden und dementsprechend eine ernstzunehmende Sicherheitsbedrohung für mobile Geräte mit dem Android-Betriebssystem darstellen.

Die Problematik, dass viele Drittanbieter-*Apps* auf der Android-Plattform auf unerwünschte Weise personenbezogene Daten mit entfernten Servern austauschen, liegt auch im Fokus von *AppFence*, das von Hornyack et al. in [21] vorgestellt wird. *AppFence* erweitert dabei die Android-Plattform um zwei grundlegende Funktionalitäten: Einerseits kann der Benutzer bei der Anfrage einer entsprechend unseriösen Drittanbieter-*App* für den Zugriff auf bestimmte personenbezogene Daten vorgetäuschte bzw. künstliche Informationen zur Verfügung stellen. Andererseits unterstützt *AppFence* auch die Bereitstellung echter personenbezogener Daten an die Drittanbieter-*Apps*, wobei allerdings die sicherheitskritischen Daten vor dem Austausch über beliebige Kommunikationsschnittstellen durch *AppFence* geschützt werden. Damit verhindert *AppFence*, dass personenbezogene Daten über unerwünschte Informationskanäle das zugrundeliegende mobile Gerät verlassen. Unter Verwendung von *AppFence* sind keine weiteren Modifikationen an den zu überwachenden Drittanbieter-*Apps* nötig.

Cheng et al. präsentieren in [10] das netzwerkbasierte Sicherheitskonzept *SmartSiren* zur Erkennung von Viren auf Android-Smartphones und -Tablets, die sich über die Netzwerkschnittstellen der Geräte verbreiten wollen. Dazu umfasst *SmartSiren* eine *App*, die die Aktivitäten der Netzwerkschnittstellen (z. B. WIFI, Bluetooth, Telefon- und Datennetz) der mobilen Geräte überwacht und entsprechende Statusmeldungen generiert. Diese werden schließlich an einen zentralen Server übertragen, der mit Hilfe der jeweiligen Statusmeldungen vieler Smartphones und Tablets die Virensuche anhand verschiedener Berechnungen und Methoden durchführt. Diese bereits im Jahr 2008 vorgestellte Lösung zeigt, welche Möglichkeiten netzwerkbasierte Sicherheitskonzepte bieten, bei denen der Kontext zwischen mehreren mobilen Geräten betrachtet wird.

In [9] stellen Burguera et al. ein System namens *Crowdroid* für die dynamische Erkennung von Schadsoftware anhand einer Verhaltensanalyse auf Android-Smartphones und -Tablets vor. Im Speziellen versucht *Crowdroid* mobile Trojaner zu finden, die andere gutartige Drittanbieter-*Apps* infizieren, um sich selber weiter zu verbreiten. Dementsprechend fokussiert sich *Crowdroid* auf die Suche nach Drittanbieter-*Apps*, die zwar den gleichen Namen und die gleiche Version aufweisen, sich jedoch signifikant in ihrem

Verhalten aufgrund eines enthaltenen Trojaners unterscheiden. Dazu umfasst *Crowdroid* eine Client-Anwendung, die direkt auf den mobilen Geräten die Systemaufrufe diverser Drittanbieter-*Apps* überwacht und die gesammelten Informationen schließlich an einen zentralen Server übermittelt. Durch die empfangenen Daten baut der Server ein spezifisches Datenmodell auf. Dieses Datenmodell wird mit Hilfe verschiedener Algorithmen analysiert, um Drittanbieter-*Apps*, die keine Schadsoftware enthalten, von solchen Anwendungen zu unterscheiden, die durch einen oder mehrere Trojaner infiziert sind. Dieses System weist hohe Erkennungsraten zwischen 85% und 100% auf. *Crowdroid* besitzt allerdings auch zwei Nachteile: Zum einen basiert die Erkennung von infizierten Drittanbieter-*Apps* ausschließlich auf der Grundlage einer Messung der Systemaufrufe der jeweiligen mobilen Anwendungen. Zum anderen kann *Crowdroid* nur dann eine durch einen Trojaner infizierte Drittanbieter-*App* erkennen, wenn zu der entsprechenden *App* auch eine passende gutartige Version der mobilen Anwendung im Analyseprozess von *Crowdroid* untersucht wird.

3.2 Technische Lösungen zu mobilem CEP

3.2.1 CEP im Kontext von Ambient Assisted Living auf mobilen Geräten

Stipkovic et al. haben in [28] ein ereignisgesteuertes AAL-System[2] vorgestellt, das die Sensordaten von Smartphones sammelt und diese direkt auf den mobilen Geräten durch eine mobile CEP-Komponente verarbeitet. Dementsprechend werden Smartphones und CEP zu einem Konzept namens *Mobile Event Processing* kombiniert. CEP ist die zentrale Komponente zur *Ereignisverarbeitung* einer EDA, wobei z. B. alle Sensoren eines Smartphones die entsprechenden Ereignisse erzeugen. Somit wird auf den mobilen Geräten ein kontinuierlicher Ereignisstrom in Echtzeit verarbeitet. Dabei werden die Sensorereignisse *gefiltert*, *angereichert*, *aggregiert*, *korreliert* und auf signifikante *Muster* untersucht, um komplexe Ereignisse mit einem hohen Informationswert zu erzeugen. Komplexe Ereignisse repräsentieren schließlich Erkenntnisse über kritische Situationen. So kann beispielsweise die Situation erkannt werden, dass eine Person gestürzt ist und Hilfe benötigt. Die jeweilige *Ereignisbehandlung* erfolgt dann z. B. durch die Ausführung eines Notrufs [28]. Das AAL-System umfasst somit alle logischen Schichten einer EDA (siehe Abschnitt 2.3.1) direkt auf den Smartphones. Die *Ereignisverarbeitung* durch die mobile CEP-Komponente *Asper* (siehe Abschnitt 2.3.6) wird erst durch hochentwickelte Betriebssysteme wie Android und mobile Geräte mit ausreichenden Hardwarekapazitäten ermöglicht. Android umfasst verschiedene *Ereignisquellen* für *Asper* wie z. B. das *Sensor Framework* oder Betriebssystem-Broadcasts. Die *Ereignisverarbeitung* des AAL-Systems gliedert sich dabei in mehrere logische EPAs (siehe Abschnitt 2.3.2).

Im Kontext des AAL-Systems werden verschiedene Architekturmodelle zur Umsetzung von CEP in Bezug auf mobile Geräte vorgestellt. Bei einer serverbasierten CEP-

[2] Ambient Assisted Living System zur Unterstützung älterer oder beeinträchtigter Menschen

Architektur bilden die mobilen Geräte lediglich die *Ereignisquellen*. Die *Ereignisverarbeitung* sowie die *Ereignisbehandlung* finden hingegen auf einem leistungsstarken, zentralen Server statt. Die Smartphones und Tablets übertragen dabei sämtliche ungefilterten und feingranularen Ereignisse über das zugrundeliegende Netzwerk an den Server, wo diese anschließend verarbeitet werden. Der Vorteil liegt darin, dass die gesamte komplexe Verarbeitung großer Ereignismengen auf einem Server ausgelagert wird und so die begrenzten Ressourcen der mobilen Geräte geschont werden. Weiterhin wird die zentrale Verarbeitung von Ereignissen verschiedener Smartphones und Tablets ermöglicht. Der Nachteil dieses Ansatzes liegt allerdings in der Abhängigkeit zum Netzwerk sowie zum Server. Ohne Netzwerkverbindung findet z. B. keine *Ereignisverarbeitung* statt. Die Übertragung großer Ereignismengen kann zudem eine starke Netzwerklast verursachen. Beim Austausch personenbezogener Daten können zusätzlich Datenschutzkonflikte entstehen [28].

Das vorgestellte AAL-System nutzt eine mobile CEP-Komponente. Dabei beinhalten die Smartphones neben den *Ereignisquellen* auch die *Ereignisverarbeitung* und die *Ereignisbehandlung*. Nachdem ein Ereignis erzeugt wurde, wird es direkt auf den mobilen Geräten verarbeitet. Dieses System besitzt somit den Vorteil, dass es unabhängig von einem Server oder dem Netzwerk ist. Zudem entsteht dabei keine Netzwerklast durch den Ereignisaustausch und personenbezogene Daten werden direkt auf den mobilen Geräten verarbeitet. Der Nachteil liegt jedoch darin, dass die mobile CEP-Komponente nur Ereignisse des zugrundeliegenden Geräts verarbeiten kann. Eine *Ereignisverarbeitung* über mehrere mobile Geräte ist demnach nicht möglich. Abbildung 3.1 stellt eine serverbasierte und eine mobile Architekturvariante für CEP-Komponenten gegenüber.

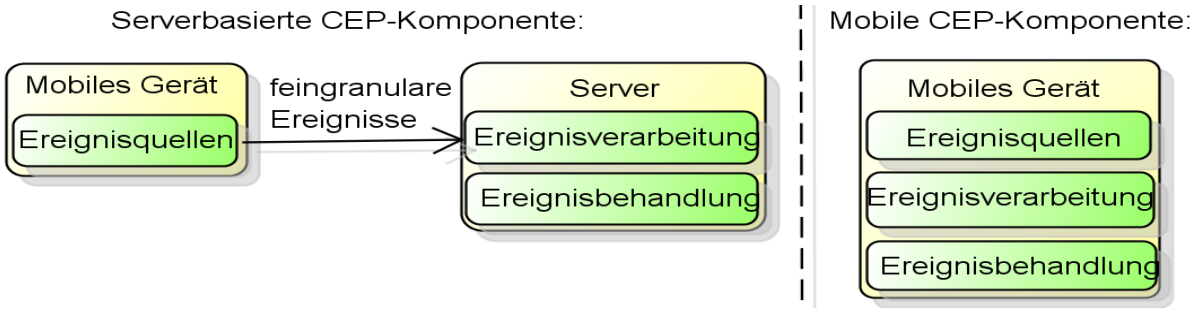

Abbildung 3.1: Serverbasierte und mobile CEP-Komponenten

Schließlich existieren auch noch hybride Varianten dieser beiden Architekturmodelle [28]. Dabei enthalten die Smartphones eine mobile CEP-Komponente, die die Ereignisse bereits auf den mobilen Geräten vorverarbeitet. Allerdings findet der Hauptteil der *Ereignisverarbeitung* schließlich auf einem zentralen Server statt. Dadurch wird einerseits die Menge der übertragenen Ereignisse reduziert und andererseits die Verarbeitung von Ereignissen verschiedener Smartphones und Tablets ermöglicht.

3.2.2 CEP zur mobilen Koordination von Rettungswagen

In [7] stellen Bruns et al. ein ereignisgesteuertes System zur automatischen Koordination von Rettungswagen vor. Dabei wird CEP zur Verdichtung bestimmter Rettungswagen-Informationen verwendet, um den Einsatzzustand eines Fahrzeugs in Echtzeit zu ermitteln und an eine zentrale Koordinationsstelle zu senden. Dieses Konzept bildet eine Erweiterung vorhandener Koordinationssysteme, indem es ein *Zustandsmodell* in Form eines endlichen Automaten für jeden Rettungswagen einführt, das in Echtzeit durch CEP automatisch aktualisiert wird. Dabei können folgende Zustände angenommen werden: *idle*, *assigned*, *assisting*, *transfer*, *out of service* und *work-break*. Neben dem Einsatzstatus sind natürlich auch die Distanz zum Patienten, die Dringlichkeit und der Typ des Rettungswagens von entscheidender Bedeutung für die Koordination der Fahrzeuge. Ähnlich zum AAL-System aus Abschnitt 3.2.1 wird eine mobile CEP-Komponente verwendet, um Sensorereignisse zu verarbeiten (*Filterung*, *Anreicherung*, *Aggregation*, *Korrelation* und *Mustererkennung*) und daraus komplexe Ereignisse zu generieren, die eine Änderung des Einsatzzustands eines Rettungswagens repräsentieren. Durch die Behandlung dieser komplexen Ereignisse wird der Status im *Zustandsmodell* aktualisiert.

Das System teilt sich dabei in zwei physikalische Schichten auf: eine zentrale Koordinationsstelle zur Organisation des gesamten Rettungssystems und mobile *Apps* in den Fahrzeugen zur Analyse des jeweiligen Einsatzzustands durch *Mobile Event Processing*. Dazu werden verschiedene Sensordaten der mobilen Geräte in den Rettungswagen (z. B. die Position) oder externe Informationen (z. B. die Verkehrslage) ausgewertet. Die mobile CEP-Komponente ist dabei in mehrere logische EPAs (siehe Abschnitt 2.3.2) unterteilt: *Ereignisfilterung*, *Bewegungserkennung* und *Situationserkennung*. Die *App* wurde für die Android-Plattform entwickelt und verwendet *Asper* (siehe Abschnitt 2.3.6) als mobile CEP-Komponente. Die Vorteile dieses Systems liegen in der zeitnahen Zustandserkennung und der Verwendung eines *Zustandsmodells* für den jeweiligen Einsatzstatus.

3.2.3 Weitere Arbeiten zum Einsatz von CEP auf mobilen Geräten

Dieser Abschnitt bietet einen Überblick über weitere Arbeiten, bei denen CEP im Kontext von mobilen Geräten zum Einsatz kommt. Dies verdeutlicht den Umfang der Forschungen zu diesem Thema in den letzten Jahren.

In [36] stellen Westhuis et al. ein System zur Lokalisierung von Büchern in Bibliotheken anhand von RFID-Transpondern vor. Diese werden mit Smartphones oder Tablets verbundenen Bluetooth-Lesegeräten erfasst, wobei die ausgelesenen Transponder-Daten über die mobilen Geräte an einen zentralen Server übertragen werden, der anschließend mit Hilfe einer CEP-Komponente eine *Ereignisverarbeitung* durchführt. Dadurch können sowohl die Positionen der suchenden Personen als auch die der gesuchten Bücher bestimmt werden. Als Ergebnis der *Ereignisverarbeitung* werden bestimmte Navigationsbefehle an die mobilen Geräte zurückgesendet und dort visualisiert. Diese Lösung lässt sich dabei auch auf andere Anwendungsdomänen z. B. in Lagersystemen anpassen.

Belenger analysiert in [4] den Einsatz von *Complex Event Processing* in einem verteilten System aus mobilen Geräten als *Ereignisquellen* und einem zentralen Server zur

Ereignisverarbeitung mittels CEP. Im Kontext dieses Systems wird dabei ausführlich die Abhängigkeit der *Ereignisverarbeitung* von dem zugrundeliegenden Netzwerk thematisiert, die dazu führt, dass Netzwerkprobleme wie z. B. Verbindungsabbrüche die gesamte Verarbeitung der CEP-Komponente auf dem zentralen Server verhindern können.

Amadei stellt in [1] ein System vor, mit dessen Hilfe Positionsdaten der Smartphones von Außendienstmitarbeitern auf einem zentralen Server mittels CEP ausgewertet werden können. Ergebnisse werden als SMS an die Smartphones zurückgesendet, sodass ein Mitarbeiter u. a. informiert wird, wenn er sich in der Nähe eines Kunden befindet.

In [2] entwickelt Arango ein System zur Qualitätssicherung, das auf mobilen Geräten Informationen über den Aufenthaltsort und die Qualität der Datennetzverbindung (z. B. Latenzzeiten) sammelt und mit Hilfe einer serverbasierten CEP-Komponente analysiert. Somit kann z. B. ein Netzbetreiber zeitnah Netzwerkprobleme erkennen und lokalisieren.

Xu et al. präsentieren in [38] ein System, das verschiedene Sensordaten von mobilen Geräten im Kontext der Gesundheitsfürsorge auswertet und überwacht. Die Kombination aus einer mobilen und einer serverseitigen CEP-Komponente ermöglicht dabei eine flexible und effiziente Verarbeitung der auf den Smartphones und Tablets gesammelten Daten. Dieser hybride Ansatz schont die begrenzten Ressourcen der mobilen Geräte und sorgt gleichzeitig für eine Reduzierung der an den Server übertragenen Ereignisdaten.

In [29] stellen Stojanovic et al. ein mobiles System vor, das CEP zur Erkennung komplexer Situationen im Gesundheits- und Fitnessbereich verwendet. Die Überwachung bestimmter Körperfunktionen (z. B. die Herzschlagfrequenz) erfolgt dabei über externe Sensoren, die über Bluetooth mit den Smartphones und Tablets verbunden sind. Die gesammelten Daten werden an einen zentralen Server gesendet, der mit Hilfe einer CEP-Komponente komplexe Situationen und Erkenntnisse mit einem hohen fachlichen Wissensgehalt aus den übertragenen Einzelereignissen ableitet. Dadurch können die Körperdaten in nahezu Echtzeit analysiert werden und die überwachten Personen erhalten eine sofortige Rückmeldung. Zudem ermöglicht der zentrale Server die *Korrelation* von Ereignissen mehrerer Personen bzw. ihrer jeweiligen mobilen Geräte.

Stipkovic analysiert in [27] die allgemeinen Möglichkeiten zur Verwendung einer mobilen *Event Processing Engine* auf Smartphones und Tablets mit Android-Betriebssystem. Dabei werden verschiedene Integrationsstrategien für die CEP-Komponente diskutiert und der grundsätzliche Einsatz von CEP auf mobilen Geräten in Bezug auf verschiedene Anwendungsszenarien kritisch bewertet.

3.3 Abgrenzung zum entwickelten IF-MAP Client

Die vorgestellten, fachlichen Konzepte bieten keine befriedigende Lösung für die Zielsetzung dieser Arbeit (siehe Abschnitt 1.2). Hostbasierte Sicherheitskonzepte erfordern zum Teil Betriebssystem-Modifikationen und fokussieren sich ausschließlich auf die Überwachung und den Schutz der mobilen Geräte selbst, während das Umfeld keine Beachtung findet (siehe Abschnitt 3.1.1). Demnach sind solche Systeme oftmals nicht in bestehende Sicherheitskonzepte von Unternehmensinfrastrukturen integrierbar, wodurch sie als Lösungen für die Ziele dieser Arbeit ungeeignet sind. *CADS* bietet ein

echtzeitfähiges und flexibles Konzept zur sicheren Integration mobiler Geräte in vorhandene IT-Infrastrukturen (siehe Abschnitt 3.1.2). Allerdings werden zwischen den *Feature Collectors* und dem *Feature Provider* sämtliche feingranularen Daten ausgetauscht, wodurch die mobilen Geräte sowie das zugrundeliegende Netzwerk durch eine große Menge übertragener Daten stark beansprucht werden und Datenschutzkonflikte bzgl. personenbezogener Daten entstehen könnten. Insgesamt bietet *CADS* als netzwerkbasiertes System zur Integration von Smartphones in bestehende Infrastrukturen mit Hilfe des IF-MAP Protokolls jedoch eine solide Basis für das Konzept des eigenen Clients. Die oftmals große Menge der übertragenen Daten sowie ggf. auftretende Datenschutzkonflikte erfordern allerdings den Entwurf eines erweiterten Konzepts, das diese Nachteile unter Verwendung einer direkt auf den mobilen Geräten befindlichen CEP-Komponente zur Datenverdichtung sowie -anonymisierung eliminiert. Zusätzlich beschränkt sich das Konzept des eigenen Clients nicht nur auf die Funktionalität eines *Feature Collectors* im Kontext von *CADS*, sondern bietet darüber hinaus flexibel anpass-, erweiter- und austauschbare Komponenten zur Sammlung, Verarbeitung und Übertragung beliebiger Smartphone-Daten in verschiedenen Anwendungsdomänen. Das *Android Usage Study System* sammelt und analysiert eine Vielzahl von Smartphone-Daten unter Beachtung von Datenschutzaspekten (siehe Abschnitt 3.1.3). Das System ist zwar nicht echtzeitfähig und bietet keine Schnittstellen zur Integration in das IF-MAP Umfeld, allerdings bildet es eine Grundlage für den eigenen Client, da es eine nahezu vollständige Sammlung der auf einem aktuellen Smartphone verfügbaren Daten unterstützt. Die Analyse weiterer Arbeiten in Bezug zur Smartphone-Sicherheit verdeutlicht, dass die meisten Sicherheitskonzepte für mobile Geräte einem hostbasierten Ansatz folgen. Ein weitaus geringerer Anteil der Forschungsarbeiten richtet sich auf netzwerkbasierte Ansätze (siehe Abschnitt 3.1.4).

Technische Ansätze zur Verwendung von CEP im Kontext von mobilen Geräten zeigen ein breites Spektrum von Anwendungsdomänen, bei denen Smartphone-Daten durch eine CEP-Komponente verarbeitet werden. Dazu zählen u. a. der Gesundheits-, Pflege- und Fitnessbereich, aber auch die Auswertung von Sensor-, System-, Kontext- und Anwendungsinformationen bei der Qualitätssicherung oder der Sicherheitsanalyse. Dabei wird deutlich, dass moderne Smartphones aufgrund der Vielzahl an Daten, die auf ihnen gesammelt werden kann, eine hervorragende Basis für verschiedene Anwendungen bieten, bei denen eine CEP-Komponente zur *Datenverarbeitung* zum Einsatz kommt. Die vorgestellten Arbeiten zeigen mehrere Integrationsmöglichkeiten für eine CEP-Komponente im Kontext von mobilen Geräten. Ein Großteil der Forschungsarbeiten legt den Fokus auf serverbasierte CEP-Komponenten (siehe Abschnitt 3.2.3). Einerseits werden dadurch die begrenzten Ressourcen der mobilen Geräte nicht durch die komplexe *Ereignisverarbeitung* einer mobilen CEP-Komponente belastet und andererseits wird die *Korrelation* von Ereignissen mehrerer Smartphones ermöglicht. Serverbasierte CEP-Komponenten besitzen jedoch den Nachteil, dass die *Ereignisverarbeitung* abhängig vom zugrundeliegenden Netzwerk und dem zentralen Server stattfindet, wobei die Übertragung einer großen Menge feingranularer Ereignisse eine starke Beanspruchung des Netzwerks sowie der Netzwerkschnittstellen der Smartphones bedingen kann. Weiterhin werden auch personenbezogene Daten von den Smartphones an

den Server zur Verarbeitung übertragen, sodass es zu Datenschutzkonflikten kommen kann. Moderne mobile Geräte ermöglichen mit ihrer leistungsstarken Hardware und ihren hochentwickelten Betriebssystemen (siehe Abschnitt 2.1) inzwischen den Einsatz von mobilen CEP-Komponenten. Das vorgestellte AAL-System (siehe Abschnitt 3.2.1) sowie das System zur Koordination von Rettungswagen (siehe Abschnitt 3.2.2) sind Beispiele für den Einsatz einer solchen mobilen CEP-Komponente. Dabei beinhalten die mobilen Geräte neben den *Ereignisquellen* auch die *Ereignisverarbeitung* und *Ereignisbehandlung*, wodurch die gesamte Verarbeitung von Ereignissen unabhängig von dem zugrundeliegenden Netzwerk oder einem zentralen Server ist und somit keine Netzwerklast durch übertragene, feingranulare Ereignisdaten verursacht werden kann. Zudem werden personenbezogene Daten direkt auf den Smartphones und Tablets verarbeitet und anonymisiert, sodass keine Datenschutzkonflikte entstehen können.

Je nach Anwendungsdomäne kann es jedoch von Nachteil sein, dass die mobile CEP-Komponente nur Ereignisse des zugrundeliegenden mobilen Geräts verarbeitet. Demnach sind auch hybride Architekturvarianten denkbar, bei denen Ereignisse durch eine mobile CEP-Komponente auf den Smartphones und Tablets vorverarbeitet und anschließend durch eine serverseitige CEP-Komponente mit Ereignissen anderer Geräte *korreliert* werden können (siehe Abschnitt 3.2.1). In der Konsequenz erfordert die in der Zielsetzung dieser Arbeit beschriebene Notwendigkeit zur Reduzierung und Anonymisierung der übertragenen Daten im Kontext des *CADS*-Systems ein Konzept für den eigenen Client, bei dem eine entsprechende *Ereignisverarbeitung* direkt auf den Smartphones und Tablets durch eine rein mobile CEP-Komponente durchgeführt wird (siehe Abschnitt 2.3.6).

Bislang existiert keine Lösung, die ein netzwerkbasiertes Sicherheitskonzept im IF-MAP Umfeld für Android-Smartphones und -Tablets mit einer rein mobilen CEP-Komponente zur Datenverdichtung und -anonymisierung verbindet. Das entworfene Konzept des eigenen Clients bildet demnach eine neue Kombination aus einem netzwerkbasierten Sicherheitssystem im IF-MAP Umfeld und einer mobilen CEP-Komponente im Kontext von Android-Smartphones und -Tablets. Der Client basiert fachlich auf Konzepten des *CADS*-Systems und gliedert sich entsprechend als ein um CEP erweiterter *Feature Collector* in dessen Architektur ein. Allerdings findet dabei im Vergleich zu herkömmlichen *Feature Collectors* sowohl eine Datenverdichtung als auch eine -anonymisierung direkt auf den mobilen Geräten statt, bevor der Datenaustausch mit dem MAP-Server über das IF-MAP Protokoll erfolgt. Die technische Grundlage für das Konzept des eigenen Clients bilden Arbeiten wie das AAL-System oder das System zur Koordination von Rettungswagen, bei denen eine mobile CEP-Komponente zum Einsatz kommt. Darüber hinaus bietet das Konzept des eigenen Clients diverse Mechanismen zur flexiblen Anpassung, Erweiterbarkeit und Austauschbarkeit einzelner Komponenten zur *Datensammlung*, *-verarbeitung* und *-übertragung*, sodass beliebige Anwendungsdomänen im Kontext von Smartphone-Daten unterstützt werden. Erst die Kombination aus einer mobilen CEP-Komponente und einem netzwerkbasierten Sicherheitskonzept ermöglicht es, dass der eigene Client einerseits echtzeitfähig, flexibel anpassbar und in vorhandene Sicherheitssysteme von Unternehmensinfrastrukturen integrierbar ist und andererseits eine große und kontinuierlich eintreffende Datenmenge effektiv verdichtet und anonymisiert.

4 Anforderungsanalyse zum entwickelten IF-MAP Client

Aufbauend auf die Analyse bestehender fachlicher und technischer Lösungen im Kontext des entwickelten IF-MAP Clients werden in diesem Kapitel die Anforderungen an den eigenen Client, angelehnt an die vorgestellten Basistechnologien, erarbeitet. Dazu erfolgt in Abschnitt 4.1 zunächst eine allgemeine Auflistung und Einordnung der wichtigsten Sensor-, System-, Kontext- und Anwendungsinformationen, die auf einem heutigen Smartphone oder Tablet mit dem Android-Betriebssystem gesammelt werden können. Anhand dieser Auflistung wird in Abschnitt 4.2 ein konkretes Anwendungsszenario für den eigenen Client konstruiert, bei dem sicherheitsrelevante Informationen der mobilen Geräte spezifiziert, gesammelt, verarbeitet und an einen MAP-Server übertragen werden. Aufbauend auf dieses Anwendungsszenario erfolgt in Abschnitt 4.3 die Ausarbeitung der funktionalen und nichtfunktionalen Anforderungen an den eigenen Client. Abschließend werden in Abschnitt 4.4 Datenschutzaspekte bzgl. der Sammlung, Verarbeitung und Übertragung personenbezogener Daten im Kontext des eigenen Clients untersucht.

