Entwicklung eines IF-MAP Clients für die Android Plattform

Bachelorarbeit im Studiengang Angewandte Informatik an der Fakultät IV - Wirtschaft und Informatik Fachhochschule Hannover

Waldemar Bender
Waldemar.Bender@stud.fh-hannover.de

29. Juli 2010

Beteiligte

Autor

Waldemar Bender
Zum Mühlenberg 18b
30952 Ronnenberg
E-Mail: Waldemar.Bender@gmx.com

Erstprüfer

Prof. Dr. rer. nat. Josef von Helden
Ricklinger Stadtweg 120
30459 Hannover

Raum 212, 354
Tel.: +49 511 9296-1500
E-Mail: josef.vonhelden@fh-hannover.de

Zweitprüfer

Ingo Bente M.Sc.
Ricklinger Stadtweg 120
30459 Hannover

Tel.: +49 511 9296-1828
E-Mail: ingo.bente@fh-hannover.de

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Bachelorarbeit selbständig 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.

Ronnenberg, den 29. Juli 2010 .....
Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Verwendung von mobilen Endgeräten spielt in der Wirtschaft eine immer größere Rolle. Die Benutzung von Mobilfunkgeräten beschränkt sich schon länger nicht mehr auf dem Telefonieren und SMS Schreiben. Es werden immer komplexere Anwendungen für Mobilfunkgeräte entwickelt. Diese Anwendungen nutzen vermehrt das Internet als Kommunikationsplattform. Beispiele hierfür sind verteilte Kalender, Plattformen zur Terminfindung oder auch Plattformen zur internen Informationsverteilung (interne Wikis).
Durch die gute technische Ausstattung und der immer besser werdenden Netzabdeckung von UMTS ist es komfortabel geworden, von unterwegs mit einem mobilen Endgerät wie z.B. einen Tablet-PC, Mobilfunkgerät oder Notebook zu arbeiten. Um dabei unternehmensinterne Ressourcen zu nutzen, muss es möglich sein, eine Netzwerkverbindung vom mobilen Endgerät zum Firmen-Netzwerk aufzubauen. Die Tatsache, dass sich diese mobilen Endgeräte nicht auf dem Firmengelände befinden, vergrößert das Risiko der Manipulation oder Entwendung der Geräte. Diese Arbeit befasst sich mit einem Teilaspekt, der entscheidend dazu beiträgt, Risiken, die von manipulierten Endgeräten ausgehen, zu minimieren.
Entstanden ist diese Arbeit in Kooperation mit dem Fraunhofer-Institut für Sichere Informationstechnologie (Fraunhofer SIT). Das Fraunhofer SIT befasst sich mit IT-Sicherheitslösungen für jegliche Auftraggeber.

1.1 Trusted Computing Group

Da diese Arbeit auf spezifizierte Standards der Trusted Computing Group aufbaut, wird folgend erläutert, welche Ziele die TCG verfolgt.
Die TCG beschreibt sich selbst wie folgt:

> The Trusted Computing Group (TCG) is a not-for-profit organization formed to develop, define and promote open, vendor-neutral, industry standards for trusted computing building blocks and software interfaces across multiple platforms. [1]

Die im Zitat genannten "trusted computing building blocks" sind Plattformen, die die Integrität verschiedener Endgeräte oder Software kontrollieren. Diese Plattformen existieren in Form eines Hardwarechips (Trusted Platform Module). Mit einem TPM wird

[1] http://www.trustedcomputinggroup.org/ (Juli 2010)

die Integrität von Hard- und Software auf dem Endgerät gemessen. Durch die Abspeicherung dieser Messung kann man feststellen, ob an dem Endgerät eine Manipulation stattgefunden hat.

1.2 Aufgabenstellung und Ziele

In dieser Arbeit soll untersucht werden, welche Möglichkeiten eine Integration von IF-MAP auf die Android Plattform ermöglicht. Hierfür soll ein Beispielszenario entwickelt werden, das eine Möglichkeit aufzeigt IF-MAP auf mobilen Endgeräten zu benutzen.
Das Ziel der Arbeit ist es, ein sinnvolles Konzept auszuarbeiten. Dieses Konzept erläutert, welche Funktionalitäten ein IF-MAP Client auf einer Android-Plattform vorweisen soll. Weiterhin muss untersucht werden, welche Besonderheiten bei der Anwendungsentwicklung auf der Android-Plattform zu beachten sind. Als Produkt dieser Arbeit soll das entwickelte Konzept vollständig implementiert werden.