4.1 Sammlung von Smartphone-Daten

Moderne Smartphones und Tablets sind sehr vielseitig in Bezug auf die eingesetzte Hard- und Software (siehe Abschnitt 2.1). Nachfolgend wird daher eine Auflistung, Kategorisierung und Einordnung der wichtigsten Informationen vorgestellt, die auf heutigen mobilen Geräten mit der Android-Plattform gesammelt werden können:

- Sensorinformationen: Messwerte von Beschleunigungs-, Gyroskop-, Magnetfeld-, Licht-, Orientierungs-, Luftdruck-, Temperatur-, Näherungs- und Positionssensoren (GPS); Informationen über aktive Sensoren (u. a. für Kamera oder Mikrofon); Benachrichtigungen z. B. bei der Aufnahme neuer Bilder oder Videos
- *Statische* Systeminformationen: Geräteinformationen (Modell- und Produktbezeichnung, Branding, Hersteller, CPU- und Mainboard-Informationen); Bildschirminformationen (Auflösung, Höhe, Breite, Pixeldichte); Systemdaten (IMEI, IMSI, Telefonnummer, Bootloader-, Kernel-, Firmware-, Betriebssystem-, SDK- und Baseband-Version, Build-Nummer, Release-Codename, API-Level)
- *Dynamische* Systeminformationen: Ressourcennutzung (CPU-, RAM- und Festspeicherauslastung); Kapazitäten des Arbeits- und Festspeichers; Akkustatus (Art der Energieversorgung, Energieverbrauch, Ladezustand, Temperatur, Spannung);

Status externer Speichermedien; Anzahl und Eigenschaften von Medien-Dateien (u. a. Musik, Videos, Bilder); Informationen zur System- und Betriebszeit (z. B. letzter Systemstart bzw. letztes Herunterfahren); Informationen zu Benutzereinstellungen (Android Debug Bridge, Telefon- und Daten-Roaming, Bildschirmhelligkeit und -ausrichtung, Lautstärke (Anrufe, Nachrichten, Medien), Klingelmodus, Vibrationsmodus, Schlafmodus, Flugmodus, unbekannte Installationsquellen, Massenspeicher-Funktionalität, *Root-Zugriff*); Status der USB-Verbindung; Informationen zu An- und Ausschaltvorgängen des Bildschirms bzw. zum Aufwachen des Betriebssystems; Anzahl und Informationen zu ein- und ausgehenden Anrufen/Nachrichten; durchschnittliche Gesprächsdauer; Informationen zu Mobilfunkzellen (Position, Netzwerk-ID, System-ID, Basestation-ID, Sendeleistung); SIM-Status; Telefon-Typ; Service-Status; Anrufstatus; aktive Netzwerkschnittstellen (WIFI, 3G/4G, Bluetooth, NFC); ein- und ausgehender Datenverkehr; MAC-Adresse; IP-Adressen; Tethering-Status; Informationen zu Access-Points (SSIDs, Geschwindigkeiten, Sicherheitsmechanismen); WIFI-Status, WLAN-Informationen (Frequenz bzw. Kanal, Signalpegel); Bluetooth-Informationen (Verbindungsstatus, Scanmodus, Name/Adresse verbundener Geräte)

- Anwendungsinformationen: Installierte Anwendungen (Name, Package-Name, Installationsquelle, Version-Name, Version-Code, Minimum-SDK, Target-SDK, Installationszeitpunkt, letzte Aktualisierung, verwendete *Permissions*, erforderliche Speicherkapazitäten, Speicherort); Benachrichtigungen bei der Installation, Aktualisierung oder Deinstallation einer Anwendung; laufende Prozesse (Prozess-ID, Prozess-Benutzer, Name, Akku-, CPU- und RAM-Nutzung, Priorität)

Welche dieser Informationen auf einem konkreten mobilen Gerät gesammelt werden können, wird letztlich durch die verwendete Android-Version und die zugrundeliegenden Hardwarekomponenten bedingt. Abhängig von einer spezifischen Anwendungsdomäne sind neben den oben aufgelisteten Informationen durchaus weitere Daten auf den Smartphones und Tablets sammelbar. Details dazu sind u. a. in [20] zu finden.

Die gesammelten Daten liegen typischerweise in Form der folgenden Datentypen vor: *String*, *Integer*, *Boolean* oder *Long*. Unter Android kann die Sammlung dieser Daten grundsätzlich auf drei verschiedene Arten erfolgen: einmalig/manuell, periodisch oder ereignisbasiert. *Statische* Systeminformationen, die sich nicht oder nur sehr selten verändern, wie z. B. die Modellbezeichnung, sollten einmalig oder nach Bedarf manuell gesammelt werden. Für *dynamische* System- und Anwendungsinformationen, wie z. B. die CPU- und RAM-Auslastung, eignet sich hingegen eine Auswertung in periodischen Zeitintervallen durch einen *TimerTask*. Einerseits sollten diese Zeitintervalle nicht zu kurz gewählt sein, um die Ressourcen der mobilen Geräte zu schonen. Andererseits dürfen sie aber auch nicht zu lang gewählt werden, da sonst keine zeitnahe Reaktion erfolgen kann. Zustandsänderungen des Betriebssystems, wie z. B. ein eingehender Anruf, werden ebenso wie sämtliche Sensorinformationen ereignisbasiert durch die Android-Plattform publiziert. Zur Erkennung von Veränderungen eines Sensorwerts muss dabei die entsprechende *Hook*-Methode des *Sensor Frameworks* in einem *Sensor*

Observer implementiert werden (siehe Abschnitt 2.2.4). Um Zustandsänderungen der Android-Plattform ermitteln zu können, müssen hingegen entsprechende *Broadcast Receiver* für die jeweiligen Systemmeldungen implementiert und registriert werden.

Im entwickelten IF-MAP Client soll zunächst jede gesammelte Information in ein entsprechendes Ereignis zur weiteren Verarbeitung durch die mobile CEP-Komponente transformiert werden. Dabei darf sich der Client jedoch ausdrücklich nicht auf eine bestimmte Anwendungsdomäne oder die oben aufgelisteten Smartphone-Daten beschränken. Stattdessen muss der eigene Client vielmehr auf einem flexibel anpass- und erweiterbaren Konzept basieren, das eine entsprechend auf die jeweilige Anwendungsdomäne zugeschnittene *Datensammlung*, *-verarbeitung* und *-übertragung* ermöglicht. Die in der Zielsetzung dieser Arbeit (siehe Abschnitt 1.2) beschriebene Notwendigkeit zur Sammlung, Verarbeitung und anschließender Übertragung einer reduzierten Menge verdichteter, sicherheitsrelevanter Smartphone-Daten im *CADS*-Umfeld ist dementsprechend nur eines von vielen Anwendungsszenarien, das durch den entwickelten IF-MAP Client unterstützt werden soll. So muss der eigene Client auch eine Unterstützung für diverse Anwendungsszenarien bieten, bei denen bestimmte Smartphone-Daten gesammelt, verarbeitet und übertragen werden sollen, deren Fokus nicht auf der Sicherheitsanalyse zur Integration von Smartphones und Tablets in Unternehmensnetze liegt. Um diese Flexibilität zu erreichen, sollen die Komponenten zur *Datensammlung*, *-verarbeitung* und *-übertragung* im entwickelten Client über schmale Schnittstellen lose miteinander gekoppelt werden. Dadurch kann z. B. die gesamte mobile CEP-Komponente gegen eine andere ersetzt werden. Weiterhin sollen die Komponenten zur *Datensammlung* und *-übertragung* über jeweilige *Factory-Patterns* entsprechend austauschbar gestaltet werden. Ein standardisierter *In*-Adapter für die *Ereignisverarbeitung* durch CEP sowie ein auf beliebige Smartphone-Daten anpassbares *Ereignismodell* müssen zusätzlich die Erweiterung des eigenen Clients zur Sammlung und Verarbeitung bisher nicht-beachteter Smartphone-Daten in Form von Ereignissen ermöglichen. Die Verarbeitung beliebiger Smartphone-Ereignisse soll durch ein deklaratives *Regelwerk* anhand von *Ereignisregeln* dynamisch anpass- und erweiterbar sein. Durch die ebenfalls mit Hilfe eines *Factory-Patterns* flexibel veränderbaren Komponenten zur *Datenübertragung* sollen die durch die mobile CEP-Komponente verdichteten Ereignisdaten der Smartphones und Tablets in ein beliebiges IF-MAP Vokabular bzw. IF-MAP Datenmodell transformiert und schließlich an einen MAP-Server übertragen werden können. Detailierte Informationen zu dem auf beliebige Anwendungsdomänen im Kontext von Smartphone-Daten anpass- und erweiterbaren Design des entwickelten IF-MAP Clients werden in Kapitel 5 aufgeführt.

4.2 Konstruktion eines Anwendungsszenarios

In diesem Abschnitt wird ein konkretes, domänenspezifisches Anwendungsszenario für den entwickelten IF-MAP Client vorgestellt, das beispielhaft die allgemeine Funktionsweise des Clients anhand einiger ausgewählter Sicherheitsinformationen der Smartphones und Tablets demonstriert. Damit bezieht sich dieses Anwendungsszenario direkt auf die in Abschnitt 1.2 erläuterte Notwendigkeit, einen IF-MAP Client im *CADS*-Umfeld ein-

zusetzen, der die übertragene Menge sicherheitsrelevanter Informationen an den MAP-Server durch eine auf den mobilen Geräten durchgeführte Datenverdichtung reduziert. Dabei sei nochmals darauf hingewiesen, dass es sich bei diesem Anwendungsszenario lediglich um ein Beispiel zur Veranschaulichung der Arbeitsweise des entwickelten Clients handelt. Letztlich ist der eigene Client so konzipiert, dass beliebige Smartphone-Daten unabhängig von einer konkreten Anwendungsdomäne gesammelt und verarbeitet werden können. Auch die *Datenübertragung* an den MAP-Server ist ausdrücklich nicht an ein konkretes IF-MAP Datenmodell gebunden. Stattdessen besitzt der eigene Client ein flexibles Konzept zur Anpassung und Erweiterung an konkrete Anwendungsdomänen, das in Kapitel 5 detailiert beleuchtet wird. Der ebenfalls in dieser Arbeit entwickelte Prototyp des Clients implementiert schließlich die in diesem Anwendungsszenario aufgezeigte Funktionalität als ein konkretes Beispiel zur Demonstration der Funktionsweise. Das vorgestellte Anwendungsszenario zur Sammlung, Verarbeitung und Übertragung sicherheitsrelevanter Smartphone-Daten basiert im Wesentlichen auf einer Teilmenge der von Bente in [5] aufgeführten Szenarien im Kontext des *CADS*-Systems. Die sich aus diesem Anwendungsszenario ergebenden einzelnen Anwendungsfälle bilden letztlich neben der in Abschnitt 1.2 beschriebenen Zielsetzung die zentrale Grundlage für die Analyse der funktionalen und nichtfunktionalen Anforderungen an den in der vorliegenden Arbeit entwickelten IF-MAP Client (siehe Abschnitt 4.3).

Dieses Szenario gliedert den entwickelten IF-MAP Client als *Feature Collector* in das *CADS*-System ein (siehe Abschnitt 3.1.2). Wie Abbildung 4.1 zeigt, werden die gesammelten und verdichteten Sicherheitsinformationen entsprechend an den MAP-Server *irond* übertragen und können anschließend u. a. durch *irondetect* analysiert oder durch *VisITMeta* visualisiert werden. Die *Datensammlung* umfasst dabei eine Teilmenge der in Abschnitt 4.1 aufgelisteten Informationen, die eine Sicherheitsanalyse der mobilen Geräte ermöglichen. Während die Geräte für andere Zwecke verwendet werden, läuft die *Datensammlung*, *-verarbeitung* und *-übertragung* in Echtzeit im Hintergrund ab.

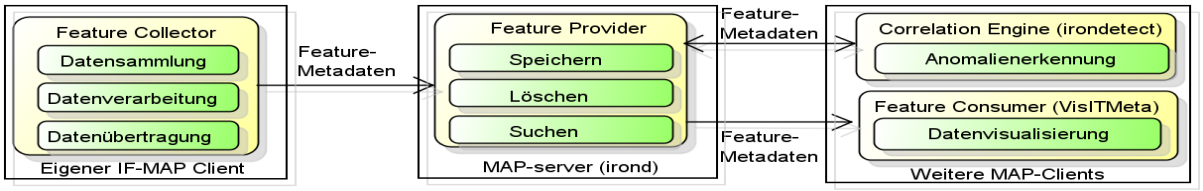

Abbildung 4.1: Einordnung des entwickelten Clients in das *CADS*-System

Das vorgestellte Anwendungsszenario zur Sammlung, Verarbeitung und Übertragung sicherheitsrelevanter Smartphone-Daten unterteilt sich im Kontext des in dieser Arbeit entwickelten IF-MAP Clients in vier einzelne Anwendungsfälle, die jeweils einen bestimmten Teilaspekt der Funktionalität des Clients umfassen. Diese einzelnen Anwendungsfälle werden nachfolgend detailiert erläutert.

Der erste Anwendungsfall ermöglicht anderen Komponenten die Ermittlung allgemeiner Informationen der mobilen Geräte, um z. B. Restriktionen für bestimmte Gerätetypen oder Betriebssystem-Versionen umzusetzen. Auf den Smartphones und Tablets werden

dazu vom Prototyp des Clients u. a. die folgenden *statischen* Systeminformationen gesammelt: Hersteller, Modellbezeichnung, Firmware- und Kernel-Version. Die daraus abgeleiteten Ereignisse werden durch CEP-*Ereignisregeln gefiltert*, *angereichert* und zu komplexen Ereignissen *korreliert*. Abschließend erfolgt die Transformation dieser komplexen Ereignisse in IF-MAP Vokabular des *Feature-Metadatenmodells* aus Abschnitt 3.1.2 sowie die *Datenübertragung* an den MAP-Server. Dadurch können z. B. Geräte mit veralteten Betriebssystemen aus dem Unternehmensnetz ausgeschlossen werden. In einem konkreten Fall könnte beispielsweise dem Smartphone eines Mitarbeiters des Unternehmens der Zugriff auf bestimmte sicherheitskritische Ressourcen verwehrt bleiben, solange dieses mit einer zu alten Android-Version betrieben wird. Erst durch die Aktualisierung des Betriebssystems seitens des Mitarbeiters würde der Zugang zu unternehmenskritischen Ressourcen freigegeben werden.

Der zweite Anwendungsfall legt den Fokus auf *Kontextinformationen*. Der Prototyp des Clients verwendet die Systemzeit sowie die Position der Smartphones und Tablets als *Kontextinformationen* für die gesammelten Daten. Diese *Kontextinformationen* werden dabei durch CEP-*Ereignisregeln* abstrahiert bzw. anonymisiert, sodass die Auswertung sicherheitsrelevanter Informationen z. B. nur dann ausgeführt wird, wenn sich das jeweilige Smartphone oder Tablet während der Arbeitszeit auf dem Unternehmensgelände befindet. Sobald sich ein Smartphone oder Tablet außerhalb des Unternehmensstandortes bzw. der Arbeitszeit befindet, werden entsprechend keine Informationen zur Sicherheitsanalyse an den MAP-Server bereitgestellt. Weiterhin werden durch die Anonymisierung der Position Datenschutzkonflikte vermieden. In einem konkreten Fall würde beispielsweise die Sicherheitsanalyse der Daten des Smartphones eines Mitarbeiters ausschließlich dann ausgeführt werden, wenn sich das mobile Gerät zu bestimmten Zeitpunkten an bestimmten Orten des Unternehmens befindet. Zusätzlich würde die Position des Smartphones abstrahiert werden, sodass kein exaktes Aufenthaltsprofil des Mitarbeiters angelegt werden kann.

Der dritte Anwendungsfall dient der Erkennung eines abnormalen Smartphone-Verhaltens. Dazu sammelt der Prototyp des Clients die folgenden *dynamischen* System- und Sensorinformationen: CPU- und RAM-Auslastung, Anzahl laufender Prozesse, Aktivitäten von Netzwerkschnittstellen und Sensoren (Lichtsensor, Kamera, An- und Ausschalten des Bildschirms) sowie Statistiken zum Netzwerkverkehr und Akkuverbrauch. Diese Informationen werden durch CEP zu komplexen Ereignissen verdichtet, bevor sie in IF-MAP Vokabular des *Feature-Metadatenmodells* transformiert und an den MAP-Server übertragen werden. Die CEP-*Ereignisregeln* umfassen dabei Mechanismen zur *Filterung*, *Anreicherung*, *Aggregation* und *Korrelation* (siehe Abschnitt 2.3.4). Sobald ein Smartphone oder Tablet z. B. ungewöhnlich viel Systemlast besitzt, veröffentlicht der Prototyp des Clients die jeweilige Information auf dem MAP-Server. Dazu zeigt Abbildung 4.2 den Ablauf zur Erkennung einer hohen Systemauslastung und der entsprechenden Reaktion als Beispiel für die Verarbeitung *dynamischer* Systeminformationen. In einem konkreten Fall könnte die Systemauslastung des Smartphones eines Mitarbeiters über längere Zeiträume analysiert werden, sodass Auffälligkeiten in Form von unerwarteten Veränderungen der Systemauslastung Hinweise auf Schadsoftware bilden und u. a. zur Alarmierung eines Administrators führen.

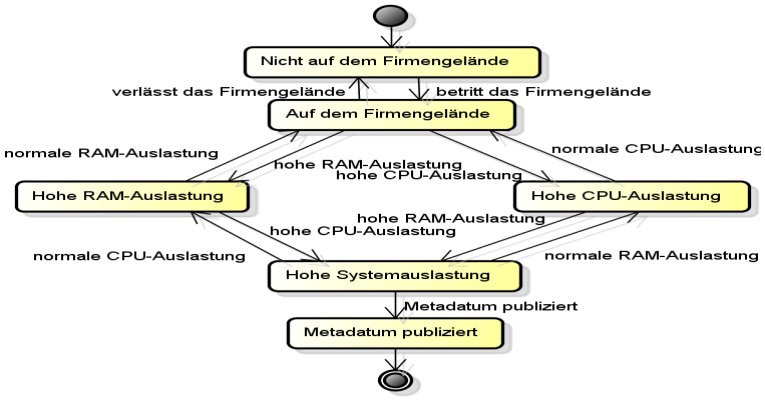

Abbildung 4.2: Ablauf zur Erkennung und Verarbeitung einer hohen Systemauslastung

Der vierte Anwendungsfall dient zur Erkennung unerwünschter Anwendungen. Auf den Smartphones und Tablets werden dazu vom Prototyp des Clients Informationen zu den installierten *Apps*, verwendeten *Permissions* und laufenden Prozessen gesammelt. Der Prototyp erzeugt auch Benachrichtigungen, sobald eine *App* installiert oder deinstalliert wird. Solche Anwendungsdaten werden dabei durch CEP-*Ereignisregeln gefiltert*, *angereichert* und *korreliert*. Mit *irondetect* können u. a. Richtlinien zu unerwünschten *Permissions* oder *Apps* spezifiziert werden, wodurch mobile Geräte, die gegen diese Richtlinien verstoßen, keinen Zugang zu unternehmenskritischen Ressourcen erhalten. In einem konkreten Fall könnte beispielsweise die Erkennung einer unerwünschten *App* auf dem Smartphone eines Mitarbeiters des Unternehmens dazu führen, dass dem jeweiligen Smartphone der Zugang zu sicherheitskritischen Ressourcen verwehrt bleibt. Erst durch die Deinstallation der entsprechenden unerwünschten *App* seitens des Mitarbeiters würde der Zugang zu den unternehmenskritischen Ressourcen freigegeben werden.

Der Prototyp bezieht alle Informationen durch spezifische *Datensammlungs*-Klassen, die über *Factory-Patterns* austauschbar gestaltet sind. Das Szenario umfasst sämtliche in Abschnitt 4.1 aufgeführten Informationskategorien, wobei die *Datensammlung* sowohl einmalig/manuell als auch periodisch durch einen *TimerTask* sowie ereignisbasiert durch *Sensor Observer* und *Broadcast Receiver* erfolgt. Dementsprechend bietet dieses Szenario eine vollständige Demonstration für die Funktionsweise der *Datensammlung* des IF-MAP Clients. Sämtliche gesammelten Informationen werden in Ereignisse des in Abschnitt 5.2 erläuterten *Ereignismodells* transformiert und über einen *In*-Adapter an die mobile CEP-Komponente übergeben. Die *Ereignisregeln* des Szenarios veranschaulichen dabei diverse Schritte zur Verdichtung der gesammelten Daten: *Filterung*, *Anreicherung*, *Anonymisierung*, *Aggregation*, *Korrelation* und *Synthese* komplexer Ereignisse. Abschließend werden ausschließlich Informationen mit einem hohen fachlichen Informationsgehalt an den MAP-Server bereitgestellt, wodurch sich die Anzahl der übertragenen Daten im Vergleich zu einem IF-MAP Client ohne mobile CEP-Komponente deutlich verringert. Zur Transformation der Ereignisse in IF-MAP Vokabular verwendet der Prototyp das *Feature-Metadatenmodell* aus Abschnitt 3.1.2. Ein weiteres *Factory-Pattern* ermöglicht dabei allerdings auch die Verwendung gänzlich verschiedener Datenmodel-

le zur Repräsentation der Smartphone-Daten. Sämtliche Komponenten des Prototyps sind lose miteinander gekoppelt, wodurch einzelne Bestandteile der *Datensammlung*, *-verarbeitung* und *-übertragung* problemlos angepasst, erweitert oder komplett ersetzt werden können.

4.3 Anforderungen an den entwickelten IF-MAP Client

In diesem Abschnitt wird eine Auflistung der Anforderungen an den entwickelten IF-MAP Client vorgestellt. Die Anforderungen leiten sich dabei aus der Identifikation der wichtigsten Informationen heutiger Smartphones und Tablets (siehe Abschnitt 4.1), dem erläuterten Anwendungsszenario (siehe Abschnitt 4.2), der Zielsetzung dieser Arbeit (siehe Abschnitt 1.2) und der Analyse bestehender Lösungsstrategien (siehe Kapitel 3) ab. Dazu erfolgt eine Unterteilung in funktionale und nichtfunktionale Anforderungen an den entwickelten IF-MAP Client.

4.3.1 Funktionale Anforderungen

Funktionale Anforderungen an die Datensammlung

- Der Client soll die Sammlung beliebiger Sensor-, System-, Kontext- und Anwendungsinformationen der mobilen Geräte unterstützen. Dazu sollen die Mechanismen zur *Datensammlung* entsprechend einer konkreten Anwendungsdomäne austausch-, anpass- und erweiterbar gestaltet sein. Weitere Komponenten, beispielsweise zum Auslesen externer Sensoren, sollen einfach ergänzt werden können.

- Die *Datensammlung* soll abhängig von der Hardwarekonfiguration und der verwendeten Betriebssystem-Version erfolgen, wobei die begrenzten Hardwareressourcen nicht übermäßig beansprucht werden dürfen. Dazu soll der Client verschiedene Arten zur Sammlung der Daten unterstützen: einmalig/manuell, periodisch über *TimerTasks* und ereignisbasiert über *Sensor Observer* oder *Broadcast Receiver*.

Funktionale Anforderungen an die Datenverarbeitung

- Die gesammelten Daten sollen in Ereignisse eines beliebig erweiterbaren *Ereignismodells* transformiert und anschließend zur Verarbeitung über eine standardisierte, schmale Schnittstelle an eine mobile CEP-Komponente übergeben werden.

- Die *Datenverarbeitung* soll durch eine mobile CEP-Komponente in nahezu Echtzeit erfolgen. Das deklarative *Regelwerk* sowie die Verarbeitungsschritte sollen dabei entsprechend einer konkreten Anwendungsdomäne flexibel angepasst und um komplexe Fachlogik (z. B. *Ereignismuster*) beliebig erweitert werden können.

- Die mobile CEP-Komponente soll die gesammelten, feingranularen und kontinuierlich eintreffenden Daten zu komplexen Ereignissen mit einem hohen fachlichen

Informationsgehalt verdichten. Dazu sollen die aus den gesammelten Daten transformierten Einzelereignisse mittels komplexer Fachlogik *gefiltert*, *angereichert*, *anonymisiert*, *aggregiert*, *korreliert* und zu komplexen Ereignissen zusammengefasst werden. Die CEP-Komponente soll dabei große Datenvolumina beherrschen.

- Die gesamte mobile CEP-Komponente soll aufgrund von standardisierten *In-* und *Out*-Adaptern gegen eine andere ausgetauscht werden können. Diese schmalen Schnittstellen sollen die Komponenten des Clients lose miteinander koppeln.

Funktionale Anforderungen an die Datenübertragung

- Die *Datenübertragung* soll an das zugrundeliegende IF-MAP Umfeld angepasst werden können. Dazu sollen die Verbindungseinstellungen zum MAP-Server entsprechend der vorhandenen IT-Infrastruktur verändert werden können, wodurch sich der Client in ein bereits bestehendes IF-MAP Umfeld eingliedern kann.

- Der Mechanismus zur Transformation der durch CEP verdichteten Ereignisse in IF-MAP Vokabular sowie zur Kommunikation mit dem MAP-Server soll entsprechend einer konkreten Anwendungsdomäne austausch-, anpass- und erweiterbar sein.

- Die Menge der an den MAP-Server übertragenen Daten soll möglichst gering gehalten werden, um die Ressourcen der Smartphones und Tablets, des MAP-Servers sowie des zugrundeliegenden Netzwerks zu schonen. Dazu soll der Client ausschließlich Informationen übertragen, die zuvor durch die mobile CEP-Komponente verdichtet wurden. Fachlich relevante Informationen sollen anderen Netzwerkkomponenten zeitnah und ohne Verzögerung zur Verfügung gestellt werden.

- Der Client soll Datenschutzkonflikte verhindern, indem personenbezogene Daten bereits vor oder während der Verarbeitung durch die mobile CEP-Komponente anonymisiert werden. Bei der anschließenden *Datenübertragung* sollen somit keine personenbezogenen Daten mit dem MAP-Server ausgetauscht werden.

Weitere funktionale Anforderungen

- Der Client soll im Kontext der Sammlung, Verarbeitung und Übertragung von Smartphone-Daten unabhängig von einer spezifischen Anwendungsdomäne sein.

- Der Client soll manuell über eine *App* durch den Benutzer gestartet oder gestoppt werden können. Während des Betriebs soll der Benutzer darüber informiert werden, ob der Client läuft und welche Daten übertragen werden.

- Im speziellen Kontext des *CADS*-Systems soll der Client die Sammlung, Verarbeitung und Übertragung sicherheitsrelevanter Daten ermöglichen. Durch die Verdichtung der Informationen mit Hilfe der mobilen CEP-Komponente sollen die an den MAP-Server übertragenen Daten, im Vergleich zu einem IF-MAP Client ohne mobile CEP-Komponente, möglichst stark reduziert und anonymisiert werden.

Der Client soll dabei Sensor-, System-, Kontext- und Anwendungsdaten zeitnah auf dem MAP-Server bereitstellen, sodass andere Netzwerkkomponenten in der Lage sind, einerseits allgemeine Informationen der Smartphones und Tablets zu ermitteln und andererseits Anomalien oder unerwünschte *Apps* zu erkennen.

4.3.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen an das Design der Anwendung

- Der Client soll für die Android-Plattform in *Java* entwickelt werden und muss kompatibel zu den derzeit am weitesten verbreiteten Android-Versionen und Gerätetypen sein. Darüber hinaus sollen auch explizit Tablets unterstützt werden.
- Der Client soll in Form einer leichtgewichtigen *App* entwickelt werden. Dies ermöglicht die einfache Installation, Aktualisierung und Deinstallation des Clients.
- Der Client soll ausschließlich *Permissions* verwenden, um Smartphone-Daten sammeln zu können. Ein *Root-Zugriff* oder andere Modifikationen am Betriebssystem dürfen keine Voraussetzungen zur Lauffähigkeit des Clients sein. Die *Datensammlung* sollte typischerweise durch Dienste und Schnittstellen des *Application Frameworks* sowie durch registrierte *Sensor Observer* und *Broadcast Receiver* erfolgen.
- Die Komponenten der Benutzeroberfläche des Clients sollten in XML-Dokumenten getrennt vom *Java*-Quellcode spezifiziert werden.

Nichtfunktionale Anforderungen an die Ausführung der Anwendung

- Der Client soll schonend mit den Ressourcen der Smartphones umgehen. Dies betrifft neben der CPU- und RAM-Auslastung vor allem auch den Energieverbrauch.
- Der Client soll auf den mobilen Geräten kontinuierlich im Hintergrund ausgeführt werden. Eine Interaktion durch den Benutzer sollte ausschließlich zum Starten bzw. Stoppen des Clients oder zur Kontrolle der übertragenen Daten nötig sein.
- Die *Datenverarbeitung* durch CEP soll direkt auf den Smartphones und Tablets durch eine mobile *Event Processing Engine* wie *Asper* erfolgen. Die verwendeten *Ereignistypen* sollten dabei z. B. in Form von *Java*-Beans spezifiziert werden.
- Die *Datenübertragung* soll über das IF-MAP Protokoll erfolgen. Unter Android muss dazu die Netzwerkkommunikation in *AsyncTasks* ausgelagert werden, damit der *Main-Thread* nicht durch langlaufende Netzwerkoperationen blockiert wird.
- Durch CEP und IF-MAP sollten große Datenmengen in nahezu Echtzeit gesammelt, verdichtet und an den MAP-Server übertragen werden können.
- Personenbezogene Daten sollen vor der Übertragung an den MAP-Server anonymisiert werden. Dazu zählt z. B. die aktuelle Position der mobilen Geräte. Die gesammelten Daten dürfen zudem nicht durch andere *Apps* ausgespäht werden.

4.4 Analyse von Datenschutzaspekten

Der Nachteil einiger im Kontext des entwickelten IF-MAP Clients bestehender Lösungen ist die unzureichende Beachtung von Datenschutzaspekten (siehe Kapitel 3). Im Folgenden wird daher analysiert, wann Smartphone-Daten durch den eigenen Client anonymisiert werden müssen, bevor sie an den MAP-Server übertragen werden.

Bei einer umfangreichen *Datensammlung*, wie sie vom entwickelten Client durchgeführt werden kann, sollte der Schutz der Identität der Smartphone-Benutzer im Vordergrund stehen. Einerseits müssen dazu personenbezogene Daten, die direkte Rückschlüsse auf die Benutzeridentität zulassen, anonymisiert werden. Andererseits muss aber auch sichergestellt werden, dass die Benutzeridentität nicht durch die Analyse vieler Einzelinformationen rekonstruiert werden kann. Da die an den MAP-Server publizierten Daten für jede Netzwerkkomponente im IF-MAP Umfeld zugänglich sind, muss eine entsprechende Anonymisierung bereits vor der *Datenübertragung* direkt auf den Smartphones durchgeführt werden. Beim Betrieb des Clients verlassen die personenbezogenen Daten also in keinem Moment das zugrundeliegende Gerät. Der Prototyp des entwickelten Clients führt dazu z. B. eine Anonymisierung der Geräteposition als Kontextparameter aller gesammelter Daten durch. Dabei wird die exakte Position der Smartphones in einem *Anreicherungsschritt* durch die mobile CEP-Komponente in anonymisierte Zustände wie *restricted_area* oder *no_restricted_area* überführt (siehe Kapitel 6). Somit wird verhindert, dass aus den Kontextparametern der publizierten Informationen ein auf genaue Aufenthaltsorte bestimmbares Verhaltensprofil der Smartphone-Benutzer angelegt werden kann. Der Prototyp des entwickelten Clients zeigt anhand der IMEI und IMSI auch, dass die Anonymisierung ebenfalls durch *Hashing-* oder *Verschlüsselungsalgorithmen* außerhalb der mobilen CEP-Komponente erfolgen kann. Während der *Datensammlung* muss zudem gewährleistet sein, dass die Informationen nicht durch andere *Apps* ausgespäht werden können. Darüber hinaus sollte der Benutzer zu jeder Zeit während des Betriebs des eigenen Clients über die übertragenen Daten informiert werden können.

Im Folgenden wird eine exemplarische Auswahl der Smartphone-Daten vorgestellt, die bei der *Datenübertragung* an den MAP-Server Konflikte mit Datenschutzbestimmungen verursachen könnten: GPS-Aufenthaltsort, Systemzeit, IMEI, IMSI, Namen, Adressen, Telefonnummern, E-Mail-Konten, Internet-Lesezeichen, Kontaktbuch- und Kalender-Informationen, Chat-Protokolle sowie SMS-Nachrichten.

5 Konzepte des entwickelten IF-MAP Clients

Nachdem im vorherigen Kapitel die Anforderungen an den eigenen IF-MAP Client spezifiziert wurden, erfolgt in diesem Kapitel der konzeptionelle Entwurf des Clients. Dazu wird in Abschnitt 5.1 zunächst ein Überblick über den allgemeinen Aufbau des Clients vorgestellt, wobei die zentralen Komponenten mit ihren jeweiligen Verantwortlichkeiten beleuchtet werden. In Abschnitt 5.2 werden die Strategien zur Sammlung beliebiger Sensor-, System-, Kontext- und Anwendungsinformationen erläutert. Die Möglichkeiten zur Verarbeitung der gesammelten Informationen mittels einer CEP-Komponente und eines entsprechenden *Regelwerks* werden in Abschnitt 5.3 aufgezeigt. Das Kapitel schließt mit der Analyse von Lösungsstrategien zur Transformation von Ereignissen in IF-MAP Vokabular sowie zur *Datenübertragung* an den MAP-Server in Abschnitt 5.4.

5.1 Allgemeiner Aufbau

Der Client wird als leichtgewichtige *App* für die Android-Plattform entworfen. Dabei unterstützt er sämtliche aktuellen Android-Versionen und kann sowohl auf Smartphones als auch auf Tablets eingesetzt werden. Für die *Datensammlung* sind keine Modifikationen am Betriebssystem (z. B. *Root-Zugriff* o. Ä.) nötig. Um seine Funktionalität auch im Hintergrund anbieten zu können, besteht der Client aus zwei Hauptkomponenten. Einerseits bildet die *Activity* `MainActivity` den Einstiegspunkt in die Client-Anwendung und dient zur Darstellung der Benutzeroberfläche, über die die Funktionen des eigenen Clients gesteuert werden können. Andererseits stellt der lokale *Service* `MainService` die zentrale Komponente zur Durchführung der *Datensammlung*, *-verarbeitung* und *-übertragung* im Hintergrund dar (siehe Abbildung 5.1).

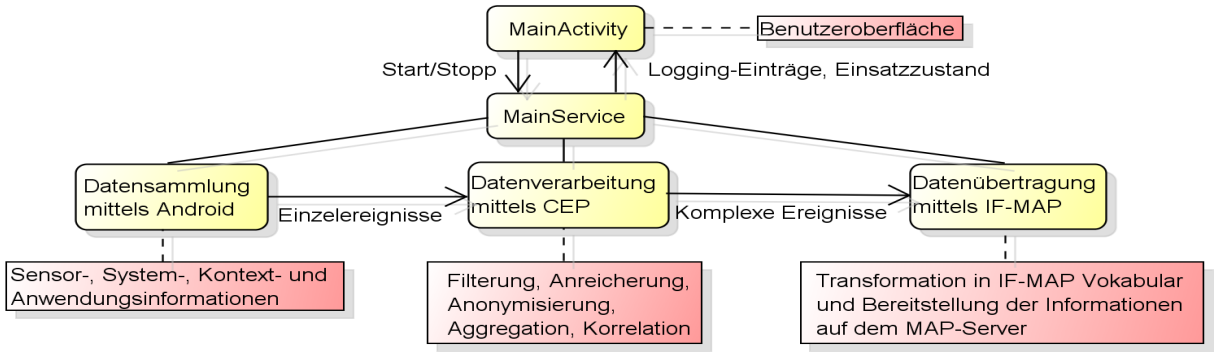

Abbildung 5.1: Schematischer Aufbau des entwickelten Clients

Diese beiden Hauptkomponenten sind sehr lose miteinander gekoppelt und besitzen jeweils ihren eigenen Lebenszyklus, wobei der Datenaustausch über *Intents* erfolgt. Über die `MainActivity` kann die Ausführung des Clients mit Hilfe entsprechender *Buttons* gesteuert werden. Ein *TextView* und eine *ProgressBar* zeigen dabei den aktuellen Einsatzzustand des Clients an. Zudem werden sämtliche an den MAP-Server übertragenen Informationen in einer Logging-Ausgabe dargestellt. Über einen weiteren *Button* kann die Sammlung, Verarbeitung und Übertragung *statischer* Systeminformationen manuell angestoßen werden. Der Ausführungsstatus des `MainService` wird durch die `MainActivity` über *Intents* reguliert. Zum Empfang des Einsatzzustands und der Logging-Einträge verwendet die `MainActivity` zwei *Broadcast Receiver*, die auf entsprechende *Intents* des `MainService` reagieren. Abbildung 5.2 zeigt dazu die Kommunikation zwischen der `MainActivity` und dem `MainService` mittels *Intents*, während Abbildung 5.3 den Lebenszyklus des `MainService` verdeutlicht. Durch die dynamische Anpassung der *Buttons* auf den jeweiligen Einsatzzustand des Clients verhindert die `MainActivity` zudem Falscheingaben. Der Zustand der Benutzeroberfläche wird über ein *Bundle* verwaltet, sodass die `MainActivity` auch nach dem Schließen und erneutem Starten den korrekten Einsatzzustand sowie sämtliche Logging-Ausgaben anzeigt.

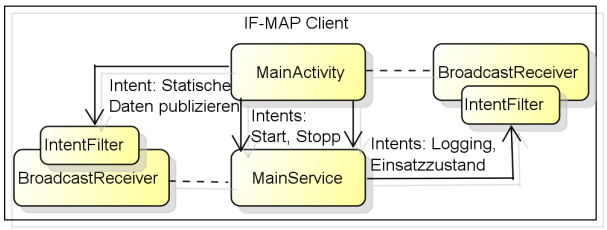

Abbildung 5.2: Kommunikation zwischen `MainActivity` und `MainService` über *Intents*

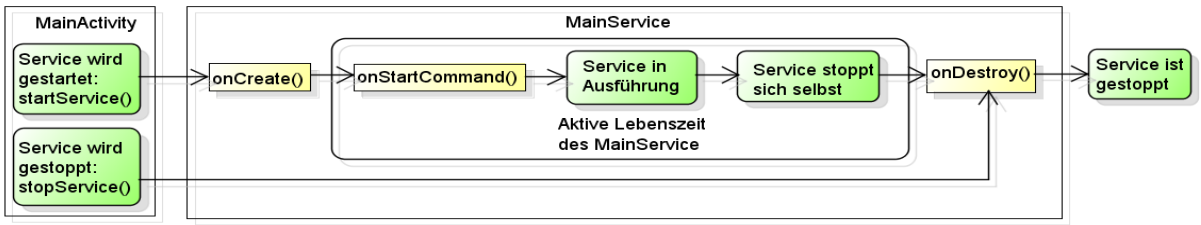

Abbildung 5.3: Lebenszyklus des `MainService`

Der `MainService` bildet die zentrale Komponente zur Sammlung, Verarbeitung und Übertragung von Daten im Hintergrund. Bei seinem Start initialisiert der `MainService` die Verbindung zum MAP-Server mit Hilfe eines *AsyncTasks* und der *ifmapj*-Bibliothek (siehe Abschnitt 5.4). Über zwei *Factory-Patterns* werden zudem verschiedene `Feature-Provider`- und `FeatureCollector`-Klassen als Schnittstellen zur Android-Plattform sowie zur Sammlung beliebiger System- und Anwendungsinformationen erzeugt. Weiterhin instanziiert und registriert der `MainService` *Sensor Observer* und *Broadcast Receiver*, um beliebige Sensorinformationen bzw. Informationen zu Zustandsänderungen

des Betriebssystems ereignisbasiert zu sammeln. Insgesamt bietet der Client damit ein flexibles Konzept zur Anpassung und Erweiterung der *Datensammlung* auf beliebige Anwendungsszenarien (siehe Abschnitt 5.2). Neben den Komponenten zur *Datensammlung* wird beim Start des `MainService` auch die mobile CEP-Komponente in Form eines EPAs instanziiert. Dabei werden entsprechend der Anwendungsdomäne die verwendeten *Ereignistypen* als *Java*-Beans sowie die für die *Ereignisverarbeitung* spezifizierten *Ereignisregeln* in Form von *Strings* beim EPA registriert. Je nach Anwendungsfall werden zudem im `MainService` verschiedene Listener zur *Ereignisbehandlung* angelegt und mit bestimmten *Ereignisregeln* verknüpft. Diese Listener, ein ebenfalls im `MainService` erzeugter, standardisierter *In*-Adapter und ein auf beliebige Anwendungsdomänen erweiterbares *Ereignismodell* ermöglichen die lose Kopplung zwischen der *Datensammlung*, *-verarbeitung* und *-übertragung*, wodurch die gesamte mobile CEP-Komponente gegen eine andere ausgetauscht werden könnte (siehe Abschnitt 5.3). Zuletzt erzeugt der `MainService` über ein weiteres *Factory-Pattern* eine konkrete Instanz des `MetadataProviders` zur Transformation der verdichteten Ereignisse aus der *Datenverarbeitung* in IF-MAP Vokabular einer entsprechenden Anwendungsdomäne sowie zur Übertragung der generierten *Metadaten* an einen MAP-Server. Auch dabei ermöglicht dieses *Factory-Pattern* ein flexibles Konzept zur Anpassung und Erweiterung des Clients an eine spezifische IF-MAP Umgebung bzw. ein spezifisches IF-MAP Datenmodell (siehe Abschnitt 5.4). Der `MainService` verwaltet weiterhin den *TimerTask* `MainTimerTask`, der in periodischen Zeitintervallen den `DynamicSystemFeatureCollector` zur Sammlung *dynamischer* Systeminformationen aufruft. Der `StaticSystemFeatureCollector` zur Sammlung *statischer* Systeminformationen wird hingegen einmalig beim Start des `MainService` aufgerufen. Zusätzlich besitzt der `MainService` einen *Broadcast Receiver*, der auf *Intents* der `MainActivity` reagiert, um die Sammlung *statischer* Systeminformationen manuell aus der Benutzeroberfläche anzustoßen. Der `MainService` bietet zudem die Möglichkeit, Zustandsinformationen und Logging-Einträge über *Intents* an die `MainActivity` zu senden. Damit der `MainService` nicht unerwartet während der Durchführung seiner Aufgaben durch die Android-Plattform beendet wird, muss er mit der Priorität einer im Vordergrund befindlichen Anwendung ausgeführt werden. Dazu wird bei seinem Start eine *Notification* erzeugt, wodurch der laufende Client jederzeit durch eine Meldung in der Statuszeile des Betriebssystems angezeigt wird und damit die gleiche Priorität einer im Vordergrund ausgeführten Anwendung besitzt. Des Weiteren dient der `MainService` auch zur Verwaltung der aktuellen Geräteposition. Dazu definiert er die beiden Attribute `locationX` und `locationY` sowie einen `LocationObserver`, der in regelmäßigen Abständen diese beiden Koordinaten mit Hilfe des GPS-Sensors oder des Netzwerkstandortes aktualisiert. Außerdem wird eine Instanz des `Anonymizers` erzeugt, der personenbezogene Daten wie z. B. die IMEI über einen *Hashing*-Algorithmus anonymisieren kann. Schließlich bietet der `MainService` den Komponenten der *Datensammlung*, *-verarbeitung* und *-übertragung* den Zugriff auf die bei seinem Start instanziierten Objekte an. Beim Beenden des Clients werden die Verbindung zum MAP-Server abgebaut, die ereignisbasierte *Datenverarbeitung* des EPAs beendet und die *Sensor Observer* sowie die *Broadcast Receiver* gestoppt, damit die Ressourcen der mobilen Geräte nicht unnötig belastet werden.

Die Arbeitsweise des Clients umfasst mehrere Schritte. `FeatureProvider`-Klassen, *Sensor Observer* und *Broadcast Receiver* bilden die Schnittstellen zur Android-Plattform, um beliebige Smartphone-Daten zu erfassen. Über `FeatureCollector`-Klassen werden die benötigten Daten manuell oder periodisch gesammelt, in Ereignisse transformiert und über einen *In*-Adapter an die *Datenverarbeitung* übergeben. *Sensor Observer* und *Broadcast Receiver* sammeln die benötigten Daten hingegen ereignisbasiert und übermitteln diese nach der jeweiligen Ereignistransformation direkt über den *In*-Adapter an die *Datenverarbeitung*. Der *In*-Adapter fügt somit sämtliche aufgetretenen Ereignisse auf standardisierte Weise in den Ereignisstrom zur Verarbeitung durch die mobile CEP-Komponente ein. Bei der *Datenverarbeitung* werden die Ereignisse durch deklarative *Ereignisregeln gefiltert*, *angereichert*, *anonymisiert*, *aggregiert*, *korreliert* und zu komplexen Ereignissen verdichtet. Komplexe Ereignisse stoßen wiederum Listener an, die die verdichteten Informationen durch einen für die jeweilige Anwendungsdomäne spezifizierten `MetadataProvider` in IF-MAP Vokabular transformieren und anschließend über *AsyncTasks* an einen MAP-Server senden. Abbildung 5.4 zeigt die fachliche Architektur des entwickelten Clients, die letztlich auch vom implementierten Prototyp umgesetzt wird. Dabei werden sowohl die einzelnen Arbeitsschritte von der *Datensammlung* bis hin zur *Datenübertragung* als auch die jeweiligen Datentransformationen zwischen den Arbeitsschritten verdeutlicht.

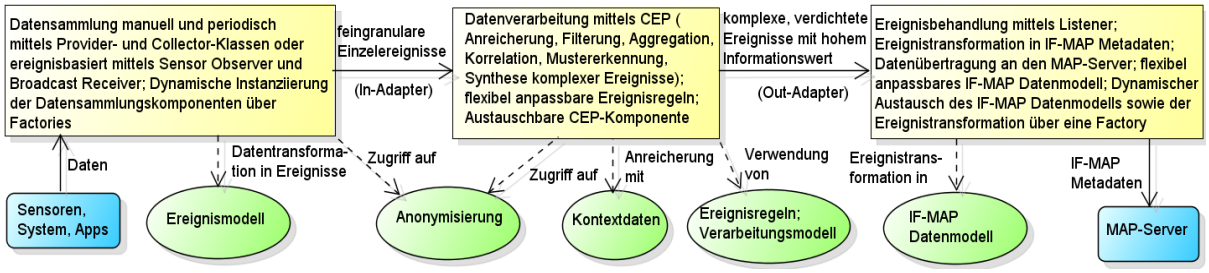

Abbildung 5.4: Fachliche Architektur des entwickelten Clients

5.2 Datensammlung

Diese Schicht umfasst die Komponenten zur Sammlung beliebiger Sensor-, System-, Kontext- und Anwendungsinformationen, deren Transformation in entsprechende Ereignisse und Übergabe an die mobile CEP-Komponente zur weiteren Verarbeitung. Damit die *Datensammlung* flexibel auf beliebige Anwendungsdomänen angepasst und erweitert werden kann, besitzt sie zwei *Factory*-Klassen: `FeatureProviderFactory` und `FeatureCollectorFactory` zur Erzeugung konkreter, auf das jeweilige Anwendungsszenario zugeschnittener `FeatureProvider`- und `FeatureCollector`-Objekte. Über jede *Factory*-Klasse können drei verschiedene `FeatureProvider`- bzw. `FeatureCollector`-Instanzen erzeugt werden. Die `FeatureProvider`-Objekte bilden die Schnittstellen zur Android-Plattform über Dienste des *Application Frameworks* (siehe Abschnitt 2.2), während die `FeatureCollector`-Objekte für die eigentliche *Datensammlung* zuständig

sind. Zur einmaligen/manuellen Sammlung *statischer* Systeminformationen werden konkrete Klassen erstellt, die die `StaticSystemFeatureProvider/Collector`-Interfaces implementieren. Die *Datensammlung* wird dabei durch den `MainService`, z. B. beim Start des Clients, angestoßen. Für die periodische Sammlung *dynamischer* Systeminformationen werden konkrete Klassen implementiert, die auf die `DynamicSystemFeature-Provider/Collector`-Interfaces aufbauen. Die *Datensammlung* wird dabei durch den `MainTimerTask` in regelmäßigen Zeitabständen angestoßen. Anwendungsinformationen werden über konkrete Ausprägungen der `ApplicationFeatureProvider/Collector`-Interfaces bezogen. Somit geben diese Interfaces die Strukturen der für die entsprechenden Anwendungsdomänen implementierten `FeatureProvider`- und `FeatureCollector`-Klassen vor. Deren Instanziierung über die *Factory*-Klassen entkoppelt die übrigen Komponenten des Clients von der jeweils konkreten Implementierung. Bei der Anpassung des Clients an eine andere Anwendungsdomäne können die entsprechend implementierten `FeatureProvider`- und `FeatureCollector`-Klassen über diese *Factory-Patterns* erzeugt werden, sodass keine weiteren Komponenten des Clients verändert werden müssen. Abbildung 5.5 zeigt exemplarisch das *Factory-Pattern* zur Erzeugung von `DynamicSystemFeatureCollector`-Komponenten innerhalb des entworfenen Clients.

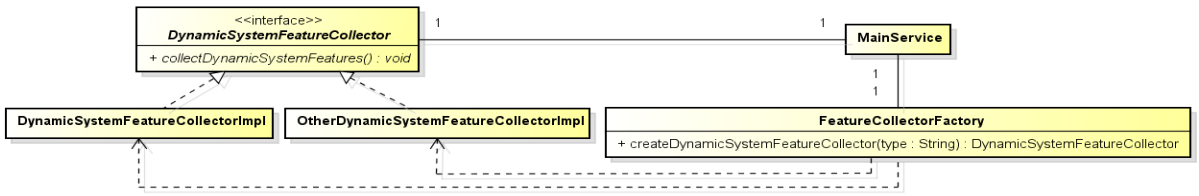

Abbildung 5.5: Umsetzung eines *Factory-Patterns* bei der *Datensammlung*

Zur ereignisbasierten Sammlung von Sensorinformationen bzw. Informationen über Zustandsänderungen des Betriebssystems werden *Sensor Observer* und *Broadcast Receiver* implementiert und für beliebige Sensoren bzw. Systemmeldungen registriert. Ein *Factory-Pattern* ist dabei nicht notwendig, da diese Komponenten bereits dynamisch durch die Android-Plattform verwaltet und aufgerufen werden. Einerseits müssen dazu *Hook*-Methoden implementiert werden, die durch das Android-Betriebssystem entsprechend angestoßen werden können. Andererseits werden diese Komponenten bei ihrer Instanziierung für bestimmte Sensoren bzw. Systemmeldungen in der Android-Plattform registriert. `FeatureProvider`-Klassen werden dabei nicht benötigt, da alle relevanten Daten beim Aufruf durch die Android-Plattform als Parameter an die *Sensor Observer* und *Broadcast Receiver* übergeben werden. Letztlich bilden diese Komponenten selbst `FeatureCollector`-Klassen, wobei die *Datensammlung* grundsätzlich ereignisbasiert durch die Android-Plattform angestoßen wird. Bei der Anpassung des Clients an eine andere Anwendungsdomäne können die entsprechend implementierten *Sensor Observer* und *Broadcast Receiver* im `MainService` des Clients flexibel für bestimmte Sensoren und Systemmeldungen registriert werden. Weitere Anpassungen des Clients erübrigen sich, da diese Komponenten nach der Registrierung durch das Android-Betriebssystem dynamisch verwaltet, ereignisbasiert aufgerufen und mit den entsprechenden Informatio-

nen versorgt werden. Damit sind *Sensor Observer* und *Broadcast Receiver* sehr lose mit den Anwendungskomponenten des Clients gekoppelt, wodurch sie mit geringem Aufwand an beliebige Anwendungsszenarien angepasst, erweitert oder auch gänzlich ausgetauscht werden können. Abbildung 5.6 bietet einen Überblick über die zentralen Komponenten zur Sammlung beliebiger Smartphone-Daten.