1.3 Aufbau

Im Kapitel 2 wird auf die in dieser Arbeit benutzten Basistechnologien eingegangen. Daraufhin werden in Kapitel 3 die Technologien TNC und IF-MAP behandelt. Dieses sind die Haupttechnologien, die in dieser Arbeit eine zentrale Rolle spielen. Die im Kapitel 4 vorgestellten Szenarien untersuchen mögliche Verwendungen eines IF-MAP Clients im mobilen Umfeld. Im Kapitel 5 wird festgelegt, welche Funktionalitäten ein IF-MAP Client auf einer Android-Plattform vorweisen muss und was die entscheidenden Erfolgskriterien sind, um diese Funktionalitäten zu realisieren. Daraus resultieren dann die Anforderungen. Anschließend wird in Kapitel 6 untersucht, welche Besonderheiten eine Implementierung eines IF-MAP Clients auf Android mit sich bringt. Im Kapitel 7 wird ein komplettes Konzept für die Integration des IF-MAP Client aufgestellt. In diesem Kapitel werden Architekturentscheidungen erläutert und die nötigen Dokumente zur Implementierung vorgestellt. Weiterhin werden durch das Konzept erfüllte Anforderungen erläutert. Besonderheiten, die bei der Realisierung aufgetreten sind, werden in Kapitel 8 behandelt. Ausserdem wird die beispielhafte Benutzung der Implementierung vorgestellt. Anschließend werden die durch die Implementierung erfüllten Anforderungen erläutert. Abschließend wird in Kapitel 9 die Zielrealisierung der Arbeit kritisch reflektiert.

2 Basistechnologien

In diesem Kapitel wird auf die verwendeten Technologien erläutert, welche die Grundlage für das Verständnis bilden.

2.1 XML

Mit XML (Extensible Markup Language) werden Daten in Form einer Baumstruktur hierarchisch angeordnet. XML gilt als MetaMarkup-Sprache, sie dient zur logischen Beschreibung von Daten. Meist wird XML benutzt, um einen Nachrichtenaustausch zwischen verschiedenen Computersystemen zu gewährleisten. Hierbei ist es irrelevant mit welchem Betriebssystem diese Systeme laufen oder welches Protokoll für den Austausch der Nachrichten benutzt wird. Diese Plattform- und Protokollunabhängigkeit ist einer der Hauptgründe bzw. -faktoren für den Erfolg von XML.

2.1.1 XML Aufbau

```
<?xml version="1.0"?>
<bachelorarbeit>
        <titel> Entwicklung eines IF-MAP Clients fuer die Android Plattform </titel>
        <autor> Waldemar Bender <autor>
        <gliederung>
                <kapitel nr="1">
                        <ueberschrift> Einleitung </ueberschrift>
                </kapitel>
        </gliederung>
</bachelorarbeit>}
```

Listing 2.1: Simple XML - Beispielcode

Das XML Dokument ist in einer hierarchischen Darstellung von Elementen angeordnet. Elemente stellen die Struktur des Dokumentes dar. Sie dienen als Datencontainer. Das Element "xml version" deklariert die benutzte Version von XML. Als zweites Tag folgt immer ein Wurzelelement. In einem XML Dokument muss immer ein Wurzelelement vorhanden sein. Wenn dieses nicht vorhanden ist, ist das XML Dokument nicht "well formed". Es entspricht dann nicht der XML Spezifikation. In diesem Wurzelelement folgen weitere Elemente. Wichtig hierbei ist, dass innere Elemente immer vor Austritt des äußeren Elements geschlossen werden müssen. Ein solches Konstrukt ist nicht "well formed":

```
<?xml version="1.0"?>
<bachelorarbeit>
        <titel> Entwicklung eines IF-MAP Clients fuer die Android Plattform </titel>
        <autor> Waldemar Bender <autor>
        <gliederung>
                <kapitel nr="1">
                        <ueberschrift> Einleitung </ueberschrift>
        </gliederung>
                </kapitel>
</bachelorarbeit>
```

Das innere Element "gliederung" schließt, bevor das Element "kapitel" geschlossen wurde. Die korrekte XML Nachricht aus Listing 2.1 kann, wie in Abbildung 2.1 gezeigt, anschaulich als Baumstruktur abgebildet werden.