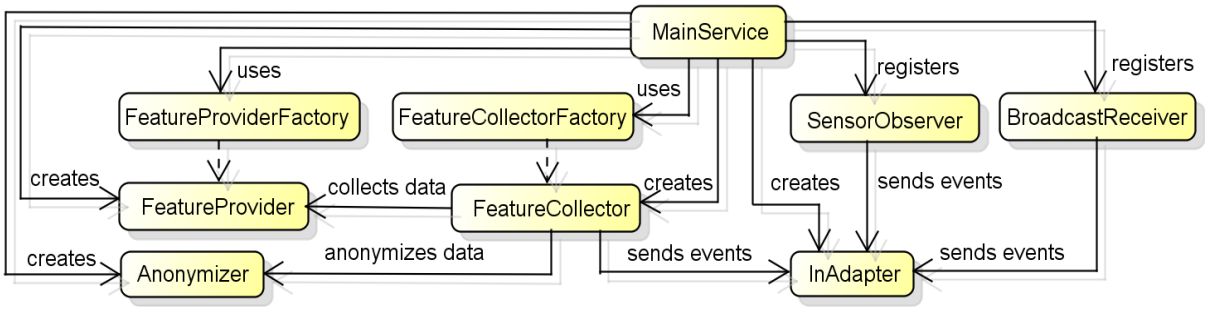

Abbildung 5.6: Zentrale Komponenten zur *Datensammlung*

In allen `FeatureCollector`-Klassen findet nach der Sammlung eine Datentransformation in entsprechende Ereignisse des in Abschnitt 5.3 vorgestellten *Ereignismodells* statt. Jede gesammelte Smartphone-Information wird demnach durch ein Ereignis repräsentiert, das anschließend mit Hilfe der mobilen CEP-Komponente weiter verarbeitet werden kann. Die Übergabe der generierten Ereignisse an die CEP-Komponente erfolgt dabei aus allen `FeatureCollector`-Klassen grundsätzlich über einen standardisierten *In*-Adapter, der die eintreffenden Ereignisse in den Ereignisstrom der mobilen CEP-Komponente einfügt. Dadurch sind die Komponenten zur *Datensammlung* über eine schmale Schnittstelle lose mit der *Datenverarbeitung* gekoppelt. Einige personenbezogene Daten werden bereits während der *Datensammlung* durch den `Anonymizer` anonymisiert, bevor sie an die CEP-Komponente übergeben werden. Für bestimmte Daten besitzen die `FeatureProvider`-Klassen mehrere unterschiedliche Zugriffsmethoden für verschiedene Android-Versionen. Insgesamt bietet diese Schicht ein Konzept, mit dem die *Datensammlung* an beliebige Daten flexibel angepasst und erweitert werden kann, wobei zusätzlich auch die Art und Weise, wie bestimmte Daten gesammelt werden, dynamisch veränderbar ist.

5.3 Datenverarbeitung

Diese Schicht umfasst den EPA als zentrale Komponente zur Verarbeitung und Verdichtung der aus den gesammelten Smartphone-Daten generierten Ereignissen (siehe Abschnitt 2.3.2). Die gesammelten Daten werden dafür bereits in den `FeatureCollector`-Klassen, *Sensor Observern* und *Broadcast Receivern* der *Datensammlung* in entsprechende Ereignisse des nachfolgend erläuterten *Ereignismodells* transformiert (siehe Abschnitt 2.3.3). Das *Ereignismodell* ist dabei so gestaltet, dass beliebige Smartphone-Daten in Form von Ereignissen repräsentiert werden können. Dadurch kann der Client mit wenig Aufwand an diverse Anwendungsdomänen angepasst und erweitert werden. Abbildung 5.7 zeigt einen Überblick über das für den eigenen Client entworfene *Ereignismodell*.

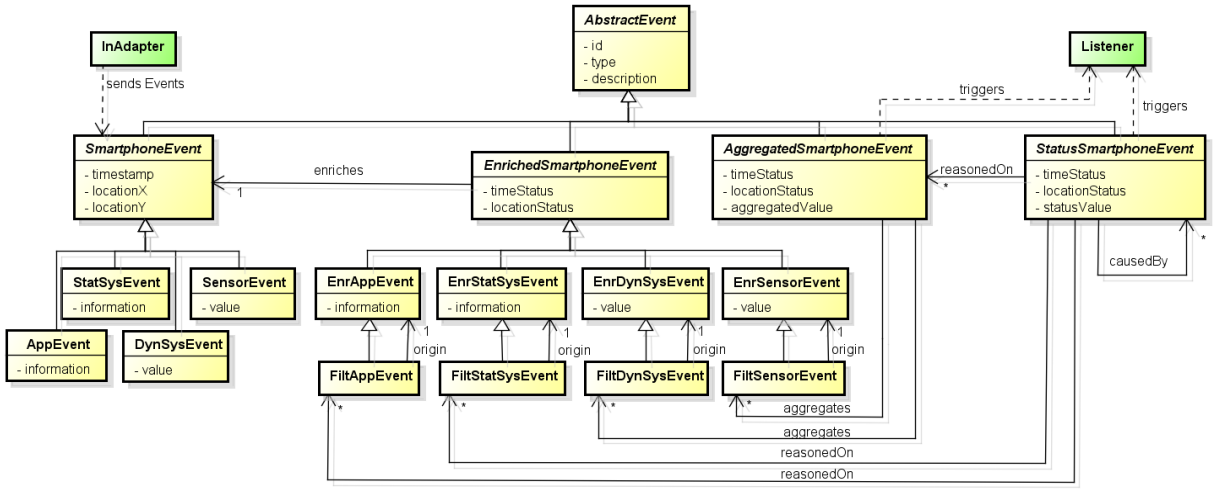

Abbildung 5.7: *Ereignismodell* des entwickelten Clients für beliebige Smartphone-Daten

Das `AbstractEvent` bildet dabei die Oberklasse für sämtliche im Kontext des entwickelten Clients verwendeten *Ereignistypen*. Dazu spezifiziert es die folgenden drei Attribute: `id`, `type` und `description`. Alle generierten und anschließend durch den *In*-Adapter übergebenen Ereignisse erben vom *Ereignistyp* `SmartphoneEvent`. Hierbei werden zusätzlich die Attribute `timestamp`, `locationX` und `locationY` spezifiziert. Durch die jeweiligen `FeatureCollector`-Klassen können vier konkrete *Ereignistypen* erzeugt werden: `StaticSystem`-, `DynamicSystem`-, `Sensor`- und `Application-Events`.

Aufgrund der in den Anforderungen dieser Arbeit beschriebenen Notwendigkeit der Reduzierung und Anonymisierung der an den MAP-Server publizierten Daten (siehe Abschnitt 4.3) verwendet der eigene Client eine direkt auf den Smartphones und Tablets befindliche, mobile CEP-Komponente. Moderne mobile Geräte und entsprechend mobile CEP-Komponenten wie *Asper* (siehe Abschnitt 2.3.6) ermöglichen dabei eine komplexe *Ereignisverarbeitung*, ohne dass die begrenzten Ressourcen der Smartphones übermäßig beansprucht werden. Durch diese Architekturvariante besteht keine Abhängigkeit zu einem externen Server, der die *Ereignisverarbeitung* mittels CEP übernimmt. Somit müssen keinerlei feingranulare Ereignisdaten über das zugrundeliegende Netzwerk an einen Server übertragen werden, wodurch sowohl das Netzwerk als auch die Netzwerkschnittstellen der mobilen Geräte geschont werden. Zusätzlich stellt diese Architekturvariante sicher, dass personenbezogene Daten direkt auf den Smartphones verarbeitet und anonymisiert werden, sodass diese kritischen Daten in keinem Verarbeitungsschritt das zugrundeliegende Gerät verlassen. Somit umfasst das entworfene Konzept des eigenen Clients aufgrund der mobilen CEP-Komponente alle drei Schichten einer EDA direkt auf den mobilen Geräten (siehe Abschnitt 2.3.1). Abbildung 5.8 zeigt dazu einen Überblick. Eine Abgrenzung zu weiteren Varianten für die Integration einer CEP-Komponente im Kontext von mobilen Geräten ist in den Abschnitten 3.3 und 7.1 zu finden.

Im ersten Verarbeitungsschritt der mobilen CEP-Komponente werden alle eintreffenden `SmartphoneEvents` mit *Kontextinformationen angereichert*. Dazu werden der

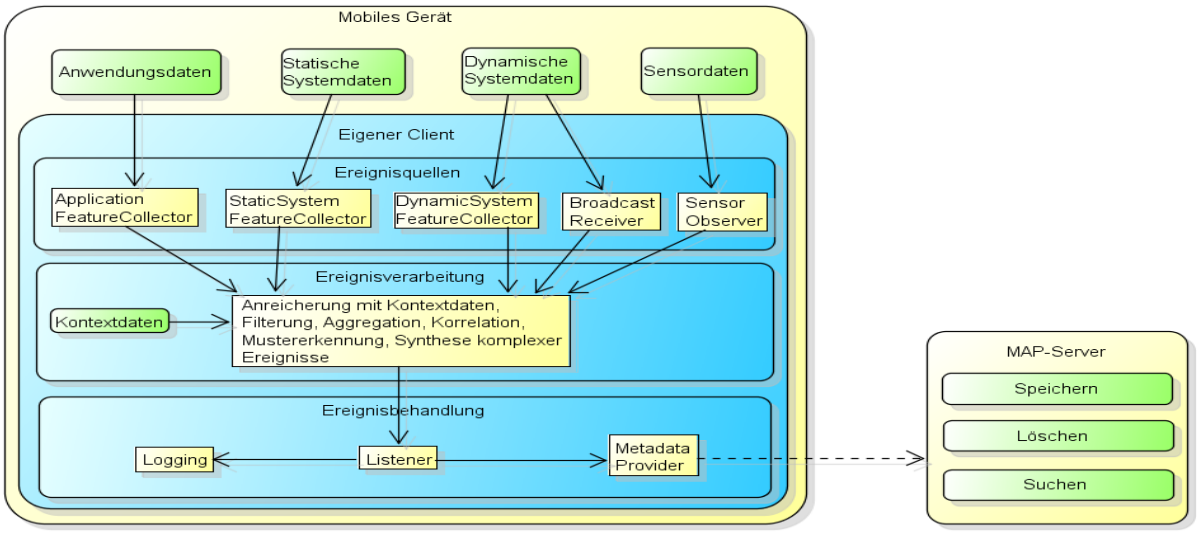

Abbildung 5.8: Logische Schichten einer EDA im Kontext des entwickelten Clients

Zeitstempel und die Positionskoordinaten durch den Aufruf externer Methoden der Klasse `ContextProvider` in die Zustandsattribute `timeStatus` und `locationStatus` abstrahiert. Es entstehen `EnrichedSmartphoneEvents`, die wiederum in den Ereignisstrom der mobilen CEP-Komponente eingefügt werden. Der nächste Verarbeitungsschritt führt eine technische und fachliche *Filterung* der `EnrichedSmartphoneEvents` durch. Einerseits überprüft die technische *Filterung* die Konsistenz der Wertebereiche bestimmter Ereignisattribute, andererseits werden durch die fachliche *Filterung* nur Ereignisse an die weiteren Verarbeitungsschritte geleitet, die sich aufgrund des `timeStatus` und `locationStatus` in einem bestimmten Verwendungskontext befinden. Diese *Filterung* sorgt demnach bereits für eine Reduktion der Ereignismenge auf ausschließlich fachlich und technisch relevante Ereignisse im Kontext der jeweiligen Anwendungsdomäne. Im nächsten Verarbeitungsschritt werden bestimmte Attribute der `FilteredDynamicSystemEvents` und `FilteredSensorEvents` durch die mobile CEP-Komponente *aggregiert*. Die *Aggregation* erfolgt dabei z. B. durch die Bestimmung von Durchschnittswerten, Summen, Minima oder Maxima der Attribute bestimmter *Ereignistypen* in vorgegebenen Zeit- und Längenfenstern (siehe Abschnitt 2.3.4). Bei der *Aggregation* werden demnach viele *gefilterte* Einzelereignisse zu einem `AggregatedSmartphoneEvent` einer höheren *Abstraktionsstufe* verdichtet, wodurch die Ereignismenge nochmals deutlich reduziert wird. Im letzten Verarbeitungsschritt werden verschiedene `AggregatedSmartphoneEvents` sowie diverse `FilteredStaticSystemEvents` und `FilteredApplicationEvents` aus unterschiedlichen Quellen *korreliert* und auf bestimmte *Ereignismuster* untersucht. Erkannte *Ereignismuster* erzeugen dabei `StatusSmartphoneEvents` als komplexe Ereignisse. Diese stellen fachlich relevante Erkenntnisse im Kontext der jeweiligen Anwendungsdomäne dar und verdichten viele Einzelereignisse zu einem neuen Ereignis mit einem entsprechend hohen Informationsgehalt und *Abstraktionsniveau*. Auch `StatusSmartphoneEvents` werden erneut in den Ereignisstrom der mobilen CEP-

Komponente eingefügt, wodurch wiederum nach weiteren *Ereignismustern* zwischen diesen komplexen Ereignissen gesucht werden kann. Der Client beinhaltet somit alle der in Abschnitt 2.3.4 aufgezeigten Verarbeitungsoperationen. Verdichtete `StatusSmartphoneEvents` stoßen schließlich Listener zur *Ereignisbehandlung* (siehe Abschnitt 2.3.5) an, die die standardisierten Schnittstellen zur *Datenübertragung* bilden. Somit wird bei der *Datenübertragung* eine deutlich reduzierte Datenmenge an den MAP-Server gesendet. `FilteredStaticSystemEvents` und `FilteredApplicationEvents` können aufgrund ihrer *String*-Attribute oft nicht sinnvoll *aggregiert* werden, sodass diese direkt *korreliert* werden. Konzeptionell kann die gesamte *Ereignisverarbeitung* des Clients durch die mobile CEP-Komponente als ein logisches EPN aus mehreren verknüpften EPAs betrachtet werden (siehe Abschnitt 2.3.2). Alle durch die *In*-Adapter eintreffenden Ereignisse werden dabei zunächst durch einen jeweiligen *Anreicherungs*- und *Filterungs*-EPA vorverarbeitet, sodass ausschließlich fachlich und technisch bedeutsame Ereignisse an den nächsten EPA weitergeleitet werden. Dieser führt dann verschiedene *Aggregierungen* der *gefilterten* und *angereicherten* Sensor- und Systemereignisse durch. *Statische* System- und Anwendungsereignisse können oft nicht sinnvoll *aggregiert* werden, sodass der *Aggregationsschritt* in diesem Fall nicht angewendet wird. Der letzte EPA dient zur *Synthese* komplexer Ereignisse durch *Korrelation* mehrerer *gefilterter* oder *aggregierter* Einzelereignisse sowie durch die Erkennung von *Ereignismustern*. Abbildung 5.9 zeigt das logische EPN des konzipierten Clients, das mehrere EPAs mit einem jeweils klar definierten Funktionsumfang sowie einer entsprechend fachlich eng zusammenhängenden und überschaubaren Menge an *Ereignisregeln* miteinander verknüpft. Auf den Smartphones und Tablets werden jedoch aus Performanzgründen alle vorgestellten Verarbeitungsschritte durch einen einzigen physikalischen EPA durchgeführt, bei dem sämtliche spezifizierten *Ereignisregeln* registriert werden.

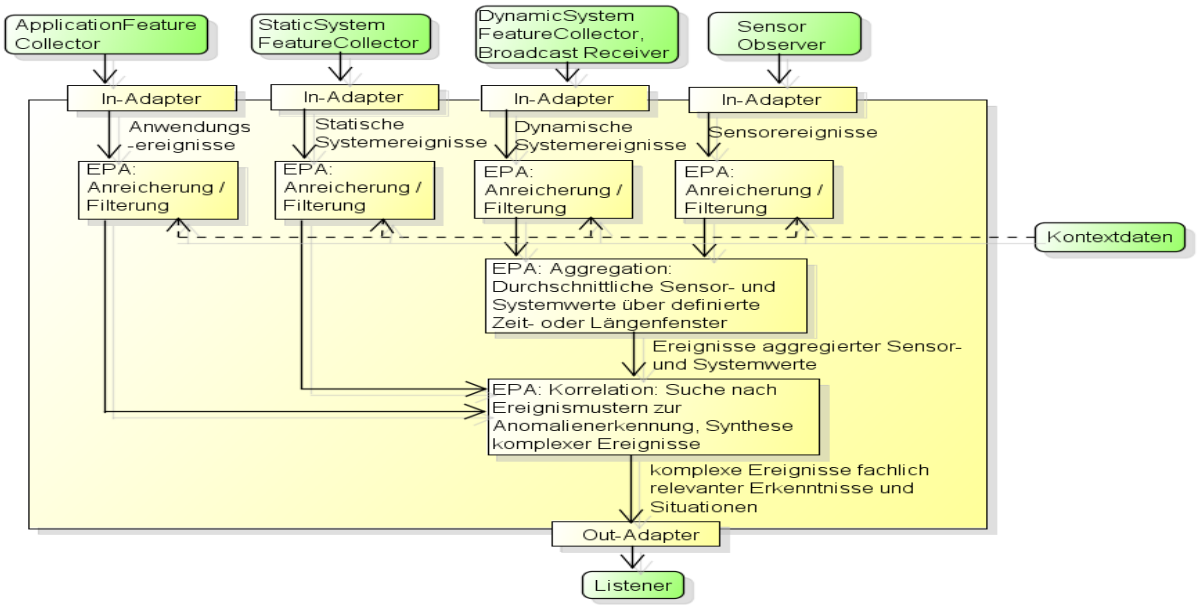

Abbildung 5.9: Logisches EPN zur *Ereignisverarbeitung* im entwickelten Client

Über den *In*-Adapter und die Listener besitzt die *Datenverarbeitung* eine lose Kopplung zur *Datensammlung* bzw. *Datenübertragung*. Dadurch kann die gesamte mobile CEP-Komponente flexibel ausgetauscht werden. Der Prototyp des Clients verwendet die mobile *Event Processing Engine Asper* (siehe Abschnitt 2.3.6). Diese ermöglicht die zeitnahe Verarbeitung und Verdichtung einer großen Menge eintreffender Ereignisse direkt auf den mobilen Geräten, ohne dass die Ressourcen der Smartphones oder des zugrundeliegenden Netzwerks zu stark belastet werden. Die *Datenverarbeitung* des Clients findet dementsprechend in nahezu Echtzeit statt, wodurch die Smartphone-Daten unmittelbar nach ihrer Sammlung verdichtet und ohne Verzögerung an den MAP-Server übertragen werden können. Die Verwendung einer mobilen CEP-Komponente, die sämtliche Verarbeitungsschritte auf den Smartphones ausführt, besitzt zudem den Vorteil, dass personenbezogene Daten direkt auf den mobilen Geräten verarbeitet und anonymisiert werden können und somit das zugrundeliegende Gerät zu keinem Zeitpunkt verlassen (siehe Abschnitt 3.2.1). Im *Anreicherungsschritt* wird die `ContextProvider`-Klasse aufgerufen, um den Zeitstempel sowie die Positionskoordinaten in Form von bestimmten Zuständen als *Kontextinformationen* zu abstrahieren bzw. zu anonymisieren. Die deklarativen *Ereignisregeln* werden in mehreren `Statement`-Klassen als statische *Strings* verwaltet.

Nachfolgend wird der Aufbau einiger exemplarischer *Ereignisregeln* vorgestellt, die in den verschiedenen Verarbeitungsschritten des in Abbildung 5.9 aufgezeigten, logischen EPNs des eigenen Clients zum Einsatz kommen. Diese *Ereignisregeln* zeigen dabei am Beispiel der CPU-Auslastung als *dynamische* Systeminformation, wie aus vielen `DynamicSystemEvents` durch *Anreicherung*, *Filterung*, *Aggregation*, *Korrelation* und *Mustererkennung* verdichtete, komplexe Ereignisse entstehen, die einen signifikanten Anstieg der CPU-Auslastung beschreiben. Letztlich kann dieses Beispiel zur *Ereignisverarbeitung* auch auf beliebig andere Anwendungsdomänen angewendet werden. Zunächst folgt in Listing 5.1 eine *Ereignisregel* zur *Kontextanreicherung* von `DynamicSystemEvents`. Dabei werden der Zeitstempel und die Positionsdaten der eintreffenden Ereignisse durch die Methoden des `ContextProviders` in Zeit- und Positionszustände abstrahiert und anonymisiert. Schließlich werden `EnrichedDynamicSystemEvents` erzeugt.

```
1 insert into EnrichedDynamicSystemEvent(id, timeStat, locStat, type, descr, sysValue)
2 select id, CP.timeStat(t) as timeStat, CP.locStat(x,y) as locStat, type, descr, sysValue
3 from DynamicSystemEvent
```

Listing 5.1: *Ereignisregel* zur *Kontextdatenanreicherung* von `DynamicSystemEvents`

Die *Ereignisregel* aus Listing 5.2 *filtert* anschließend `CpuLoadEvents` aus den `EnrichedDynamicSystemEvents`. Dabei erfolgt sowohl eine fachliche *Filterung* anhand der `Id` sowie der Zeit- und Positionszustände als auch eine technische *Filterung* zur Konsistenzprüfung des Wertebereichs des `sysValue`-Attributs. Schließlich werden entsprechende `CpuLoadEvents` erzeugt.

```
1 insert into CpuLoadEvent(id, timeStat, locStat, type, descr, cpuLoad)
2 select id, timeStat, locStat, type, descr, sysValue as cpuLoad
3 from EnrichedDynamicSystemEvent(id = 'cpu', timeStat = 'working_time',
4                 locStatus = 'restricted_area', sysValue between 0 and 100)
```

Listing 5.2: *Ereignisregel* zur *Filterung* von `DynamicSystemEvents`

Listing 5.3 zeigt eine *Ereignisregel* zur *Aggregation* der `CpuLoadEvents`. Dabei wird die durchschnittliche CPU-Auslastung aller eintreffenden `CpuLoadEvents` über einen Zeitraum von 30 Sekunden berechnet. Da es sich um ein *Batch*-Zeitfenster handelt, erfolgt diese Berechnung jeweils erst nach Ablauf dieses Zeitfensters. Dabei entstehen entsprechende `CpuLoadAverageEvents`.

```
1 insert into CpuLoadAverageEvent(id, timeStat, locStat, type, descr, cpuLoadAverage)
2 select id, timeStat, locStat, type, descr, avg(cpuLoad) as cpuLoadAverage
3 from CpuLoadEvent.win:time_batch(30 sec)
4 group by id, timeStat, locStat, type, descr
```

Listing 5.3: *Ereignisregel* zur *Aggregation* von `CpuLoadEvents`

Die *Ereignisregel* aus Listing 5.4 *korreliert* mehrere `CpuLoadAverageEvents` und sucht dabei nach *Mustern*, die einen signifikanten Anstieg der CPU-Auslastung beschreiben. Es entstehen komplexe `CpuLoadStatusEvents`. Das Auftreten eines solchen komplexen Ereignisses stößt letztlich einen entsprechenden Listener zur *Ereignisbehandlung* an.

```
1 insert into CpuLoadStatusEvent(id, timeStat, locStat, type, descr, cpuLoadStatus)
2 select id, e2.timeStat, e2.locStat, type, descr, 'CpuLoad_high'
3 from pattern[every e1=CpuLoadAverageEvent -> e2=CpuLoadAverageEvent(
4                 cpuLoadAverage > e1.cpuLoadAverage+25) where timer:within(2 min)]
```

Listing 5.4: *Ereignisregel* zur *Korrelation* von `CpuLoadAverageEvents`

Die *Ereignisregeln* können dabei auf die Verarbeitung beliebiger Smartphone-Daten im Kontext diverser Anwendungsszenarien angepasst werden. Die entsprechend des Anwendungsfalls spezifizierten *Ereignisregeln* werden bei der Initialisierung des EPAs im `MainService` registriert. Je nach Anwendungsdomäne können schließlich beliebige Listener mit den einzelnen *Ereignisregeln* im `MainService` verknüpft werden. Die Anpassungen der ereignisbasierten *Datenverarbeitung* des eigenen Clients an eine neue Anwendungsdomäne betreffen demnach ausschließlich das *Regelwerk*, die Listener sowie die Initialisierung des EPAs im `MainService`. Abbildung 5.10 zeigt die zentralen Komponenten der ereignisbasierten *Datenverarbeitung* durch CEP.

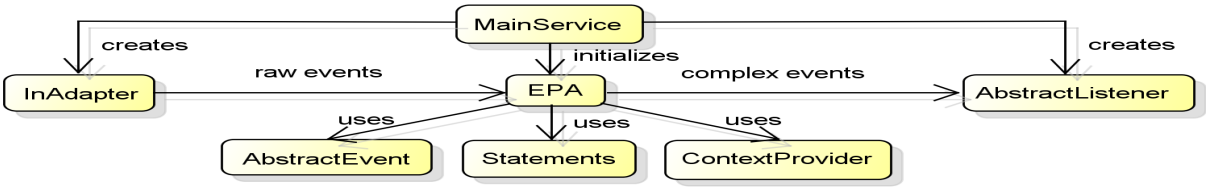

Abbildung 5.10: Zentrale Komponenten zur *Datenverarbeitung*

5.4 Datenübertragung

Diese Schicht umfasst die Komponenten zur Transformation der in der *Datenverarbeitung* verdichteten Informationen in Form von *aggregierten* oder komplexen Ereignissen in entsprechendes IF-MAP Vokabular. Zudem beinhaltet die *Datenübertragung* auch die Komponenten zur Kommunikation mit dem MAP-Server. Zur Unterstützung diverser Anwendungsszenarien, bei denen eine Vielzahl verschiedener Daten gesammelt und verarbeitet werden kann, spezifiziert der eigene Client ein domänenunabhängiges Datenmodell zur

Repräsentation beliebiger Smartphone-Daten im IF-MAP Umfeld, das das von der TCG standardisierte IF-MAP Datenmodell erweitert (siehe Abschnitt 2.4.4). Dieses erweiterte Datenmodell basiert auf dem in Abschnitt 3.1.2 vorgestellten *Feature-Metadatenmodell*, das im Kontext des *CADS*-Systems eingesetzt wird. Weiterhin bietet der Client auch die Möglichkeit, den gesamten Mechanismus zur Transformation der verdichteten Ereignisse in IF-MAP Vokabular über die *Factory*-Klasse `MetadataProviderFactory` beliebig auszutauschen. Je nach Anwendungsszenario kann demnach nicht nur das standardmäßig eingesetzte *Feature-Metadatenmodell* auf beliebige Smartphone-Daten angepasst und erweitert werden, sondern auch das gesamte IF-MAP Datenmodell gegen ein anderes ausgetauscht werden. Dazu gibt das `MetadataProvider`-Interface die Struktur für die jeweils konkret implementierten `MetadataProvider`-Klassen zur Transformation der Ereignisse in IF-MAP Vokabular vor. Deren Instanziierung erfolgt über die `MetadataProviderFactory`, wodurch die übrigen Komponenten des Clients von den konkreten Implementierungen der jeweiligen `MetadataProvider`-Klassen entkoppelt werden. Bei der Anpassung des entwickelten Clients an eine andere Anwendungsdomäne können die entsprechend implementierten `MetadataProvider`-Klassen über dieses *Factory-Pattern* erzeugt werden, sodass keine weiteren Komponenten des Clients verändert werden müssen. Abbildung 5.11 zeigt entsprechend die Umsetzung dieses *Factory-Patterns* bei der *Datenübertragung*.

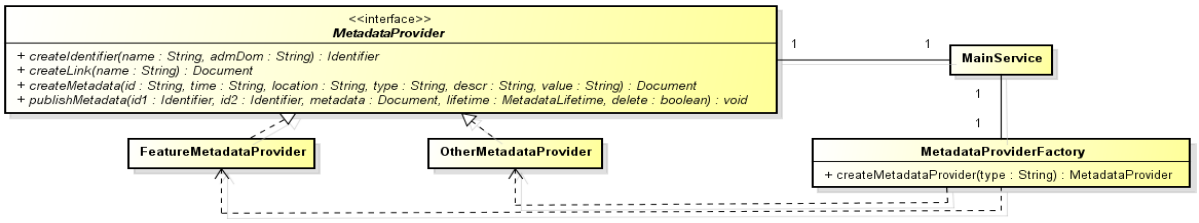

Abbildung 5.11: Umsetzung eines *Factory-Patterns* bei der *Datenübertragung*

Der Prototyp des Clients nutzt den `FeatureMetadataProvider`, der entsprechend die Methoden des `MetadataProvider`-Interfaces implementiert und über die `MetadataProviderFactory` im `MainService` des Clients erzeugt wird. Diese Klasse wird verwendet, um die in der *Datenverarbeitung* verdichteten Ereignisse in das auf beliebige Smartphone-Daten anpass- und erweiterbare *Feature-Metadatenmodell* zu transformieren und die so generierten *Metadaten* anschließend an den MAP-Server zu übertragen. Andere `MetadataProvider`-Klassen können mit Hilfe dieser Methoden eigene IF-MAP Datenmodelle zur Repräsentation der gesammelten Smartphone-Daten entsprechend einer jeweiligen Anwendungsdomäne umsetzen. Grundsätzlich dient der entwickelte Client ausschließlich zum Veröffentlichen von Informationen (siehe Abschnitt 2.4.3). Es sind jedoch auch Szenarien denkbar, bei denen der eigene Client *request-for-investigation-Metadaten* vom MAP-Server abonniert, sodass andere IF-MAP Komponenten die Sammlung, Verarbeitung und Übertragung bestimmter Smartphone-Daten anstoßen könnten.

Das im Client verwendete *Feature-Metadatenmodell* umfasst zwei Kernkomponenten. *Features* beschreiben Eigenschaften der Smartphones und Tablets als Schlüssel-Wert-Paare in Form von *Metadaten*. Dazu enthält ein *Feature* eine ID, einen Datentyp,

einen Wert, eine Beschreibung und *Kontextinformationen*. *Feature-Metadaten* werden mit *Category-Identifiern* verknüpft, die zur semantischen Strukturierung der *Features* dienen. Ein *Category-Identifier* umfasst demnach fachlich eng in Beziehung stehende *Feature-Metadaten*. *Categories* können aber auch untereinander organisiert werden, wodurch eine Baumstruktur zur Verwaltung der *Features* aufgebaut werden kann. Je zwei *Category-Identifier* werden dabei durch einen *Link* mittels *subcategory-of-Metadaten* verbunden. Diese Baumstruktur wird schließlich mit dem das jeweilige mobile Gerät repräsentierenden *device-Identifier* des von der TCG standardisierten, ursprünglichen Datenmodells verknüpft. Das abstrakte *Feature-Metadatenmodell* ermöglicht die Transformation aller gesammelten und verdichteten Daten in IF-MAP Vokabular, indem es um neue *Features* erweitert werden kann (siehe Abschnitt 3.1.2). Abbildung 5.12 zeigt dazu die Baumstruktur des *Feature-Metadatenmodells*. Wie oben beschrieben wurde, unterstützt der eigene Client aber auch den Austausch des gesamten Mechanismus zur Transformation der verdichteten Ereignisse in IF-MAP Vokabular. Insgesamt kann der Client somit flexibel an beliebige, bereits bestehende IF-MAP Umgebungen angepasst werden. Neben sicherheitsrelevanten Daten im Kontext des *CADS*-Systems können durch das *Feature-Metadatenmodell* auch diverse andere Smartphone-Daten in IF-MAP Vokabular transformiert und an den MAP-Server gesendet werden. IF-MAP bietet dabei standardisierte Kommunikationsoperationen (siehe Abschnitt 2.4.5), wodurch der Client flexibel in vorhandene Infrastrukturen integriert werden kann. Durch die zeitnahe *Datenübertragung* können andere Netzwerkkomponenten in nahezu Echtzeit auf die bereitgestellten Daten reagieren.

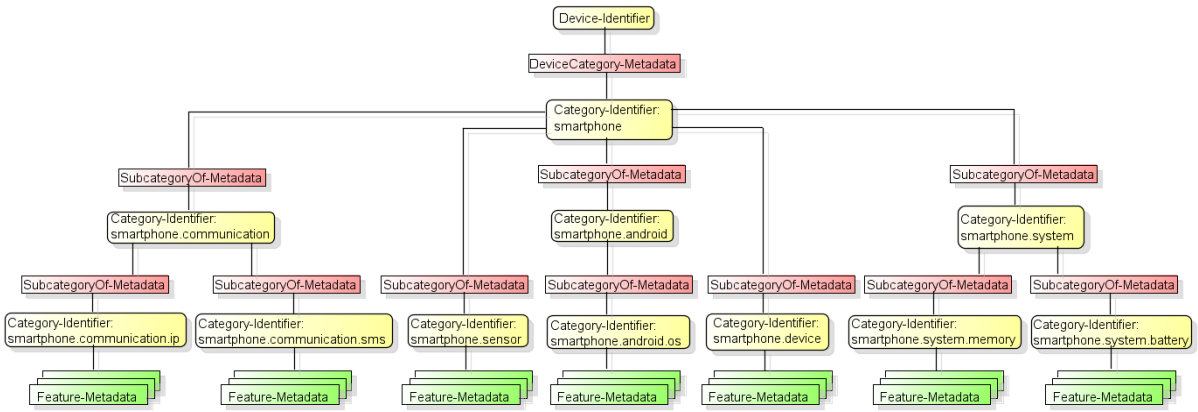

Abbildung 5.12: *Feature-Metadatenmodell* zur Verwaltung von Smartphone-Daten

Mit Hilfe des `FeatureMetadataProviders` können beliebige *Category-Identifier* zur Strukturierung der *Feature-Metadaten* erzeugt werden. Dazu werden über die *ifmapj*-Bibliothek *identity-Identifier* vom Typ *other* generiert. Im Konstruktor des `FeatureMetadataProviders` wird der Aufbau der Baumstruktur aus *Category-Identifiern* angestoßen, die mit dem entsprechenden *device-Identifier* des ursprünglichen IF-MAP Datenmodells verknüpft wird und somit die Basis für alle weiteren publizierten *Features* darstellt. Die erzeugten *Category-Identifier* sind über den `FeatureMetadataProvider` auch für andere Komponenten des Clients verfügbar. Listener bilden die Schnittstellen zur

Datenverarbeitung, wobei deren `update()`-Methode durch die mobile CEP-Komponente aufgerufen wird, sobald der Bedingungsteil einer verknüpften *Ereignisregel* erfüllt ist. Das auslösende Ereignis wird dabei als Parameter an die `update()`-Methode übergeben. Abbildung 5.13 zeigt die Listener-Hierarchie des entwickelten Clients. Diese gliedert sich an die Ereignishierarchie des in Abschnitt 5.3 vorgestellten *Ereignismodells* an, sodass die jeweiligen *Ereignistypen* in spezifischen, fachlichen Listenern behandelt werden können. Der `AbstractListener` bildet, ähnlich zum `AbstractEvent`, einen abstrakten Obertyp für alle Listener und dient zum Auslesen allgemeiner Ereignisattribute. Das Auslesen spezieller fachlicher Ereignisattribute sowie die Transformation der Ereignisse in IF-MAP Vokabular finden schließlich in den spezifischen Listenern statt.

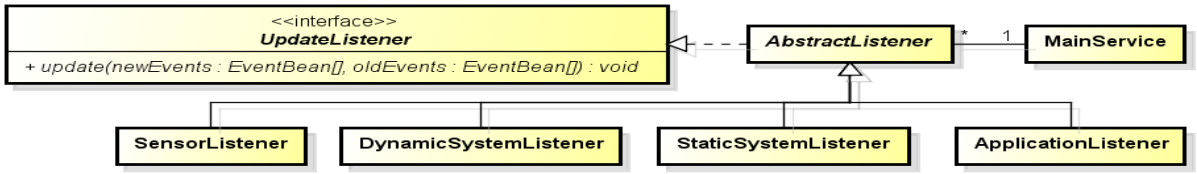

Abbildung 5.13: Listener-Hierarchie des entwickelten Clients zur *Ereignisbehandlung*

Die Listener nutzen die über das *Factory-Pattern* erzeugte Instanz des `MetadataProviders`, um die Daten der übergebenen Ereignisse in IF-MAP Vokabular zu transformieren. Die generierten *Metadaten* werden anschließend ebenfalls mit Hilfe des `MetadataProviders` an den MAP-Server übertragen. Dabei wird ein `PublishUpdate`-Objekt der *ifmapj*-Bibliothek erzeugt, an das zwei *Identifier*, das *Metadatum* und die Lebensdauer übergeben werden können. Schließlich wird die *Datenübertragung* durch den *AsyncTask* `PublishTask` ausgeführt, indem das beim Start erzeugte SSRC-Objekt verwendet wird.

Der Verbindungsaufbau und -abbau zum MAP-Server wird beim Start bzw. Ende des `MainService` angestoßen. Dazu werden die *AsyncTasks* `ConnectTask` und `DisconnectTask` verwendet. Beim Verbindungsaufbau wird ein `BasicAuthConfig`-Objekt der *ifmapj*-Bibliothek mit den Verbindungsinformationen aus der `IfmapConfig`-Klasse initialisiert. Damit werden schließlich der SSRC und ARC zum MAP-Server aufgebaut (siehe Abschnitt 2.4.5). Im `DisconnectTask` wird die Verbindung über die *ifmapj*-Bibliothek beendet. Die gesamte Kommunikation mit dem MAP-Server erfolgt somit durch *AsyncTasks*. Diese kapseln potentiell langlaufende Netzwerkoperationen in eigene *Threads*, sodass der *Main-Thread* des Clients nicht blockiert wird. Sowohl für das Verbindungsmanagement als auch für den Datenaustausch mit dem MAP-Server werden Methoden der *ifmapj*-Bibliothek verwendet, die die technischen Verbindungsdetails abstrahieren (siehe Abschnitt 2.4.6). Abbildung 5.14 zeigt die Komponenten der *Datenübertragung*.

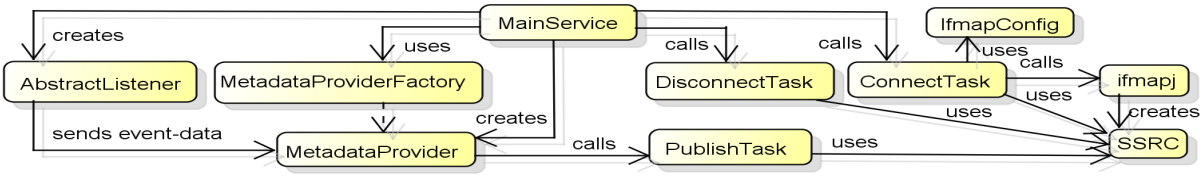

Abbildung 5.14: Zentrale Komponenten zur *Datenübertragung*

6 Entwicklung des Prototyps

Aufbauend auf die vorgestellten Konzepte, Strategien und Modelle zur Umsetzung eines domänenunabhängigen IF-MAP Clients in Form einer leichtgewichtigen Android-*App*, der beliebige Smartphone-Daten sammelt, diese über eine entsprechend anpass- oder komplett austauschbare mobile CEP-Komponente verarbeitet und letztlich durch ein flexibel erweiter- oder ebenfalls komplett austauschbaren Mechanismus in IF-MAP Vokabular transformiert sowie abschließend an einen MAP-Server überträgt, erfolgt in diesem Kapitel die Entwicklung eines Prototyps dieses Clients. Dieser implementiert dabei die Funktionalität des konkreten, in Abschnitt 4.2 aufgezeigten Anwendungsszenarios zur Sammlung, Verdichtung und Übertragung von sicherheitsrelevanten Smartphone-Daten im Kontext des *CADS*-Systems als ein Beispiel zur Demonstration der grundsätzlichen Arbeitsweise des entworfenen Konzepts für den eigenen Client. In den folgenden Abschnitten werden dazu exemplarisch einige entscheidende Ausschnitte der implementierten Komponenten zum zentralen Aufbau des Prototyps (siehe Abschnitt 6.1), zur *Datensammlung* (siehe Abschnitt 6.2), zur *Datenverarbeitung* (siehe Abschnitt 6.3) und zur *Datenübertragung* (siehe Abschnitt 6.4) vorgestellt sowie erläutert. Für eine detailierte Beschreibung der Komponenten sei hier zudem ausdrücklich auf die Quellcode-Dokumentation des Prototyps verwiesen. Abschließend werden in Abschnitt 6.5 Besonderheiten beschrieben, die bei der Implementierung aufgetreten sind.

6.1 Hauptkomponenten des Prototyps

Die `MainActivity` und der `MainService` bilden die beiden zentralen Hauptkomponenten des Prototyps. Diese befinden sich neben dem `MainTimerTask` im *application*-Package des Prototyps und werden in der Manifest-Datei als Einstiegspunkt in die Client-Anwendung bzw. als lokaler *Service* spezifiziert. Zudem werden dabei die *minSdk*- und *targetSdk*-Version auf API-Level 8 (2.2, Froyo) bzw. 21 (4.4, KitKat) gesetzt sowie die *Permissions* zum Zugriff auf das Internet und auf die Geräteposition spezifiziert. Dementsprechend unterstützt der Prototyp eine Vielzahl verschiedener Android-Betriebssysteme und ist somit auf dem Großteil der bisher verkauften mobilen Geräte einsatzfähig. Die Methoden `onStart()` und `onStop()` der `MainActivity` werden durch einen Klick auf den jeweiligen *Button* angestoßen und starten bzw. stoppen den `MainService`. Weiterhin realisiert die `MainActivity` *Broadcast Receiver*, um den Einsatzstatus sowie Logging-Einträge des `MainService` in Form von selbstdefinierten *Intents* zu empfangen. Der `MainService` realisiert dazu die entsprechenden Methoden `sendRunningStatus()` und `sendLogEntry()`. Die Benutzeroberfläche ist über XML-Dokumente im *res*-Verzeichnis des Prototyps strikt getrennt vom *Java*-Quellcode definiert, wodurch das Design oder auch die Sprache mit wenig Aufwand angepasst wer-

den können. Alle Komponenten der Benutzeroberfläche sind somit als XML-Elemente spezifiziert, sodass sich Veränderungen an der Benutzeroberfläche nicht auf den *Java*-Quellcode auswirken. Das *libs*-Verzeichnis umfasst die beiden Bibliotheken *ifmapj* zur IF-MAP Kommunikation und *Asper* als mobile CEP-Komponente. Basierend auf den Konzepten zum Aufbau des entworfenen Clients aus Abschnitt 5.1 zeigt Abbildung 6.1 einen Ausschnitt des Klassenmodells des Prototyps mit seinen Hauptkomponenten.

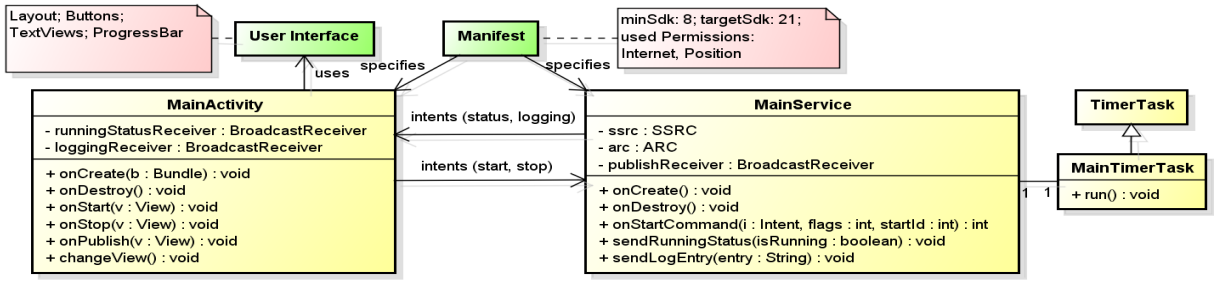

Abbildung 6.1: Klassenmodell: Hauptkomponenten des Prototyps

Nachdem in der `onCreate()`-Methode des `MainService` die wesentlichen Komponenten zur *Datensammlung*, *-verarbeitung* und *-übertragung* initialisiert wurden, wird in der `onStartCommand()`-Methode durch den Aufruf von `startForeground()` der `MainService` mit der Priorität einer im Vordergrund befindlichen Anwendung ausgeführt. Dadurch kann der Client im Hintergrund betrieben werden, ohne dass er vom Android-System frühzeitig beendet wird. *Sensor Observer* und *Broadcast Receiver* zur ereignisbasierten *Datensammlung* werden über die Methoden `registerSensorObserver()` und `registerBroadcastReceiver()` registriert. Zudem wird die *Datensammlung* über den `MainTimerTask` und über den `StaticSystemFeatureCollector` angestoßen (siehe Listing 6.1). In der `onDestroy()`-Methode des `MainService` werden die Verbindung zum MAP-Server abgebaut, die ereignisbasierte *Datenverarbeitung* des EPAs beendet und die *Sensor Observer* sowie die *Broadcast Receiver* gestoppt.

```
1  public int onStartCommand(final Intent intent, final int flags, final int startId) {
2          startForeground(1, getNotification());
3          registerSensorObserver();
4          registerBroadcastReceiver();
5          timer.schedule(mainTimerTask, 0, 5000);
6          staticSystemFeatureCollector.collectStaticSystemFeatures();
7          return super.onStartCommand(intent, flags, startId);
8  }
9  private Notification getNotification() {
10         NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
11         builder.setSmallIcon(R.drawable.ic_launcher).setContentTitle("MAP-Client");
12         Notification notification = builder.build();
13         return notification;
14 }
```

Listing 6.1: `onStartCommand()`-Methode des `MainService`

6.2 Komponenten zur Datensammlung

`FeatureProvider`-Klassen bilden die Schnittstellen zur Android-Plattform, um *statische* oder *dynamische* System- und Anwendungsinformationen über Dienste des *Application*

Frameworks zu beziehen. Aus Gründen der Anpassbarkeit des eigenen Clients an beliebige Anwendungsdomänen im Kontext von Smartphone-Daten, werden die konkreten `FeatureProvider`-Instanzen über die `FeatureProviderFactory` im `MainService` geladen, wodurch die übrigen Komponenten des Prototyps von deren jeweiliger Implementierung entkoppelt sind. Sowohl die `FeatureProviderFactory` als auch die `FeatureProvider`-Interfaces und konkreten -Klassen sind im *featureprovider*-Package des Prototyps zu finden. Die eigentliche Sammlung der *statischen* oder *dynamischen* System- und Anwendungsinformationen erfolgt in `FeatureCollector`-Klassen (siehe Listing 6.2). Dabei werden die jeweiligen Daten über die `FeatureProvider`-Klassen bezogen und mit *Kontextinformationen* in Form von der Systemzeit und der Geräteposition erweitert. Schließlich erfolgt in den `FeatureCollector`-Klassen die Transformation der gesammelten Daten in Ereignisse des in Abschnitt 5.3 vorgestellten, beliebig anpass- und erweiterbaren *Ereignismodells*. Die entsprechenden *Ereignistypen* liegen im Prototyp in Form von *Java*-Beans im *event*-Package vor (siehe Abschnitt 6.3). Die aus den gesammelten Daten generierten Ereignisse werden letztlich durch den standardisierten `InAdapter` in den Ereignisstrom der CEP-Komponente eingefügt (siehe Listing 6.3).

```
1  private void sendCpuLoad() {
2          inAdapter.sendEvent(new DynamicSystemEvent(
3                  "cpu",
4                  System.currentTimeMillis(),
5                  main.getLocationX(),
6                  main.getLocationY(),
7                  IfmapConfig.QUANTITATIVE,
8                  "cpuLoad-description",
9                  dynamicSystemFeatureProvider.getCpuLoad()));
10 }
```

Listing 6.2: Sammlung der CPU-Auslastung und Transformation in ein Ereignis

```
1  public class InAdapter {
2          private EPServiceProvider epa;
3          private EPRuntime epRuntime;
4          public InAdapter() {
5                  epa = EPServiceProviderManager.getProvider(MainService.EPA_URI);
6                  epRuntime = epa.getEPRuntime();
7          }
8          public void sendEvent(AbstractEvent event) {
9                  epRuntime.sendEvent(event);
10         }
11 }
```

Listing 6.3: Standardisierter `InAdapter` als schmale Schnittstelle zur *Datenverarbeitung*

Konkrete `FeatureCollector`-Klassen werden ebenfalls über ein *Factory-Pattern* durch die `FeatureCollectorFactory` im `MainService` erzeugt. Dadurch kann der gesamte Mechanismus zur Sammlung von Smartphone-Daten an beliebige Anwendungsdomänen angepasst, erweitert oder ausgetauscht werden, ohne dass andere Komponenten des Prototyps verändert werden müssen. Sowohl die `FeatureCollectorFactory` als auch die `FeatureCollector`-Interfaces und konkreten -Klassen sind im *featurecollector*-Package des Prototyps zu finden. Die Sammlung *dynamischer* Systeminformationen durch den `DynamicSystemFeatureCollector` wird in periodischen Zeitintervallen durch den *TimerTask* `MainTimerTask` angestoßen. Die Sammlung *statischer* System- und Anwen-

dungsinformationen erfolgt hingegen durch den manuellen Aufruf der jeweiligen `FeatureCollector`-Klasse. Für die ereignisbasierte Sammlung von Sensor- oder Systeminformationen werden *Sensor Observer* und *Broadcast Receiver* verwendet. Aufbauend auf den Konzepten zur *Datensammlung* des entworfenen Clients aus Abschnitt 5.2 zeigt Abbildung 6.2 einen Ausschnitt des Klassenmodells des Prototyps mit seinen zentralen Komponenten zur *Datensammlung*.

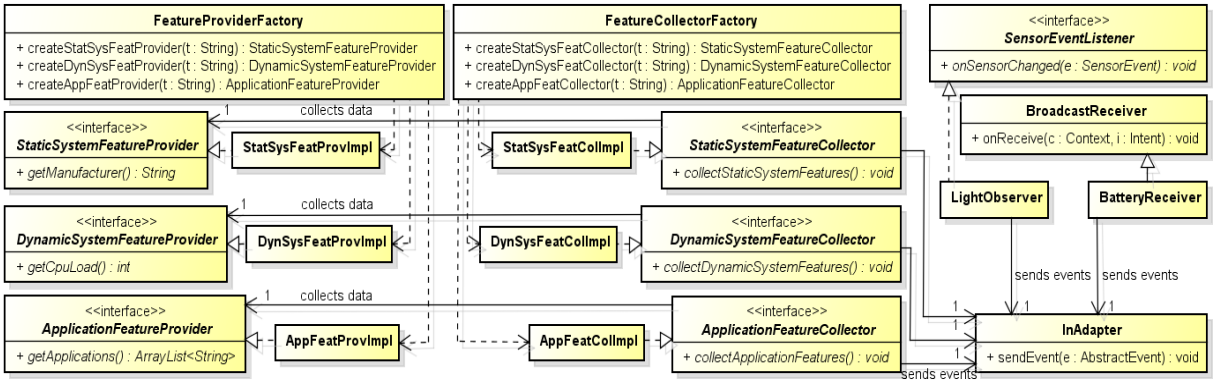

Abbildung 6.2: Klassenmodell: Zentrale Komponenten der *Datensammlungsschicht*

Entsprechend der Anwendungsdomäne können im `MainService` des Prototyps beliebige *Sensor Observer* und *Broadcast Receiver* des *observer*-Packages für die ereignisbasierte *Datensammlung* registriert werden. Nach der Registrierung werden diese Komponenten schließlich durch die Android-Plattform verwaltet und angestoßen. Demnach sind keine weiteren Anpassungen an den übrigen Komponenten des Prototyps erforderlich. Personenbezogene Daten können bereits bei der Transformation in entsprechende Ereignisse in den `FeatureCollector`-Klassen durch die `anonymize()`-Methode des `Anonymizers` anonymisiert werden. Alternativ kann die Anonymisierung auch im *Anreicherungsschritt* der mobilen CEP-Komponente umgesetzt werden. Der Prototyp stellt dazu beide Lösungsvarianten vor. So wird beispielsweise die IMEI bereits bei der *Datensammlung* anonymisiert, während die Geräteposition im *Anreicherungsschritt* durch die CEP-Komponente in abstrakte Zustände überführt wird. Der `Anonymizer` befindet sich im *util*-Package des Prototyps.