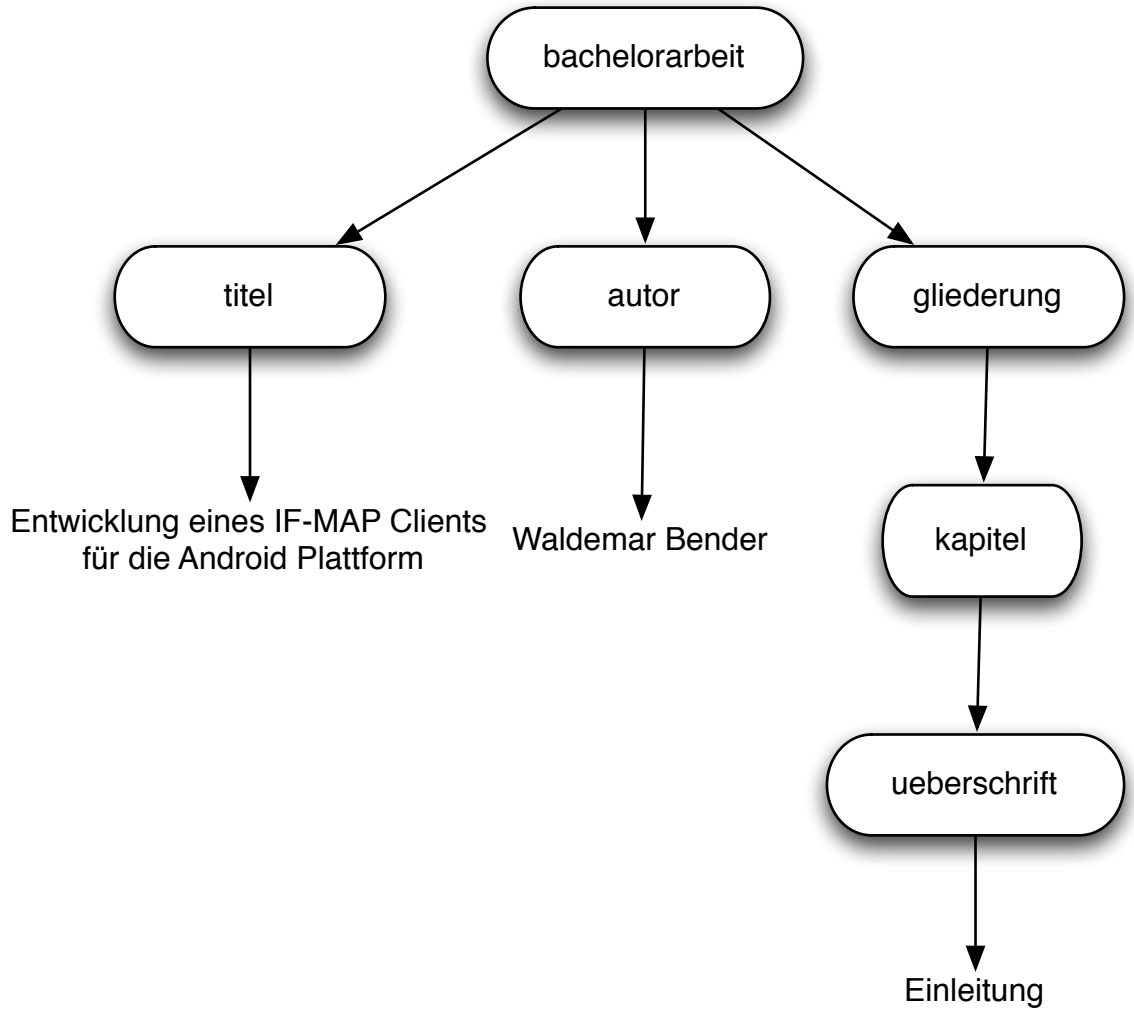

Abbildung 2.1: XML Nachricht in Baumstruktur dargestellt

2.1.2 Attribute und Werte

Es gibt zwei Möglichkeiten Daten an die jeweiligen Elemente anzuhängen. Die erste Möglichkeit ist das Anfügen als Attribut. Hierbei wird ein key-value-paar in das tag integriert. Ein Besipiel hierfür wäre in dem XML-Beispieldokument das Element "kapitel", das ein Attribut "nr" (key) mit Wert "1" (value) beinhaltet. Attribute werden immer dann benutzt, wenn sie keine eigentlichen Informationen, sondern Metadaten für die jeweiligen Elemente beschreiben. Die zweite Möglichkeit ist, die Daten direkt im Element abzulegen. Der Speicherort befindet sich dabei zwischen dem öffnenden tag und dem schließenden tag. In dem XML-Beispieldokument wäre das im Element "titel" der Wert "Entwicklung eines IF-MAP Clients fuer die Android Plattform".