6.3 Komponenten zur Datenverarbeitung

Die zentrale Komponente der *Datenverarbeitung* des Prototyps bildet die mobile *Event Processing Engine Asper*. Diese wird beim Start in der `createEpa()`-Methode des `MainService` initialisiert (siehe Listing 6.4). Dabei wird zunächst ein `Configuration`-Objekt angelegt, bei dem sämtliche in dem *event*-Package des Prototyps in Form von *Java*-Beans implementierten *Ereignistypen* registriert werden. Das `AbstractEvent` bildet die abstrakte Oberklasse für alle Ereignisse, die durch die mobile CEP-Komponente verarbeitet werden sollen. Das davon erbende `SmartphoneEvent` stellt wiederum die Oberklasse für sämtliche in der *Datensammlung* generierten Ereignisse dar: `SensorEvent`, `StaticSystemEvent`, `DynamicSystemEvent` und `ApplicationEvent`. Mit Hilfe dieses

`Configuration`-Objekts wird schließlich eine Instanz der *Asper*-Engine über den `EPServiceProviderManager` in Form eines EPAs erzeugt. Zu diesem wird ein `EPAdministrator`-Objekt instanziiert, mit dessen Hilfe die jeweiligen *Ereignisregeln* inkl. ihrer Listener beim EPA über die Methoden `createEPL()` bzw. `addListener()` registriert werden.

```
1  private void createEpa() {
2          Configuration epaConfig = new Configuration();
3          epaConfig.addEventType("AbstractEvent", AbstractEvent.class.getName());
4          epaConfig.addEventType("SmartphoneEvent", SmartphoneEvent.class.getName());
5          epaConfig.addEventType("DynamicSystemEvent", DynamicSystemEvent.class.getName());
6
7          epa = EPServiceProviderManager.getProvider(EPA_URI, epaConfig);
8          epaAdmin = epa.getEPAdministrator();
9
10         epaAdmin.createEPL(DynSysStatements.dynamicSystemEnrichStatement);
11         epaAdmin.createEPL(DynSysStatements.cpuLoadFilterStatement);
12         epaAdmin.createEPL(DynSysStatements.cpuLoadAverageStatement);
13         epaAdmin.createEPL(DynSysStatements.cpuLoadHighStatement).
14                                                 addListener(dynamicSystemListener);
15 }
```

Listing 6.4: Initialisierung der *Asper*-Instanz im `MainService`

Die bei der *Datensammlung* erzeugten Ereignisse werden über den standardisierten `InAdapter` als schmale Schnittstelle zur *Datenverarbeitung* in den Ereignisstrom des EPAs eingefügt (siehe Listing 6.3). Dazu wird der bereits im `MainService` erzeugte EPA über den `EPServiceProviderManager` geladen. Anschließend wird ein `EPRuntime`-Objekt für den EPA angelegt, mit dessen Hilfe die generierten Ereignisse in den Ereignisstrom des EPAs über die `sendEvent()`-Methode eingefügt werden.

Sämtliche *Asper-Ereignisregeln* werden in Form von statischen *Strings* in den entsprechenden `Statement`-Klassen spezifiziert. Diese besitzen eine SQL-ähnliche Syntax, die jedoch um typische CEP-Befehle, wie etwa *Sliding Windows* oder *EventPattern*-Konstrukte, erweitert wird. Durch die ereignisbasierte *Datenverarbeitung* des EPAs werden die eintreffenden Ereignisse verdichtet, indem ihre Anzahl durch *Filterung*, *Anreicherung*, *Aggregation* und *Korrelation* reduziert wird, während der Informationsgehalt der einzelnen Ereignisse steigt. Aufbauend auf den Konzepten zur *Datenverarbeitung* des entworfenen Clients aus Abschnitt 5.3 zeigt Abbildung 6.3 einen Ausschnitt des Klassenmodells des Prototyps mit seinen zentralen Komponenten zur *Datenverarbeitung*.

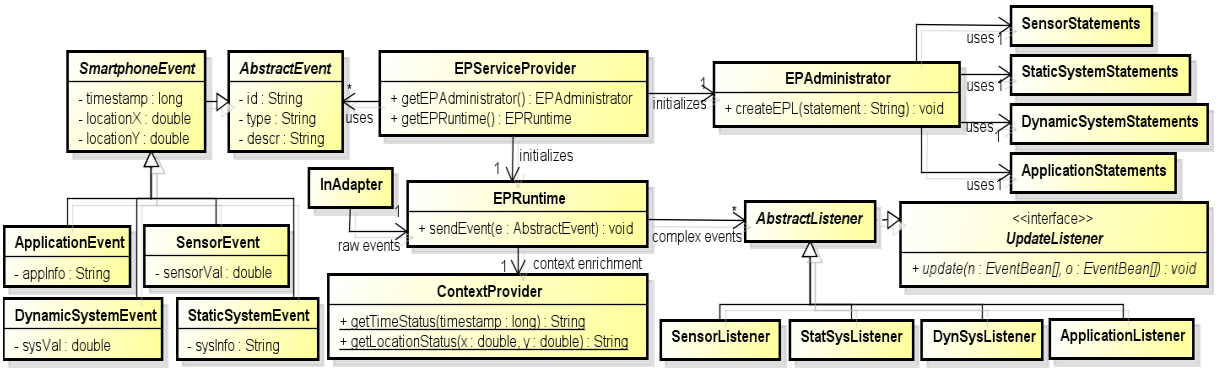

Abbildung 6.3: Klassenmodell: Zentrale Komponenten der *Datenverarbeitungsschicht*

Im Folgenden werden anhand der CPU-Auslastung als *dynamische* Systeminformation die Schritte zur Datenverdichtung des Prototyps erläutert, die letztlich in Form von *Ereignisregeln* beim EPA registriert sind. Die Verarbeitungsschritte entsprechen dabei dem in Abbildung 5.9 vorgestellten Konzept eines logischen EPNs. Um die begrenzten Ressourcen der Smartphones und Tablets jedoch nicht übermäßig zu belasten, werden in der Praxis alle erläuterten Verarbeitungsschritte in einem einzigen physikalischen EPA zusammengefasst. Für die detailierte Syntax der jeweiligen *Ereignisregeln* sei hier auf die Beispiele aus Abschnitt 5.3 oder den Quellcode des Prototyps verwiesen. Bei der *Datensammlung* wird die CPU-Auslastung in periodischen Zeitintervallen ermittelt, in `DynamicSystemEvents` transformiert und über den `InAdapter` in den Ereignisstrom des EPAs eingefügt. Zunächst werden die `DynamicSystemEvents` mit *Kontextinformationen* über die Methoden `getTimeStatus()` und `getLocationStatus()` des `ContextProviders` *angereichert* (siehe Listing 6.5) sowie fachlich *gefiltert*. Die *Ereignisregel* zur *Kontextanreicherung* überführt dabei die exakten Zeit- und Positionsangaben der eintreffenden `DynamicSystemEvents` in abstrakte Zustände, sodass eine Anonymisierung stattfindet. Der nächste Schritt selektiert die Ereignisse zur CPU-Auslastung und überprüft gleichzeitig deren Attribute auf Konsistenz. Konsistente Ereignisse werden anschließend *aggregiert*, sodass die durchschnittliche CPU-Auslastung über einen bestimmten Zeitraum ermittelt wird. Durch ein *Batch*-Zeitfenster wird erst nach Ablauf eines Zeitintervalls ein neues Ereignis zur durchschnittlichen CPU-Auslastung erzeugt. Somit werden in diesem Schritt bereits viele Einzelereignisse zur CPU-Auslastung in ein *aggregiertes* Ereignis einer höheren *Abstraktionsstufe* verdichtet. Im letzten Schritt werden diese *aggregierten* Ereignisse *korreliert* und untereinander auf bestimmte *Ereignismuster* zu signifikanten Veränderungen der CPU-Auslastung anhand spezifischer Schwellenwerte geprüft. Dabei werden komplexe Zustandsereignisse zur CPU-Auslastung erzeugt, die wiederum einen mit der jeweiligen *Ereignisregel* verknüpften Listener anstoßen. Ein komplexes Ereignis zur Zustandsänderung der CPU-Auslastung bildet demnach eine Verdichtung vieler Einzelereignisse der *Datensammlung*. Jeder Verarbeitungsschritt des Prototyps erzeugt entsprechende Ereignisse und fügt sie erneut in den Ereignisstrom des EPAs ein, sodass diese für weitere *Ereignisregeln* verfügbar sind. Die `Statement`-Klassen, der `InAdapter` und der `ContextProvider` liegen im *cep*-Package des Prototyps vor. Zur Reduktion der Komplexität wird jeder der oben erläuterten Verarbeitungsschritte jeweils durch eine eigene *Ereignisregel* realisiert. Insgesamt implementiert der Prototyp des entwickelten Clients somit mehr als 60 *Ereignisregeln* zur Demonstration der Datenverdichtung durch eine mobile CEP-Komponente anhand verschiedener sicherheitsrelevanter Sensor-, System-, Kontext- und Anwendungsdaten der Smartphones. Für jede *dynamische* Sensor-, System- und Anwendungsinformation existiert ein entsprechend zur CPU-Auslastung vergleichbarer Verarbeitungsprozess bestehend aus mehreren *Ereignisregeln* zur *Anreicherung*, *Filterung*, *Aggregation*, *Korrelation* und *Mustererkennung*. *Statische* System- und Anwendungsinformationen können oft nicht sinnvoll *aggregiert* werden, wodurch dieser Verarbeitungsschritt je nach Informationstyp entsprechend ausgelassen wird. Viele *Ereignisregeln* des Prototyps besitzen demnach eine sehr ähnliche Struktur und unterscheiden sich letztlich nur durch ihre Parametrisierung. Zur Verbesserung der Übersichtlichkeit sind die *Ereignisregeln* entsprechend ihrer Fachlichkeit in

den jeweiligen `Statement`-Klassen spezifiziert.

```
public static String getTimeStatus(long timestamp) {
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeZone(TimeZone.getDefault());
        calendar.setTimeInMillis(timestamp);
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        if (hour >= 8 && hour <= 17) {
                return "working_time";
        }
        return "leasure_time";
}
```

Listing 6.5: `ContextProvider` zur *Kontextdatenanreicherung*

6.4 Komponenten zur Datenübertragung

Die mit den *Ereignisregeln* verknüpften Listener bilden die Schnittstellen zwischen der *Datenverarbeitung* und -*übertragung*. Als abstrakte Oberklasse für alle weiteren Listener implementiert der Prototyp den `AbstractListener`. Dieser liest in seiner `update()`-Methode die allgemeinen Attribute aus, die in jedem *angereicherten* `SmartphoneEvent` enthalten sind. Zudem nutzt der Prototyp folgende konkrete Listener: `SensorListener` für Sensorereignisse, `StaticSystemListener` für Ereignisse von *statischen* Systeminformationen, `DynamicSystemListener` für Ereignisse von *dynamischen* Systeminformationen und `ApplicationListener` für Anwendungsereignisse. Listing 6.6 zeigt einen Ausschnitt des `DynamicSystemListeners`. Dieser erbt vom `AbstractListener` und wird dabei u. a. von der `CpuLoadStatus`-*Ereignisregel* angestoßen, die nach komplexen Ereignissen zu signifikanten Zustandsänderungen der CPU-Auslastung sucht. Sobald ein solches `CpuLoadChangeEvent` erkannt wird, ruft die CEP-Komponente die `update()`-Methode des `DynamicSystemListeners` auf und übergibt das auslösende `CpuLoadChangeEvent`.

```
@Override
public void update(EventBean[] newEvents, EventBean[] oldEvents) {
        super.update(newEvents, oldEvents);
        EventBean event = newEvents[0];
        switch (super.getId()) {
        case "cpu":
                handleCpuLoadStatusData((String) event.get("cpuLoadStatus"));
                break;
        }
}
private void handleCpuLoadStatusData(String cpuLoadStatus) {
        Document metadata = super.getFeatureMetadataProvider().createMetadata(
                super.getId(), super.getTimeStatus(), super.getLocationStatus(),
                super.getType(), super.getDescription(), cpuLoadStatus);

        super.getFeatureMetadataProvider().publishMetadata(
                super.getFeatureMetadataProvider().getSmartphoneSystemCpu(),
                null, metadata, MetadataLifetime.session, true);
}
```

Listing 6.6: `DynamicSystemListener` zum Auslesen der CPU-Auslastung

Die Unterscheidung der auslösenden Ereignisse in den Listenern erfolgt anhand des `id`-Attributs. Zusätzlich zu den allgemeinen Ereignisdaten, die bereits durch den `AbstractListener` ausgelesen wurden, werden in den konkreten Listenern weitere spezifische At-

tribute der Ereignisse gelesen. In Listing 6.6 wird beispielsweise das spezifisch zum auslösenden `CpuLoadChangeEvent` gehörende *String*-Attribut `cpuLoadStatus` gelesen. Die Transformation der Ereignisse in IF-MAP Vokabular und die anschließende Übertragung der entsprechend generierten *Metadaten* erfolgen durch den `MetadataProvider`. Dieser wird im Prototyp ebenso wie die `FeatureProvider`- und `FeatureCollector`-Klassen über ein *Factory-Pattern* geladen, wodurch der Mechanismus zur Ereignistransformation sowie das zugrundeliegende IF-MAP Datenmodell je nach Anwendungsdomäne beliebig ausgetauscht werden können. Der Prototyp des Clients verwendet ein konkretes Datenmodell, das auf dem *Feature-Metadatenmodell* des *CADS*-Systems basiert. Dazu implementiert der Prototyp die konkrete `FeatureMetadataProvider`-Klasse, die im `MainService` über die `MetadataProviderFactory` instanziiert wird. Diese implementiert die Methoden `createIdentifier()`, `createLink()` und `createMetadata()` zur Erzeugung des entsprechenden IF-MAP Vokabulars. Listing 6.7 zeigt exemplarisch einen Ausschnitt der `createMetadata()`-Methode, mit deren Hilfe die in den Listenern ausgelesenen Ereignisdaten in jeweilige *Feature-Metadaten* umgewandelt werden können.

```
1  public Document createMetadata(String id, String timeStat,
2                  String locStat, String type, String descr, String value) {
3          Document document = documentBuilder.newDocument();
4          Element feature = document.createElementNS(IfmapConfig.NAMESPACE,
5                  IfmapConfig.NAMESPACE_PREFIX + ":feature");
6
7          feature.setAttributeNS(null, "ifmap-cardinality", "multiValue");
8          feature.setAttribute("ctx-timeStatus", timeStat);
9          feature.setAttribute("ctx-locationStatus", locStat);
10
11         Element idElement = document.createElement("id");
12         idElement.setTextContent(id);
13         feature.appendChild(idElement);
14         ...
15         document.appendChild(feature);
16         return document;
17 }
```

Listing 6.7: `createMetadata()`-Methode zur Erzeugung von *Feature-Metadaten*

Die in Form von XML-Dokumenten generierten *Feature-Metadaten* werden anschließend über die `publishMetadata()`-Methode des `FeatureMetadataProviders` an den MAP-Server publiziert. Zusätzlich zu den übergebenen *Metadaten* können dieser Methode bis zu zwei *Identifier* als Parameter übergeben werden, mit denen die *Metadaten* verknüpft werden sollen. *Feature-Metadaten* werden jedoch üblicherweise mit nur einem *Category-Identifier* verbunden und bilden somit die Blätter in der entstehenden Baumstruktur des *Feature-Metadatenmodells*. Weiterhin erwartet die `publishMetadata()`-Methode einen `MetadataLifetime`-Parameter mit dessen Hilfe die Verweildauer der publizierten *Metadaten* im MAP-Server geregelt wird. Abschließend kann über den *boolean*-Parameter `delete` gesteuert werden, ob bereits im MAP-Server vorhandene *Metadaten* desselben Typs durch das neue *Metadatum* ersetzt werden sollen oder nicht. Dazu sei hier auf den Quellcode des Prototyps verwiesen. Die eigentliche *Datenübertragung* an den MAP-Server wird schließlich durch den *AsyncTask* `PublishTask` in einem gesonderten *Thread* durchgeführt. Der Verbindungsaufbau und -abbau zum MAP-Server wird im Prototyp durch die *AsyncTasks* `ConnectTask` und `DisconnectTask` ausgeführt. Die

Komponenten zur Transformation der Ereignisse in IF-MAP Vokabular sowie sämtliche Klassen zur Kommunikation mit dem MAP-Server sind im *ifmap*-Package des Prototyps zu finden. Die Listener als Schnittstellen zur *Datenverarbeitung* liegen im *eventlistener*-Package des Prototyps vor. Aufbauend auf den Konzepten zur *Datenübertragung* des entworfenen Clients aus Abschnitt 5.4 zeigt Abbildung 6.4 einen Ausschnitt des Klassenmodells des Prototyps mit seinen zentralen Komponenten zur *Datenübertragung*.

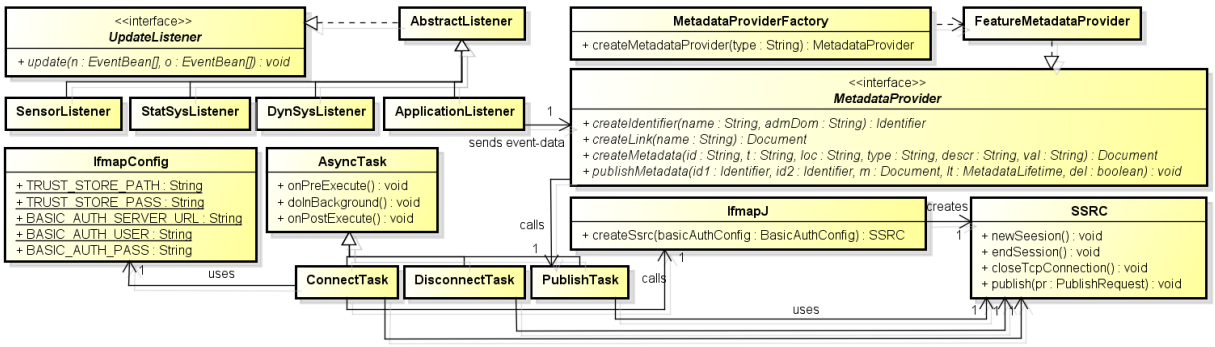

Abbildung 6.4: Klassenmodell: Zentrale Komponenten der *Datenübertragungsschicht*

6.5 Besonderheiten bei der Implementierung

Die Hauptkomponenten `MainActivity` und `MainService` sind für die Ausführung auf der Android-Plattform von Relevanz und müssen daher im Manifest des Prototyps deklariert werden. Die `MainActivity` spezifiziert dabei einen *Intent-Filter*, sodass sie den Einstiegspunkt in die Client-Anwendung bildet. Letztlich dient sie jedoch nur zur Anzeige der Benutzeroberfläche, während die fachliche Funktionalität des eigenen Clients durch den `MainService` im Hintergrund ausgeführt wird. Dieser lokale *Service* kann ausschließlich durch Komponenten des Prototyps mit Hilfe von *Intents* gesteuert werden. Andere auf dem Smartphone befindliche *Apps* haben keinen Zugriff auf den `MainService` und die damit verbundene *Datensammlung*, *-verarbeitung* und *-übertragung*. Der Datenaustausch zwischen dem `MainService` und der `MainActivity` erfolgt über den `LocalBroadcastManager`, wodurch diese Daten ebenfalls nicht durch andere *Apps* ausgespäht werden können. Der Zustand der Benutzeroberfläche wird in der `MainActivity` über ein *Bundle* gesichert, damit diese auch nach einem Neustart der `MainActivity` korrekt angezeigt wird. Durch eine *Notification* besitzt der `MainService` die gleiche Priorität einer im Vordergrund befindlichen Anwendung. Dies minimiert das Risiko, dass der `MainService` bei Speicherknappheit frühzeitig durch das Android-Betriebssystem beendet wird.

Im Manifest werden zudem die benötigten *Permissions* spezifiziert. Zur Aufgabendurchführung benötigt der entwickelte Client grundsätzlich die Berechtigungen zum Zugriff auf das Internet und auf die Geräteposition. Da je nach Anwendungsdomäne beliebige Daten der Smartphones und Tablets gesammelt werden können, sind ggf. weitere *Permissions* notwendig und müssen entsprechend im Manifest ergänzt werden. Der Prototyp nutzt z. B. zusätzlich die *Permission* zum Zugriff auf die Kamera. Damit der Prototyp eine maximale Kompatibilität zu den populärsten Android-Versionen bietet,

sind im Manifest die *minSdk*- und *targetSdk*-Version auf 8 bzw. 21 gesetzt. Bei der *Datensammlung* erfordert dieses breite Spektrum unterstützter Betriebssysteme teilweise die Überprüfung der zugrundeliegenden Android-Version zur Laufzeit, damit die jeweils korrekten Zugriffsmechanismen auf das *Application Framework* verwendet werden.

Der Prototyp besitzt zwei Abhängigkeiten zu den Bibliotheken *ifmapj* und *Asper*. Diese befinden sich im *libs*-Verzeichnis und müssen in den *Build-Path* eingebunden werden. Durch die Größe der *Asper*-Bibliothek benötigt die Kompilierung des Prototyps etwas Zeit. Auf den Smartphones und Tablets nimmt der installierte Prototyp ca. 14 MB Festspeicher ein und benötigt je nach verwendetem Gerät einige Sekunden zum Starten. Trotz der Verwendung einer mobilen CEP-Komponente kann beim Betrieb des Prototyps keine übermäßige Ressourcenauslastung auf den Smartphones und Tablets festgestellt werden, sodass die mobilen Geräte für gewöhnliche Zwecke verwendet werden können, während der Prototyp im Hintergrund ausgeführt wird. Wie die Erfahrungen aus der Testphase zeigen, ist der Ressourcenverbrauch des Prototyps dabei vergleichbar mit dem einer im Hintergrund laufenden Multimedia-Anwendung, kann aber je nach Anzahl der zu verarbeitenden Ereignisse variieren. Beim Start des `MainService` werden *Sensor Observer* und *Broadcast Receiver* im Android-System zur ereignisbasierten *Datensammlung* registriert (siehe Abschnitt 6.1). Damit die Ressourcenauslastung des Prototyps minimiert wird, werden alle Registrierungen direkt beim Stopp des `MainService` aufgehoben. Sämtliche *Sensor Observer* des Prototyps verwenden *normale* Abfrageintervalle, damit die Auswertung der Sensorwerte nicht unnötig oft durchgeführt wird. Aufgrund einer Vielzahl verschiedener Gerätetypen und -modelle mit unterschiedlichen Hardwareausstattungen prüft der Prototyp des eigenen Clients bei der Registrierung der *Sensor Observer*, ob das zugrundeliegende Gerät die benötigten Sensoren besitzt. Demnach registriert der Prototyp nur die *Sensor Observer*, für die die entsprechenden Sensoren auf dem jeweiligen mobilen Gerät verfügbar sind.

Die Definition der *Ereignisregeln* für die Verarbeitung durch die *Asper*-Instanz erfolgt anhand von *Strings* in mehreren `Statement`-Klassen des Prototyps (siehe Abschnitt 6.3). Teilweise erschwert die Regeldefinition mit Hilfe dieser einfachen *Strings* die Fehlersuche, da innerhalb der *Strings* keinerlei Unterstützungen durch Syntaxhervorhebungen o. Ä. möglich sind. Zudem ist die EPL, die von *Asper* zur Regeldefinition verwendet wird, nicht kompatibel mit anderen CEP-Produkten, wodurch das *Regelwerk* beim Austausch der CEP-Komponente entsprechend angepasst werden müsste. Einerseits besitzt die *Asper*-EPL zwar eine SQL-ähnliche Syntax und ist dadurch relativ leicht zu lesen, andererseits existieren jedoch auch signifikante Erweiterungen durch CQL-Befehle u. a. zur Definition von *Sliding Windows* oder *Ereignismustern* (siehe Abschnitt 2.3.4). Somit vermischt die *Asper*-EPL verschiedene Konzepte, wodurch die *Ereignisregeln* wiederum unübersichtlicher werden können und entsprechend schwerer zu verstehen sind. Zur Erhaltung der Übersichtlichkeit teilt der Prototyp die *Ereignisregeln* in die Klassen `SensorStatements`, `StaticSystemStatements`, `DynamicSystemStatements` und `ApplicationStatements` auf. Zudem gliedert sich die Verarbeitung einzelner Ereignisse in mehrere Schritte, die jeweils durch eine eigene *Ereignisregel* spezifiziert sind. Somit umfasst eine *Ereignisregel* des Prototyps immer nur einen Teilschritt zur ereignisbasierten *Datenverarbeitung* und bleibt daher übersichtlich und verständlich (siehe Abschnitt 6.3).

Die Kommunikation des Prototyps mit dem MAP-Server wird über *AsyncTasks* abgewickelt, damit der *Main-Thread* nicht durch langlaufende Netzwerkoperationen blockiert wird. Die Verbindungseinstellungen können über die `IfmapConfig`-Klasse an eine bestehende IF-MAP Infrastruktur angepasst werden. Der Prototyp verwendet für die Authentifizierung am MAP-Server den Keystore von *ironcontrol* der *Trust@HsH*-Forschungsgruppe. Dieser befindet sich in einem speziellen, unter Android lesbaren Format. Details dazu sind im Kontext von *ironcontrol* auf der Webseite der *Trust@HsH*-Forschungsgruppe [34] zu finden. Der Keystore des Prototyps liegt im *assets*-Verzeichnis und wird bei der Erzeugung des `BasicAuthConfig`-Objekts im `ConnectTask` gelesen.

Die Aufgabendurchführung des Prototyps gliedert sich in drei Schichten: *Datensammlung*, *-verarbeitung* und *-übertragung*. Diese sind über schmale Schnittstellen lose gekoppelt, wobei jeweils eine Datentransformation stattfindet. Gesammelte Daten werden zunächst für die Verarbeitung in Ereignisse transformiert. Anschließend werden die verarbeiteten Ereignisse für die Übertragung wiederum in IF-MAP Vokabular umgewandelt. Abbildung 6.5 zeigt dazu zusammenfassend den gesamten Verarbeitungsprozess des Prototyps anhand *dynamischer* Systeminformationen zur CPU-Auslastung. Da die Sammlung einiger Daten nicht auf einem *Android Virtual Device* erfolgen kann, wird der Prototyp auf realen mobilen Geräten getestet. Die Testumgebung umfasst eine IF-MAP Infrastruktur, bestehend aus dem MAP-Server *irond* und dem MAP-Client *VisITMeta* (siehe Abschnitt 2.4.6). Die mit dem Prototyp gesammelten und verdichteten Daten werden dabei in IF-MAP Vokabular des *Feature-Metadatenmodells* transformiert und an *irond* übertragen. *VisITMeta* bezieht diese publizierten Daten und visualisiert sie als Graphenstruktur. Bei den Tests des Prototyps treten jedoch sporadische Anzeigeprobleme in *VisITMeta* in Bezug auf *Feature-Metadaten* auf, sodass in der Konsequenz eine unvollständige Graphenstruktur präsentiert wird. Da es sich dabei allerdings um ein reines Problem innerhalb der *VisITMeta*-Anwendung handelt und sämtliche Daten durch den eigenen Client korrekt gesammelt, verarbeitet und an den MAP-Server übertragen werden, liegt dieser Fehler klar außerhalb des Fokus der vorliegenden Arbeit und betrifft somit nicht das Konzept des eigenen Clients. Die Durchführung von Tests zur Erkennung einiger sicherheitskritischer Situationen stellt teilweise eine Herausforderung in Bezug auf die bewusste Herbeiführung bestimmter komplexer Sachverhalte, wie z. B. die signifikante Veränderung der CPU-Auslastung, auf den realen Smartphones dar.

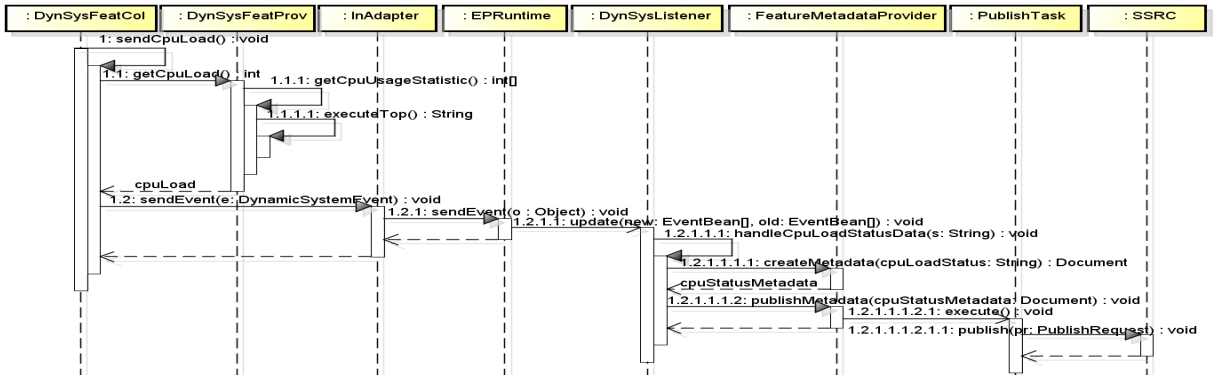

Abbildung 6.5: Sequenzdiagramm: Datenverarbeitungsprozess des entwickelten Clients

7 Bewertung von CEP im Kontext des entwickelten IF-MAP Clients

Nachdem in den vorherigen Kapiteln das Konzept sowie ein Prototyp für den eigenen Client vorgestellt wurden, bei dem eine mobile *Event Processing Engine* als zentrale Komponente zur *Datenverarbeitung* auf den Smartphones und Tablets zum Einsatz kommt, bietet dieses Kapitel einerseits in Abschnitt 7.1 einen Ausblick darauf, welche weiteren Anwendungsmöglichkeiten von CEP im Kontext des eigenen Clients denkbar wären, die über das entworfene Konzept und den entwickelten Prototyp hinausgehen. Andererseits zeigt der Abschnitt 7.2 aber auch auf, welche Grenzen für CEP im Umfeld des IF-MAP Clients auf mobilen Geräten bestehen.

7.1 Ausblick von CEP

CEP im Umfeld mobiler Geräte

Aufgrund der in Abschnitt 4.1 aufgezeigten, enormen Vielfalt an Informationen, die auf modernen Smartphones erfasst werden kann, bieten solche Geräte grundsätzlich eine hervorragende Basis für verschiedene Anwendungsdomänen, bei denen Smartphone-Daten ereignisbasiert durch eine EDA in Verbindung mit einer entsprechenden CEP-Komponente verarbeitet werden sollen. Einerseits besitzen heutige Smartphones und Tablets dazu die Möglichkeit, verschiedene Daten ihrer physikalischen Umgebung mit Hilfe diverser interner oder auch externer Sensoren in teilweise sehr kurzen Zeitabständen zu ermitteln. Andererseits ermöglichen hochentwickelte, mobile Betriebssysteme auch den Zugriff auf dutzende System- und Anwendungsinformationen. Insgesamt bilden moderne Smartphones und Tablets somit aufgrund ihrer großen Informationsvielfalt typische *Ereignisquellen* einer EDA im Kontext diverser Anwendungsszenarien (siehe Abschnitt 2.3.1). Mobile Anwendungen zur Auswertung *dynamischer*, *zeitkritischer* und kontinuierlich eintreffender Sensor-, System- und Anwendungsdaten mit teilweise hohen Änderungsraten erfordern aufgrund der limitierten Smartphone-Ressourcen einen Mechanismus, der große Datenmengen effizient und in nahezu Echtzeit verarbeiten kann. Bei solchen Anwendungen bietet sich die Umsetzung einer EDA in Verbindung mit einer mobilen oder serverbasierten CEP-Komponente an (siehe Abschnitt 2.3.2). Bestehende Systeme im Bereich des *Mobile Event Processing* zeigen dazu bereits eindrucksvoll, dass die Kombination von mobilen Geräten und einer ereignisbasierten *Datenverarbeitung* mit Hilfe einer entsprechenden CEP-Komponente weitreichendes Potential für verschiedene Anwendungsdomänen besitzt (siehe Abschnitt 3.2). Im Kontext mobiler Geräte bietet

sich eine ereignisbasierte *Datenverarbeitung* durch CEP demnach grundsätzlich dann an, wenn eine mobile Anwendung große Mengen *dynamischer* Smartphone-Daten trotz der begrenzten Ressourcen effizient, kontinuierlich und in nahezu Echtzeit verarbeiten soll.

CEP in Verbindung mit dem IF-MAP Kommunikationsprotokoll

Der in dieser Arbeit entwickelte Client erweitert die vielversprechende Kombination aus mobilen Geräten und CEP um das IF-MAP Protokoll als zusätzliche Technologie zum standardisierten Datenaustausch mit anderen Systemen (siehe Abschnitt 2.4). Dabei kann der eigene Client über flexibel anpass-, erweiter- und austauschbare Komponenten als universelle Plattform in diversen Anwendungsdomänen zum Einsatz kommen, bei denen beliebige Smartphone-Daten gesammelt, verarbeitet und anderen Systemen zur Verfügung gestellt werden sollen (siehe Kapitel 5). Die ereignisbasierte *Datenverarbeitung* des entworfenen Clients wird mit Hilfe einer rein mobilen CEP-Komponente direkt auf den Smartphones und Tablets umgesetzt, sodass die gesammelten Daten bereits vor ihrer Übertragung an den MAP-Server zu komplexen Ereignissen fachlich relevanter Situationen mit einem hohen Informationswert verdichtet und ggf. anonymisiert werden können. Aufgrund der Verwendung einer rein mobilen CEP-Komponente besitzt der eigene Client die Eigenschaft, eine Vielzahl von Sensor-, System- und Anwendungsdaten trotz der begrenzten Smartphone-Ressourcen in nahezu Echtzeit verarbeiten zu können und zeitnahe Reaktionen wie die Publikation entsprechend generierter *Metadaten* eines beliebig anpassbaren IF-MAP Datenmodells einzuleiten. Der Prototyp demonstriert die allgemeine Funktionsweise des entworfenen Clients anhand einiger sicherheitsrelevanter Smartphone-Daten. Nach der Sammlung werden diese Daten direkt auf den Smartphones durch die mobile CEP-Komponente verdichtet und anonymisiert, sodass letztlich eine reduzierte Datenmenge an den MAP-Server zur weiteren Analyse durch andere Systeme im Kontext von *CADS* übertragen werden muss (siehe Abschnitt 3.1.2).

Neben dem Einsatz im Bereich der IT-Sicherheit sind weitere Anwendungsszenarien aus verschiedenen Themenfeldern denkbar, bei denen der eigene Client aufgrund der Kombination einer mobilen CEP-Komponente und der Verwendung des IF-MAP Protokolls eine entsprechende Basis bilden kann. Dazu zählen z. B. Anwendungen aus dem Gesundheits- und Fitnessbereich, die Körperdaten über externe Sensoren erfassen und direkt auf den mobilen Geräten verarbeiten. Erkenntnisse über kritische Körperwerte werden anschließend als IF-MAP *Metadaten* über einen MAP-Server mit anderen Systemen ausgetauscht, damit diese weitere Analysen durchführen oder entsprechende Reaktionen einleiten können. Weitere Anwendungsmöglichkeiten sind auch im Logistikbereich denkbar, bei denen der eigene Client auf den Smartphones der Lieferanten zum Einsatz kommen könnte, um GPS- oder Verkehrsdaten auszuwerten und somit in nahezu Echtzeit problematische Situationen im Auslieferungsprozess festzustellen. Das IF-MAP Protokoll kann schließlich dazu verwendet werden, Erkenntnisse über Lieferschwierigkeiten zeitnah mit der Zentrale oder anderen Lieferanten auszutauschen, sodass schnelle Reaktionen in Form von Routenänderungen oder Zusatzfahrten eingeleitet werden können.

Erweiterte Datenverarbeitung durch die mobile CEP-Komponente

Die rein mobile CEP-Komponente des Prototyps verdichtet viele feingranulare Einzelereignisse zu komplexen Ereignissen, die fachlich relevante Situationen zur Smartphone-Sicherheit darstellen. Bei der Behandlung solcher komplexen Ereignisse werden entsprechende *Metadaten* an einen MAP-Server publiziert, sodass wiederum andere Systeme im Kontext von *CADS*, wie *irondetect*, eine Sicherheitsanalyse anhand dieser *Metadaten* durchführen und ihrerseits angemessene Reaktionen einleiten können (siehe Abschnitt 3.1.2). Neben der reinen Veröffentlichung von *Metadaten* könnte die *Ereignisbehandlung* des Clients dahingehend erweitert werden, dass beim Auftreten komplexer Ereignisse direkte Reaktionen auf den Smartphones z. B. in Form von Warnmeldungen oder Zugriffsverweigerungen angestoßen werden. Außerdem könnte die mobile CEP-Komponente um zusätzliche Regeln zur komplexen Anomalienerkennung erweitert werden, wie sie durch *irondetect* durchgeführt wird. Diese Ansätze würden die Abhängigkeit des Clients zu anderen Analysesystemen im IF-MAP Umfeld reduzieren. Die gesamte *Datenverarbeitung* und -*analyse* könnte ereignisbasiert mit Hilfe der mobilen CEP-Komponente des Clients durchgeführt werden, wodurch sich der Vorteil ergibt, dass sämtliche Verarbeitungs- und Analyseschritte durch *Ereignisregeln* in einer einheitlichen Syntax formuliert werden und somit die Verständlichkeit und Wartbarkeit des Gesamtsystems verbessert wird.

Neben dieser Erweiterung der mobilen CEP-Komponente um Funktionalitäten zur direkten Erkennung und Behandlung komplexer Anomalien wäre die Anpassung vorhandener *Ereignisregeln* um zusätzliche Sprachkonstrukte zur erweiterten *Datenverarbeitung* denkbar. Die verwendete, mobile CEP-Komponente *Asper* (siehe Abschnitt 2.3.6) bietet eine mächtige EPL zur Formulierung komplexer *Ereignisregeln* bzw. *Ereignismuster*. Im Folgenden werden daher einige zusätzliche Sprachkonstrukte der *Asper*-EPL erläutert, mit deren Hilfe eine erweiterte *Datenverarbeitung* des Clients umgesetzt werden könnte. Die *Select*-Klausel der *Ereignisregeln* kann um die Schlüsselwörter `istream`, `irstream` und `rstream` ergänzt werden. Diese steuern, welche Ereignisströme an die verknüpften Listener übergeben werden. Bei einem *Sliding Window* sorgt z. B. `istream` dafür, dass nur die das Fenster betretenden Ereignisse beachtet werden, während durch Angabe von `rstream` nur die austretenden Ereignisse von Relevanz sind. Das Schlüsselwort `irstream` bildet eine Kombination beider Varianten. In bestimmten Anwendungsdomänen können somit auch relevante Ereignisse, die ein *Sliding Window* verlassen haben, in entsprechendes IF-MAP Vokabular transformiert und an einen MAP-Server publiziert werden. Mehrfach vorkommende Ereignisse können hingegen durch das zusätzliche Schlüsselwort `distinct` aus dem Ausgabestrom einer *Ereignisregel gefiltert* werden, wodurch eine erweiterte und verbesserte Datenverdichtung realisiert werden kann. Über die `output`-Klausel wird die Ausgaberate einer *Ereignisregel* gesteuert und stabilisiert. Der Ausgabestrom kann dabei u. a. durch die Angabe eines Zeitintervalls oder über die Anzahl der auszugebenen Ereignisse reguliert werden. Diese Stabilisierung bietet damit auch eine weitreichende Kontrolle der Menge der an den MAP-Server zu übertragenden Daten, sodass z. B. bestimmte Schwellenwerte bzgl. der Übertragungsrate nicht überschritten werden. Mit Hilfe der `order by`-Klausel werden Ereignisse anhand bestimmter Attribute sortiert. Das Schlüsselwort `limit` kann in

Kombination mit der `output`- oder `order by`-Klausel verwendet werden, um die Ereignisanzahl im Ausgabestrom auf eine bestimmte Anzahl oder einen bestimmten Bereich zu beschränken und somit eine weitere Reduktion der zu übertragenden Daten zu erreichen. Außerdem können mehrere Ereignisströme durch *Left Outer Joins*, *Right Outer Joins*, *Full Outer Joins* oder *Inner Joins* verknüpft werden, sodass mehrere Einzelereignisse verschiedener Quellen zu einem neuen Ereignis zusammengefasst und somit verdichtet werden. Innerhalb des `pattern[]`-Sprachkonstrukts kann die Wiederholung der *Mustererkennung* auch durch die Verwendung des `repeat`-Operators gesteuert werden. Dabei wird die Suche nach einem *Ereignismuster* für eine bestimmte Anzahl an Durchgängen wiederholt, sodass Anwendungsdomänen unterstützt werden können, bei denen eine limitierte *Mustererkennung* erforderlich ist. Durch sogenannte *Pattern Guards* sind weitere Einschränkungen innerhalb des `pattern[]`-Konstrukts realisierbar. Über das `timer:within()`-Sprachkonstrukt kann dabei z. B. die Erkennung eines *Ereignismusters* auch zeitlich begrenzt werden. Weiterhin erlauben `timer:withinmax()`-*Pattern Guards* die Angabe einer maximalen Anzahl erkannter *Ereignismuster* bis die Ausführung endet. *Pattern Guards* ermöglichen somit diverse Einschränkungen innerhalb der *Mustererkennung*, sodass insgesamt eine stärkere und präzisere Datenverdichtung umgesetzt werden kann. Detailierte Informationen zu weiteren Sprachkonstrukten der *Asper*-EPL sind in [17] zu finden.

Der Prototyp des entworfenen Clients umfasst viele *Ereignisregeln*, die eine ähnliche Struktur besitzen und sich letztlich nur in der Parametrisierung unterscheiden (siehe Abschnitt 6.3). Zur Verbesserung der Übersichtlichkeit des *Regelwerks* sind parametrisierbare *Ereignisregeln* denkbar, sodass ähnliche *Ereignisregeln* zusammengefasst werden können und ihre Anzahl dadurch reduziert wird. Zusätzlich sind im Kontext des eigenen Clients *Ereignisregeln* vorstellbar, die dynamische, am üblichen Smartphone-Nutzungsverhalten ermittelte Schwellenwerte zur Optimierung der Erkennung komplexer Situationen verwenden. Außerdem ist eine erweiterte *Kontextanreicherung* mit zusätzlichen Geräteinformationen denkbar, sodass die gesammelten Daten mit stärkerem Bezug zu den zugrundeliegenden Smartphones oder Tablets verarbeitet werden können. Insgesamt ermöglichen diese Erweiterungen der mobilen CEP-Komponente eine dynamischere sowie an dem jeweiligen Gerätekontext und die jeweils übliche Nutzung der Smartphones flexibel angepasste, ereignisbasierte *Datenverarbeitung* durch ein übersichtliches *Regelwerk*. Somit kann auf den mobilen Geräten anhand der dynamisch ermittelten Schwellenwerte sowie der *Kontextanreicherung* spezifischer Geräteinformationen eine präzisere Erkennung fachlich relevanter Situationen erfolgen, wodurch u. a. eine verbesserte Datenverdichtung bzw. Anomalienerkennung umgesetzt werden könnte.

Weitere Strategien zur Integration von CEP im Kontext mobiler Geräte

Zur ereignisbasierten *Datenverarbeitung* nutzt der entworfene Client eine rein mobile CEP-Komponente. Diese Architekturvariante besitzt den Vorteil, dass die gesamte *Ereignisverarbeitung* direkt auf den mobilen Geräten stattfindet und somit keine Abhängigkeit zu einer serverbasierten CEP-Komponente besteht. Zudem werden das Netzwerk sowie die Netzwerkschnittstellen der Smartphones und Tablets nicht durch

die Übertragung feingranularer Ereignisdaten belastet. Außerdem ermöglicht die mobile CEP-Komponente die Verarbeitung und Anonymisierung personenbezogener Daten direkt auf den Smartphones, bevor diese Informationen als *Metadaten* an einen MAP-Server übertragen werden (siehe Abschnitt 3.3). Somit verlassen diese kritischen Daten in keinem Schritt des ereignisbasierten Verarbeitungsprozesses das zugrundeliegende Gerät. Dies erfordert allerdings die Ausführung der gesamten komplexen *Ereignisverarbeitung* auf den begrenzten Smartphone-Ressourcen. Moderne mobile Geräte besitzen üblicherweise ausreichende Hardwarekapazitäten zur Unterstützung einer solchen mobilen CEP-Komponente. Ältere Smartphones mit leistungsschwächerer Hardware könnten jedoch durch die komplexe *Ereignisverarbeitung* überlastet werden. Zudem hängt die Ressourcenauslastung maßgeblich von der Anzahl der zu verarbeitenden Ereignisse und der Komplexität der *Ereignisregeln* ab. Für den eigenen Client sind demnach auch andere Architekturvarianten zur Integration einer CEP-Komponente denkbar. Dazu bildet beispielsweise die hybride *Datenverarbeitung* aus einer mobilen Ereignisvorverarbeitung und einer serverseitigen, komplexen *Ereignisverarbeitung* einen alternativen Ansatz (siehe Abschnitt 3.3). Die ressourcenintensiven Schritte der *Ereignisverarbeitung* werden dabei auf einem zentralen, leistungsstarken Server ausgelagert, während auf den mobilen Geräten eine Ereignisvorverarbeitung mit entsprechend geringen Leistungsanforderungen umgesetzt wird. Letztlich wird somit eine reduzierte Ereignismenge an den zentralen Server übertragen, wodurch das zugrundeliegende Netzwerk sowie die Netzwerkschnittstellen der Smartphones geschont werden. Außerdem können personenbezogene Daten direkt auf den mobilen Geräten vorverarbeitet und anonymisiert werden, bevor sie an die serverseitige CEP-Komponente übertragen werden. Weiterhin ermöglicht die hybride CEP-Architektur aufgrund des zentralen Servers die Verknüpfung von Ereignissen mehrerer Smartphones, sodass sich damit weitere Anwendungsdomänen für den Client erschließen. Abbildung 7.1 zeigt einen Überblick über eine hybride CEP-Architektur.

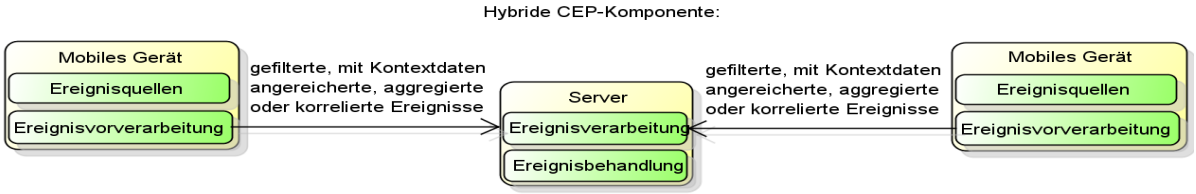

Abbildung 7.1: Hybride CEP-Architektur

Die anpassbaren Komponenten des eigenen Clients zur Erzeugung und Übertragung spezifischer IF-MAP *Metadaten* eines beliebigen Datenmodells bilden eine entsprechende Basis für hybride CEP-Architekturen, bei denen vorverarbeitete Ereignisse als IF-MAP *Metadaten* über einen MAP-Server an eine serverseitige CEP-Komponente übertragen werden. Im Kontext von *CADS* (siehe Abschnitt 3.1.2) bildet der Prototyp des Clients in Verbindung mit der *Correlation-Engine irondetect* eine mit hybriden CEP-Architekturen vergleichbare Kombination zur Verarbeitung sicherheitsrelevanter Smartphone-Daten. Zu der oben vorgestellten, hybriden CEP-Architektur besteht dabei allerdings der Unterschied, dass die durch den Prototyp publizierten *Metadaten* durch *irondetect* nicht

ereignisbasiert mit Hilfe von CEP weiterverarbeitet werden. In speziellen Anwendungsdomänen, bei denen besonders leistungsschwache mobile Geräte zum Einsatz kommen, die keine *Ereignisverarbeitung* unterstützen, ist für den entwickelten Client unter Umständen eine rein serverbasierte CEP-Komponente denkbar (siehe Abschnitt 3.3). Demgegenüber steht allerdings die in der Zielsetzung dieser Arbeit beschriebene Notwendigkeit zur Reduzierung und Anonymisierung der über das Netzwerk übertragenen Daten.

Erweiterung von CEP um eine Zustandsverwaltung

Ähnlich zum System zur Koordination von Rettungswagen (siehe Abschnitt 3.2.2) ist für den eigenen Client eine Erweiterung um eine Zustandsverwaltung denkbar. Der Client spezifiziert dabei ein an die konkrete Anwendungsdomäne angepasstes *Zustandsmodell*. Die CEP-Komponente führt die übliche *Ereignisverarbeitung* durch, wobei ein komplexes Ereignis keine direkte Publikation eines *Metadatums* anstößt, sondern einen Übergang im *Zustandsmodell* bewirkt. Erst bestimmte Zustandsübergänge bzw. das Erreichen bestimmter Zustände führen zur Veröffentlichung von *Metadaten*, wodurch die Verwendung des *Zustandsmodells* eine zusätzliche Datenverdichtung ermöglicht. Außerdem besitzen Ereignisse zwar einen Auftrittszeitpunkt, definieren aber keine Gültigkeitsdauer. Das *Zustandsmodell* bildet Ereignisse auf spezifische Zustände ab, die eine definierte Gültigkeitsdauer besitzen. Ein Zustand ist dabei solange gültig, bis ein Ereignis den Übergang zu einem anderen Zustand auslöst. Zudem können durch das *Zustandsmodell* alle fachlich relevanten Situationen formal spezifiziert und in Beziehung zueinander gesetzt werden. Somit garantiert diese Erweiterung, dass sämtliche durch komplexe Ereignisse ausgelösten Zustandsübergänge sowie die entsprechend publizierten *Metadaten* eine Konsistenz in Bezug auf das jeweils spezifizierte *Zustandsmodell* aufweisen.

7.2 Grenzen von CEP

CEP zur Verarbeitung statischer Smartphone-Daten

Neben den in Abschnitt 7.1 vorgestellten Erweiterungen der CEP-Komponente, zeigt dieser Abschnitt die Grenzen des Einsatzes von CEP im Kontext des eigenen Clients auf. Die ereignisbasierte *Datenverarbeitung* mit Hilfe einer CEP-Komponente eignet sich vor allem für *dynamische*, *zeitkritische* und kontinuierlich eintreffende Daten, die in großen Mengen auftreten und sich in kurzen Zeitintervallen ändern. *Statische* System- und Anwendungsdaten, die sich nicht oder nur sehr selten verändern, können zwar grundsätzlich durch CEP verarbeitet werden, allerdings sind dabei sowohl der Aufwand für die Umsetzung einer ereignisbasierten *Datenverarbeitung* mit Hilfe von *Ereignisregeln* als auch der durch die Verwendung einer mobilen CEP-Komponente gestiegene Leistungsbedarf des Clients oftmals nicht gerechtfertigt. Für Anwendungsdomänen, bei denen der Client ausschließlich *statische* Smartphone-Daten verarbeitet, ist es daher sinnvoll, von einer ereignisbasierten *Datenverarbeitung* mit Hilfe der mobilen CEP-Komponente abzusehen und stattdessen auf *Java*-Bordmittel zur *Datenverarbeitung* zurückzugreifen. Im Kontext solch spezieller Anwendungsdomänen wird somit aufgrund des Verzichts einer mobilen

CEP-Komponente der Ressourcenverbrauch des Clients optimiert. Durch die reduzierte Menge eingesetzter Technologien wird zudem die Übersichtlichkeit und Verständlichkeit des Clients verbessert. Letztlich sollte für jede Anwendungsdomäne des Clients die Verwendung von CEP kritisch hinterfragt werden, indem der Mehraufwand aufgrund des Einsatzes dieser zusätzlichen und komplexen Technologie sowie den dadurch gestiegenen Leistungsanforderungen gegen die Schlüsseleigenschaft einer ereignisbasierten *Datenverarbeitung* durch CEP in Form einer effizienten und nahezu echtzeitfähigen Verarbeitung einer großen und kontinuierlich eintreffenden Datenmenge abgewogen wird.

Leistungsanforderungen einer mobilen CEP-Komponente

Die durch die mobile CEP-Komponente des eigenen Clients verursachte Belastung der limitierten Smartphone-Ressourcen wird maßgeblich von der Menge der zu verarbeitenden Ereignisse sowie der Komplexität der verwendeten *Ereignisregeln* beeinflusst. Die mit Hilfe des Prototyps demonstrierte Verdichtung einer relativ überschaubaren Menge exemplarischer Sicherheitsdaten stellt auf modernen Smartphones kein Problem bzgl. der Ressourcenauslastung dar. Wie die Erfahrungen aus der Testphase zeigen, sind die Leistungsanforderungen des Prototyps dabei in etwa mit denen einer im Hintergrund ausgeführten Multimedia-Anwendung vergleichbar. Allerdings führen Anwendungsdomänen, bei denen deutlich größere Ereignismengen verarbeitet oder komplexere *Ereignisregeln* verwendet werden, zu einem signifikanten Anstieg des Ressourcenverbrauchs der mobilen CEP-Komponente, wodurch die Smartphones stärker belastet werden können. In der Konsequenz könnten die mobilen Geräte aufgrund einer Überlastung während des Betriebs des eigenen Clients nicht sinnvoll für andere Tätigkeiten genutzt werden. Außerdem hätte eine solche Überbeanspruchung der Smartphone-Hardware auch einen starken Anstieg des Energieverbrauchs zufolge. Im Kontext des Clients ist daher je nach zu verarbeitender Ereignismenge sowie Komplexität der verwendeten *Ereignisregeln* von dem Einsatz einer rein mobilen CEP-Komponente abzuraten. Stattdessen sind für solche Anwendungsdomänen andere CEP-Architekturen denkbar, bei denen die ressourcenintensiven Verarbeitungsschritte ausgelagert und somit die begrenzten Smartphone-Ressourcen geschont werden (siehe Abschnitt 7.1).