2.1.3 Wohlgeformtheit

Damit ein XML Dokument wohlgeformt ist, müssen folgende Regeln eingehalten werden:

- Die XML-Version ist angegeben.
- Es ist genau ein Wurzelelement vorhanden.
- Für jeden Anfangs-Tag folgt ein Ende-Tag.
- Die Elemente sind korrekt geschachtelt.
- Alle Attributwerte stehen in Anführungsstrichen oder Apostrophen.
- Ein Attributname wird nicht mehrfach in einem Element verwendet.

[8] Durch diese Regeln ist eine einwandfreie und definierte Möglichkeit gegeben, ein solches Dokument zu parsen. Dadurch kann gewährleistet werden, dass das XML-Dokument auf jedem System gleich interpretiert wird.

2.1.4 Namensräume

Namensräume wurden eingeführt, um Namenskollisionen vorzubeugen. In einem XML Dokument kann es vorkommen, dass verschiedene Elemente den gleichen Namen besitzen. Namensräume können mit Paketstrukturen in Programmiersprachen verglichen werden. Ein Namensraum wird als Präfix dem Elementnamen vorangestellt. Außerdem muss der Namensraum als URI (uniform resource identifier) eindeutig definiert werden. Hier ein Beispiel:

```
<?xml version '1.0' encoding ='UTF-8'>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

</soapenv:Envelope>
```

Listing 2.2: XML Schema mit einem Namensraum

Das Element "Envelope" gehört dem Namensraum "soapenv" an. Der Namesraum wird durch die eindeutige URI "http://schemas.xmlsoap.org/soap/envelope/" identifiziert.

Weiterhin kann ein XML Dokument mehrere Namensräume besitzt. Es ist auch möglich, einen Namensraum als Standardnamensraum (default-namespace) festzulegen. Dabei wird bei der Namensraum-Deklaration, nach xmlns kein Name für den Namensraum definiert. Alle Elemente, die daraufhin folgen und denen kein Präfix voransteht, gehören automatisch zu diesem Standardnamensraum.

2.2 XML-Schema

XML-Schema ist eine Meta-Sprache, die dazu dient, den Sprachumfang für ein XML-Dokument zu definieren. Sie definiert, welche Elemente in einem Dokument vorkommen, wie oft diese vorkommen dürfen und welche Kind-Elemente diese beinhalten dürfen. Mit einem XML-Schema ist es möglich, eine Sprache zu definieren. Mit dieser Sprache gibt es eine klar definierte Abgrenzung, wie ein XML-Dokument auszusehen hat. Dadurch wird die Interoperabilität zwischen Systemen gewährleistet. Bei der Entwicklung von Anwendungen, die XML als Nachrichtenformat nutzen, ist es daraufhin möglich, genau zu definieren, was für Typen von Nachrichten empfangen und was für Typen von Nachrichten versendet werden dürfen. XML-Schmemata werden selber mit XML beschrieben.

2.2.1 Namensraum

Eine XML-Schema besitzt ihren eigenen Namensraum. Als Wurzelelement wird das Element "schema" verwendet.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
  targetNamespace="http://www.trustedcomputinggroup.org/2010/IFMAP/2">

</xsd:schema>
```

Listing 2.3: XML Schema mit einem Namensraum

Der Namensraum ist definiert als "xsd". Vor jedem XML-Schema Element muss das Präfix "xsd" vorangestellt werden. Als Standardnamensraum wird in diesem Beispiel "http://www.trustedcomputinggroup.org/2010/IFMAP/2" gewählt. Der Targetnamespace gibt das Ziel der Beschränkung an. Also gehören alle Elemente, die in diesem Schema definiert werden, dem Targetnamespace an. [8]

2.2.2 Elemente und Attribute deklarieren

Elemente deklariert man mit dem Schema-Element "element". Den Namen des Elementes definiert man als Attribut. Weiterhin kann der Typ des Wertes festgelegt werden.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
  targetNamespace="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
        <xsd:element name="Name" type="xsd:string" />
        <xsd:element name="Alter" type="xsd:decimal" />
</xsd:schema>
```

Es sind im oberen Beispiel die Elemente Name vom Typ "string" und Alter vom Typ "decimal" im Sprachraum definiert. Weiterhin ist es auch möglich, seine eigenen Typen zu erstellen. Ein Element kann auch weitere Elemente beinhalten. Eigene Elemente werden mit einem ComplexeType erzeugt:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
  targetNamespace="http://www.trustedcomputinggroup.org/2010/IFMAP/2">