Umsetzung eines physikalischen EPNs auf mobilen Geräten

Zur Strukturierung des ereignisbasierten Datenverarbeitungsprozesses definiert der eigene Client ein *Verarbeitungsmodell* in Form eines logischen EPNs, bestehend aus mehreren EPAs zur *Anreicherung*, *Filterung*, *Aggregation*, *Korrelation* und *Mustererkennung* (siehe Abschnitt 5.3). Jeder logische EPA besitzt eine klar definierte Teilaufgabe am gesamten Verarbeitungsprozess und umfasst dadurch eine relativ kleine und fachlich eng zusammenhängende Menge von *Ereignisregeln*. Somit verbessert ein solches EPN die Verständlichkeit, Übersichtlichkeit und Wartbarkeit der ereignisbasierten *Datenverarbeitung* durch CEP. Gleichzeitig bildet das EPN die Grundlage für die physikalische Verteilung einzelner Verarbeitungsschritte auf mehreren Systemen. Der Einsatz einer mobilen CEP-Komponente im Kontext des entworfenen Clients beschränkt jedoch die physika-

lische Umsetzung des logischen EPNs zur *Datenverarbeitung*. Aus Performanzgründen werden dabei alle durch die verschiedenen logischen EPAs repräsentierten Verarbeitungsschritte mit Hilfe einer einzigen physikalischen Instanz der mobilen CEP-Komponente auf den Smartphones durchgeführt. Dies hat zur Folge, dass sämtliche *Ereignisregeln* unabhängig ihrer Fachlichkeit mit Hilfe eines einzigen physikalischen EPAs gegen die kontinuierlich eintreffenden Ereignisse ausgeführt werden. Diese Einschränkung begründet sich in den begrenzten Hardwarekapazitäten der mobilen Geräte und bedingt einen teilweisen Verlust der Verständlichkeit und Übersichtlichkeit der *Ereignisverarbeitung*.

Definition von Ereignisregeln im Kontext des entwickelten IF-MAP Clients

Die *Ereignisregeln* der im eigenen Client verwendeten CEP-Komponente *Asper* (siehe Abschnitt 2.3.6) werden als einfache *Strings* spezifiziert, wodurch keine Hilfen bei der Regeldefinition durch Syntaxhervorhebungen o. Ä. existieren. Diese fehlende Werkzeugunterstützung kann eine langwierige und aufwändige Regeldefinition bzw. Fehlersuche oder -behebung zur Folge haben. Die *Asper*-EPL dient zur Definition der jeweiligen Datenverarbeitungsschritte in Form von *Ereignisregeln*. Die verwendeten *Ereignistypen* werden hingegen explizit durch die Implementierung entsprechender *Java*-Beans oder implizit über die `insert into`-Klausel in den *Ereignisregeln* spezifiziert. Die Definition eines dedizierten *Ereignismodells* ist in *Asper* jedoch nicht möglich, sodass sich z. B. bestimmte Beziehungen zwischen den *Ereignistypen* nicht explizit beschreiben lassen. Zudem ist die *Asper*-EPL nicht kompatibel zu anderen CEP-Produkten, wodurch die *Ereignisregeln* beim Austausch der CEP-Komponente in das jeweilige Format überführt werden müssen. Aufgrund einer SQL-ähnlichen Syntax sind *Asper-Ereignisregeln* zwar relativ leicht zu lesen, allerdings besitzen sie auch signifikante Erweiterungen zur Umsetzung typischer CEP-Befehle. Somit sind *Asper-Ereignisregeln* durch die Vermischung verschiedener Sprachparadigmen wiederum unübersichtlicher und schwerer zu verstehen.

CEP als zusätzliche komplexe Technologie

Die Umsetzung einer ereignisbasierten *Datenverarbeitung* durch CEP erfordert den Umgang mit einer zusätzlichen, komplexen Technologie im Kontext des eigenen Clients, der eine Einarbeitung seitens der Entwickler voraussetzt. Neben den Vorteilen dieser Technologie im Bereich der zeitnahen Verarbeitung großer und kontinuierlich eintreffender Ereignismengen sowie der Erkennung komplexer Situationen anhand spezifischer *Ereignismuster*, bedingt die Verwendung einer CEP-Komponente aufgrund des *Ereignis-* und *Verarbeitungsmodells*, des *Regelwerks*, der *Ereignisquellen* und der *Ereignisbehandlung* eine komplexere Gesamtarchitektur des Clients. Nicht zuletzt angesichts der durch den Einsatz einer mobilen CEP-Komponente steigenden Leistungsansprüche sollte demnach die Verwendung einer ereignisbasierten *Datenverarbeitung* durch CEP im Kontext des Clients grundsätzlich für jede Anwendungsdomäne kritisch hinterfragt werden.

8 Fazit

Dieses Kapitel bildet den Abschluss der vorliegenden Arbeit. Zunächst werden die erzielten Ergebnisse in Abschnitt 8.1 zusammengefasst und anschließend in Abschnitt 8.2 kritisch bewertet. Schließlich bietet Abschnitt 8.3 einen Ausblick auf die zukünftigen Entwicklungen des eigenen Clients.

8.1 Zusammenfassung

Moderne Smartphones und Tablets gewinnen im geschäftlichen Umfeld vieler Unternehmen zunehmend an Bedeutung. Allerdings rücken gerade mobile Geräte mit dem Android-Betriebssystem von *Google* aufgrund der dominierenden Marktposition und der offenen Plattform immer stärker in den Fokus krimineller Aktivitäten. Demnach stellt die sichere Integration solcher Geräte in die eigene IT-Infrastruktur für viele Unternehmen eine Schlüsselaufgabe dar, um einerseits die Produktivität der Mitarbeiter zu erhöhen und andererseits die Bedrohungen, die durch solche Geräte für die unternehmenskritischen Ressourcen ausgehen, zu minimieren. Die vorliegende Arbeit befasst sich daher mit dem primären Ziel, einen IF-MAP Client in Form einer Android-*App* zu entwickeln, der die sichere Integration von Smartphones und Tablets in Unternehmensnetze ermöglicht. Dazu sammelt der entworfene Client sicherheitsrelevante Smartphone-Daten im Hintergrund und stellt diese mit Hilfe des IF-MAP Protokolls anderen Systemen für die weitere Analyse auf einem MAP-Server zur Verfügung. Der Unterschied zu ähnlichen vorhandenen Systemen besteht in der Verwendung einer direkt auf den Smartphones und Tablets eingesetzten, mobilen CEP-Komponente, die die gesammelten Daten bereits vor der Übertragung an den MAP-Server verdichtet und anonymisiert. Somit werden sowohl das Netzwerk aufgrund der reduzierten Menge übertragener Daten geschont als auch Datenschutzkonflikte durch die frühzeitige Anonymisierung personenbezogener Daten verhindert. Die Zielsetzung der vorliegenden Arbeit umfasst zudem, dass der Client aufgrund flexibel anpass-, erweiter- und austauschbarer Komponenten in den Bereichen der *Datensammlung*, *-verarbeitung* und *-übertragung* diverse weitere Anwendungsdomänen unterstützen soll, bei denen beliebige, über das Themenfeld der IT-Sicherheit hinausgehende Smartphone-Daten von Relevanz sind. Zu den weiteren Zielen dieser Arbeit zählen neben der Analyse und Abgrenzung bestehender fachlicher und technischer Lösungen auch die Konstruktion eines konkreten Anwendungsszenarios sowie die Realisierung des entworfenen Konzepts in Form eines Prototyps, der die Funktionsweise des Clients anhand dieses Anwendungsszenarios demonstriert. Schließlich beinhaltet die Zielsetzung dieser Arbeit auch die Bewertung des grundsätzlichen Einsatzes von CEP im Client. Einerseits erfolgt dazu ein Ausblick auf zusätzliche Erweiterungen der CEP-Komponente, andererseits werden aber auch die Grenzen von CEP im Kontext des Clients aufgezeigt.

Anhand dieser Zielsetzung bildet der Client eine Kombination der drei Technologien Android, CEP und IF-MAP, deren Grundlagen neben einem Überblick über mobile Geräte und Plattformen in Kapitel 2 näher erläutert werden. Dabei werden auch die beiden im Client eingesetzten Bibliotheken *Asper* und *ifmapj* vorgestellt. Kapitel 3 bietet eine Analyse bestehender fachlicher Systeme im Bereich der Smartphone-Sicherheit sowie technischer Systeme zum Einsatz von CEP im Kontext mobiler Geräte. Da bisher keine zufriedenstellende Lösung für die Ziele dieser Arbeit existiert, stellt der Client letztlich eine Kombination mehrerer Systeme dar, bei der das netzwerkbasierte Sicherheitskonzept *CADS* die zentrale Grundlage bildet. Im Kontext der Auswertung sicherheitsrelevanter Smartphone-Daten gliedert sich der eigene Client als modifizierter *Feature Collector* in die *CADS*-Architektur ein. Allerdings wird aufgrund der durch CEP umgesetzten Datenverdichtung eine im Vergleich zu herkömmlichen *Feature Collectors* reduzierte Datenmenge an den MAP-Server übertragen. Zudem spezifiziert *CADS* das im Client verwendete *Feature-Metadatenmodell* zur Repräsentation von Smartphone-Daten als IF-MAP Vokabular. Die technischen Lösungen demonstrieren verschiedene Architekturvarianten zur Integration einer CEP-Komponente im Umfeld mobiler Geräte. Der Client nutzt dabei eine rein mobile CEP-Komponente, die die Datenverdichtung und -anonymisierung direkt auf den Smartphones ausführt. In Kapitel 4 werden die Anforderungen an den Client anhand eines Anwendungsszenarios im Kontext sicherheitsrelevanter Smartphone-Daten analysiert. Zusätzlich erfolgt eine Auflistung und Kategorisierung der auf mobilen Geräten sammelbaren Daten sowie eine Analyse zu Datenschutzkonflikten. In Kapitel 5 wird der Client entworfen. Das Konzept des Clients erlaubt dabei eine Ausführung im Hintergrund. Zudem werden die Strategien zur Umsetzung flexibel anpass-, erweiter- und austauschbarer Komponenten zur *Datensammlung*, *-verarbeitung* und *-übertragung* vorgestellt. Aufgrund der Instanziierung über *Factory*-Klassen kann die *Datensammlung* beliebig angepasst werden. Außerdem können im eigenen Client verschiedene *Sensor Observer* und *Broadcast Receiver* registriert werden. Durch die Anpassung der deklarativen *Ereignisregeln* kann die *Datenverarbeitung* modifiziert werden. Schmale Schnittstellen ermöglichen auch den Austausch der gesamten CEP-Komponente. Über ein weiteres *Factory-Pattern* werden die Komponenten zur Generierung und Übertragung entsprechender IF-MAP *Metadaten* erzeugt. Somit kann einerseits das verwendete *Feature-Metadatenmodell* beliebig angepasst und andererseits das gesamte Datenmodell ausgetauscht werden. In Kapitel 6 wird das entworfene Konzept als Prototyp zur Sammlung, Verarbeitung und Übertragung einiger exemplarischer Sicherheitsdaten im Kontext von *CADS* realisiert. Zudem werden die Besonderheiten bei der Implementierung erläutert. In Kapitel 7 erfolgt eine Bewertung zur grundsätzlichen Verwendung von CEP im Kontext des eigenen Clients, indem einerseits Erweiterungsmöglichkeiten vorgestellt und andererseits die Einschränkungen des Einsatzes von CEP im Client aufgezeigt werden.

8.2 Bewertung

Das in Abschnitt 4.2 entwickelte Anwendungsszenario im Kontext sicherheitsrelevanter Smartphone-Daten bildet eine hervorragende Basis zur Analyse der Anforderungen an den entwickelten Client bzgl. der Zielsetzungen dieser Arbeit. Unter Verwendung

leistungsstarker mobiler Geräte werden dabei sämtliche Anforderungen durch den eigenen Client erfüllt. Dieser zeichnet sich vor allem dadurch aus, dass er aufgrund flexibel anpass-, erweiter- und austauschbarer Komponenten die Sammlung, Verarbeitung und Übertragung beliebiger Smartphone-Daten unterstützt und demzufolge unabhängig von einer konkreten Anwendungsdomäne ist. Ein wesentlicher Vorteil des Konzepts besteht in der Umsetzung einer ereignisbasierten *Datenverarbeitung* mit Hilfe einer mobilen CEP-Komponente (siehe Abschnitt 5.3). Der eigene Client ermöglicht somit eine echtzeitfähige Verarbeitung großer und kontinuierlich eintreffender Datenmengen. Ein flexibel erweiterbares *Ereignismodell* sowie beliebig modifizierbare *Ereignisregeln* unterstützen hierbei eine an konkrete Anwendungsdomänen anpassbare Verdichtung vieler Einzelereignisse zu komplexen Ereignissen einer höheren *Abstraktionsstufe*. Dadurch erreicht der Client letztlich die geforderte Eigenschaft, eine reduzierte und anonymisierte Datenmenge an den MAP-Server zu übertragen. Von Vorteil sind zudem die flexiblen Komponenten zur Generierung und Übertragung von IF-MAP *Metadaten*, sodass der Client in beliebige IF-MAP Infrastrukturen integriert werden kann (siehe Abschnitt 5.4). Positiv zu bewerten sind außerdem die Eigenschaften, dass der Client die populärsten Android-Versionen unterstützt und als leichtgewichtige *App* keine tiefgreifenden Modifikationen am Betriebssystem erfordert. Durch die Umsetzung eines *Services* erfüllt der Client zudem die Anforderung, seine gesamte Funktionalität im Hintergrund ausführen zu können (siehe Abschnitt 5.1). Der ressourcenschonende Umgang des Clients ist jedoch aufgrund des Einsatzes der mobilen CEP-Komponente kritisch zu betrachten. Die Leistungsanforderungen hängen dabei maßgeblich vom Umfang der zu verarbeitenden Ereignisse sowie der Komplexität des Verarbeitungsprozesses der CEP-Komponente ab, wodurch leistungsschwächere Geräte überlastet werden können und einen signifikant gestiegenen Energieverbrauch aufweisen (siehe Abschnitt 7.2). Grundsätzlich bedingt der Einsatz von CEP im Kontext des IF-MAP Clients den Umgang mit einer zusätzlichen komplexen Technologie, die die Übersichtlichkeit und Verständlichkeit des Clients für außenstehende Entwickler negativ beeinflusst. Demnach sollte die ereignisbasierte *Datenverarbeitung* durch CEP im Kontext des Clients grundsätzlich für jede Anwendungsdomäne kritisch hinterfragt werden. Abschließend soll diese kritische Bewertung verdeutlichen, dass das vorgestellte Konzept des Clients zwar eine angemessene Lösung für viele Anwendungsdomänen bietet, allerdings sollten je nach konkretem Anwendungsszenario auch Anpassungen und Erweiterungen des Clients in Betracht gezogen werden.

8.3 Ausblick

Die Bewertung zeigt, dass der entwickelte IF-MAP Client neben der Verarbeitung von Sicherheitsinformationen auch eine Unterstützung für diverse weitere Anwendungsdomänen bietet, bei denen beliebige Smartphone-Daten gesammelt, durch CEP verarbeitet und an einen MAP-Server übertragen werden sollen. Somit könnte der Client in Zukunft auch in anderen Anwendungsgebieten, wie z. B. dem Gesundheits- und Fitnessbereich zum Einsatz kommen, die über das Konzept der sicheren Integration von Smartphones und Tablets in IT-Infrastrukturen hinausgehen. Die Kombination der Technologien CEP und IF-MAP bietet dabei eine hervorragende Basis für sämtliche zukünftige An-

wendungsdomänen, bei denen große Datenmengen effizient verarbeitet und auf standardisierte Weise mit anderen Netzwerkkomponenten ausgetauscht werden sollen. Hinsichtlich der Eliminierung der aufgezeigten Defizite in Bezug auf die durch CEP verursachten Leistungsanforderungen wären verschiedene Architekturvarianten zur Integration einer CEP-Komponente im Umfeld mobiler Geräte denkbar, die entsprechend der zukünftigen Anwendungsbereiche innerhalb des Clients umgesetzt werden könnten. Dazu zählen z. B. eine rein serverbasierte CEP-Architektur oder eine hybride Variante aus einer mobilen und einer serverseitigen CEP-Komponente. Eine weitere Option für zukünftige Entwicklungen des Clients bildet der Austausch der mobilen CEP-Komponente gegen ein anderes System zur *Datenverarbeitung*. Je nach Anwendungsdomäne stellt auch die Erweiterung der CEP-Komponente um eine Zustandsverwaltung eine Möglichkeit für kommende Entwicklungen des Clients dar. Ein zusätzlicher Schwerpunkt für künftige Arbeiten am entwickelten IF-MAP Client kann in der Erweiterung des *Regelwerks* für die ereignisbasierte *Datenverarbeitung* bestehen. Dabei wären u. a. parametrisierbare *Ereignisregeln* vorstellbar, sodass ähnliche *Ereignisregeln* zusammengefasst werden können und somit die Übersichtlichkeit des gesamten *Regelwerks* verbessert wird. Außerdem wären eine stärkere Datenverdichtung mit Hilfe weiterer Sprachkonstrukte innerhalb bestimmter *Ereignisregeln* sowie eine grundsätzliche Erweiterung des *Regelwerks* auf zusätzliche Aspekte zur Anomalienerkennung vorstellbar. Durch eine direkte Erkennung und Behandlung von Anomalien auf den Smartphones und Tablets könnte in zukünftigen Anwendungsbereichen die Abhängigkeit zu anderen Systemen im IF-MAP Umfeld reduziert werden. Zusätzliche Erweiterungen der ereignisbasierten *Datenverarbeitung* werden in Abschnitt 7.1 erläutert. Auch die mobilen Geräte selbst befinden sich derzeit in einem kontinuierlichen Entwicklungsprozess. Einerseits hat dies zur Folge, dass immer mehr Informationen über diverse Datenquellen (z. B. neuartige interne oder externe Sensoren) der Smartphones und Tablets bezogen werden können und diese mobilen Geräte somit in Zukunft in den Fokus weiterer Anwendungsdomänen des entwickelten IF-MAP Clients rücken werden. Andererseits werden die steigenden Hardwarekapazitäten der mobilen Geräte künftig eine noch komplexere *Datenverarbeitung* mittels einer mobilen CEP-Komponente ermöglichen, ohne dass die Ressourcen der Smartphones und Tablets zu stark beansprucht werden. Dazu ist in Zukunft beispielsweise ein physikalisches EPN aus mehreren verknüpften EPAs im Kontext des entwickelten IF-MAP Clients auf den mobilen Geräten denkbar, sodass ein strukturierterer Datenverarbeitungsprozess realisiert werden könnte. Auch in Bezug auf die Kommunikation des entwickelten Clients über das IF-MAP Protokoll sind zukünftige Erweiterungen und Modifikationen vorstellbar. Dabei könnte z. B. das derzeit verwendete *Feature-Metadatenmodell* gegen ein anderes Datenmodell ausgetauscht werden, bei dem sämtliche Informationen direkt mit dem jeweiligen *device-Identifier* verknüpft werden, sodass eine flachere Baumstruktur entsteht. Darüber hinaus wären in Zukunft für den Client auch Abonnements für *request-for-investigation-Metadaten* beim MAP-Server denkbar, sodass andere Netzwerkkomponenten die Sammlung, Verarbeitung und Übertragung bestimmter Daten explizit anstoßen könnten. Letztlich wäre es für zukünftige Entwicklungen auch vorstellbar, das gesamte Konzept des in dieser Arbeit entworfenen IF-MAP Clients für die Android-Plattform auf andere mobile Plattformen wie iOS von *Apple* zu portieren.

Literaturverzeichnis

[1] Amadei, D. : *Joining Oracle Complex Event Processing and J2ME to React to Location and Positioning Events*. Verfügbar online unter `http://www.oracle.com/technetwork/articles/amadei-cep-090595.html`, 2010. – Abgerufen am 15.10.2014

[2] Arango, M. : Mobile QoS management using complex event processing (industry article). In: *Proceedings of the 7th ACM international conference on Distributed event-based systems*, ACM, 2013, S. 115–122

[3] Asper: *Asper - Esper for Android*. Verfügbar online unter `https://github.com/mobile-event-processing/Asper`, 2014. – Abgerufen am 22.09.2014

[4] Bellenger, D. : *Complex Event Processing in verteilten, mobilen Umgebungen - Masterarbeit*. Hochschule Hannover, 2011

[5] Bente, I. : *Towards a Network-based Approach for Smartphone Security - Dissertation*. Universität der Bundeswehr München, 2013

[6] BITKOM: *Smartphone-Boom setzt sich 2014 ungebrochen fort*. Verfügbar online unter `http://www.bitkom.org/de/presse/8477_78640.aspx`, 2014. – Abgerufen am 08.09.2014

[7] Bruns, R. ; Dunkel, J. ; Billhardt, H. ; Lujak, M. ; Ossowski, S. : Using Complex Event Processing to Support Data Fusion for Ambulance Coordination. In: *Proceedings of the 17th International Conference on Information Fusion*, IEEE, 2014, S. 1–7

[8] Bruns, R. ; Dunkel, J. : *Event-Driven Architecture - Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Springer, 2010

[9] Burguera, I. ; Zurutuza, U. : Crowdroid: behavior-based malware detection system for Android. In: *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, 2011, S. 15–25

[10] Cheng, J. ; Wong, S. ; Yang, H. : SmartSiren: virus detection and alert for smartphones. In: *Proceedings of the 5th international conference on Mobile systems, applications, and services*, ACM, 2008, S. 258–271

[11] DECOIT: *DECOIT IF-MAP Client*. Verfügbar online unter `https://github.com/decoit/Android-IF-MAP-Client`, 2013. – Abgerufen am 08.10.2014

[12] Dunkel, J. ; Koschel, A. : *Software Engineering III - Vorlesungsskript - SoSe2014*. Hochschule Hannover, 2014

[13] Eckert, M. ; Bry, F. : Complex event processing (CEP). In: *Informatik-Spektrum 32.2*, Springer, 2009, S. 163–167

[14] Enck, W. ; Gilbert, P. ; Chun, B. ; Cox, L. ; Jung, J. ; McDaniel., P. ; Sheth, A. : TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. In: *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010, S. 1–15

[15] Enck, W. ; Ongtang, M. ; McDaniel, P. : *Mitigating Android software misuse before it happens*. Verfügbar online unter `http://www.enck.org/pubs/NAS-TR-0094-2008.pdf` : Pennsylvania State University, 2008

[16] EsperTech: *Esper - Complex Event Processing*. Verfügbar online unter `http://esper.codehaus.org/`, 2014. – Abgerufen am 22.09.2014

[17] EsperTech: *Esper Reference Documentation*. Version 5.0.0, 2014

[18] Funk, C. ; Garnaeva, M. : *Kaspersky Security Bulletin 2013/2014 - Statistik für das Jahr 2013*. Verfügbar online unter `http://www.viruslist.com/de/analysis?pubid=200883839#02`, 2013. – Abgerufen am 09.09.2014

[19] Gartner: *Gartner Says Annual Smartphone Sales Surpassed Sales of Feature*. Verfügbar online unter `http://www.gartner.com/newsroom/id/2665715`, 2014. – Abgerufen am 17.09.2014

[20] Google: *Android Developers*. Verfügbar online unter `http://developer.android.com/develop/index.html`, 2014. – Abgerufen am 25.09.2014

[21] Hornyack, P. ; Han, S. ; Jung, J. ; Schechter, S. : These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In: *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, S. 639–652

[22] IDC: *Smartphone OS Market Share, Q2 2014*. Verfügbar online unter `http://www.idc.com/prodserv/smartphone-os-market-share.jsp`, 2014. – Abgerufen am 08.09.2014

[23] Ruhe, T. : *Development of an Android Usage Study System - Masterarbeit*. Hochschule Hannover, 2012

[24] Seeger, B. : *Kontinuierliche Kontrolle - Complex Event Processing: Auswertung von Datenströmen*. Verfügbar online unter `http://heise.de/-905334`, 2010. – Abgerufen am 01.10.2014

[25] StartMobile: *Die Meilensteine in der Geschichte der Handys und Mobilfunktelefone*. Verfügbar online unter `http://www.startmobile.net/die-meilensteine-in-der-geschichte-der-handys-und-mobilfunktelefone/`, 2012. – Abgerufen am 18.09.2014

[26] Statista: *Statistiken und Studien zum Thema Smartphones*. Verfügbar online unter `http://de.statista.com/themen/581/smartphones/`, 2014. – Abgerufen am 07.09.2014

[27] Stipkovic, S. : *Complex Event Processing auf mobilen Systemen - Bachelorarbeit*. Hochschule Hannover, 2012

[28] Stipkovic, S. ; Bruns, R. ; Dunkel, J. : Pervasive Computing by Mobile Complex Event Processing. In: *Proceedings of the 10th IEEE International Conference on e-Business Engineering (ICEBE 2013)*, IEEE, 2013, S. 318–323

[29] Stojanovic, N. ; Xu, Y. ; Stojadinovic, A. ; Stojanovic, L. : Using mobile-based complex event processing to realize collaborative remote person monitoring. In: *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, ACM, 2014, S. 225–235

[30] TCG: *TCG Architecture Overview*. Version 1.4, 2007

[31] TCG: *TNC Architecture for Interoperability Specification*. Version 1.5 - Revision 3, 2012

[32] TCG: *TNC IF-MAP Metadata for Network Security*. Version 1.1 - Revision 8, 2012

[33] TCG: *TNC IF-MAP Binding for SOAP Specification*. Version 2.2 - Revision 9, 2014

[34] Trust@HsH-Forschungsgruppe: *Trust@HsH*. Verfügbar online unter `http://trust.f4.hs-hannover.de/`, 2014. – Abgerufen am 06.10.2014

[35] Vogel, L. : *Android Development*. Verfügbar online unter `http://www.vogella.com/tutorials/android.html`, 2014. – Abgerufen am 26.09.2014

[36] Westhuis, J. ; Bruns, R. ; Dunkel, J. ; Knop, W. ; Eichstädt, T. : Intelligente Lokalisierung von Gegenständen mit Complex Event Processing. In: *BTW Workshops*, 2011, S. 33–42

[37] Wikipedia: *Mobilgerät*. Verfügbar online unter `http://de.wikipedia.org/wiki/Mobilger%C3%A4t`, 2014. – Abgerufen am 10.10.2014

[38] Xu, Y. ; Stojanovic, N. ; Stojanovic, L. ; Kostic, D. : An Approach for Dynamic Personal Monitoring based on Mobile Complex Event Processing. In: *Proceedings of International Conference on Advances in Mobile Computing and Multimedia*, ACM, 2013, S. 464

Anhang

A. CD-ROM

Die beigefügte CD-ROM umfasst die folgenden Inhalte:

- Sourcen des implementierten Client-Prototyps (*Eclipse*-Projekt)
- Kompilierter und gepackter Client-Prototyp als Android-*App* (.apk-Datei)
- Diese Arbeit in elektronischer Form (.pdf-Datei)
- Abbildungen und Diagramme der vorliegenden Arbeit