  <xsd:element name="publish" type="PublishRequestType"/>

  <xsd:complexType name="PublishRequestType">
    <xsd:sequence>
        <xsd:element name="update" type="xsd:string"/>
        <xsd:element name="notify" type="xsd:string"/>
        <xsd:element name="delete" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

In diesem Beispiel existiert ein Element "publish". Dieses Element hat die drei Kindselemente "update", "notify" und "delete". Hierbei wird die Definition zwecks Übersicht in zwei Teile geteilt. Es ist jedoch auch möglich, den ComplexeType gleich im Element zu definieren. Das Element "sequence" erzwingt, dass alle Kindselemente in der angegebenen Reihenfolge existieren müssen. Es bestehen zwei weitere Möglichkeiten Elemente anzuordnen. Das Element "all" besagt, dass alle Elemente bestehen müssen, jedoch können diese in beliebiger Reihenfolge angeordnet werden. Um nur ein Element aus der Liste benutzen zu müssen, ist das Element "choice" vorhanden. Weiterhin ist es möglich, Wiederholungsangaben zu treffen.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
  targetNamespace="http://www.trustedcomputinggroup.org/2010/IFMAP/2">

  <xsd:element name="publish" type="PublishRequestType"/>

  <xsd:complexType name="PublishRequestType">
    <xsd:sequence>
      <xsd:choice minOccurs="1" maxOccurs="unbounded">
              <xsd:element name="update" type="xsd:string"/>
              <xsd:element name="notify" type="xsd:string"/>
              <xsd:element name="delete" type="xsd:string"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>

</xsd:schema>
```

In diesem Beispiel wäre es so, dass eines dieser Elemente mindestens einmal vorkommen muss. Jedoch dürfen die Elemente unbestimmt oft vorkommen. Die Deklaration

eines Attributes erfolgt genauso mit dem Element "attribute". Der Name wird, wie beim Element über das Attribut "name" gesetzt. Ebenso muss ein type für das Attribut festgelegt werden. Weiterhin ist zu beachten, dass Attributdeklarationen immer hinter den Elementdeklarationen stehen müssen. [8] Außerdem ist es möglich, für Attribute weitere Optionen zu setzen (z.b einen Standardwert, die Festlegung von Werten, Grenzwerte bei Zahlen usw.) .

2.3 SOAP

SOAP ist ein Nachrichtenprotokoll, das eine Kommunikation zwischen Anwendungen ermöglicht. Jedoch ist SOAP immer auf darunterliegende Transportprotokolle angewiesen. Das Transportprotokoll ist frei wählbar. In der Praxis wird, wie auch in dieser Arbeit, meist auf HTTP(S) zurückgegriffen. Eine SOAP-Nachricht wird immer in eine XML-Umgebung geschrieben.

2.3.1 Nachrichtenaufbau

Im Folgenden das Beispiel einer SOAP Nachricht:

```
<?xml version '1.0' encoding ='UTF-8'>
<soapenv:Envelope
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:ifmap="urn:trustedcomputinggroup.org:2010:IFMAP:2">
        <soapenv:Header>
        </soapenv:Header>

        <soapenv:Body>
                <ifmap:subscribe session-id="432"
                xmlns:ifmap="urn:trustedcomputinggroup.org:2010:IFMAP:2"
                xmlns:meta="urn:trustedcomputinggroup.org:2010:IFMAP-METADATA:2">
                        <update name="35" result-filter="meta:role"
                        match-links="meta:access-request-ip max-depth="3">
        ________________________<ip-address value="192.0.2.11" type="IPv4"/>
        ________________________</update>
        ________________</ifmap:subscribe>
        ________</soapenv:Body>
</soapenv:Envelope>
```

Listing 2.4: Beispiel einer SOAP Nachricht

Zunächst fällt die Aufteilung der SOAP-Nachricht in Header und Body auf. Der SOAP-Body wird verwendet, um die eigentlichen Nutzdaten zu transportieren. In einem SOAP-Header werden protokollspezifische Daten gespeichert. Insbesondere wenn zwischen Sender und Empfänger weitere Zwischenempfänger vorhanden sind (Intermediary's). Da in diesem Projekt der SOAP-Header nicht verwendet wird, wird hier nicht weiter auf diesen eingegangen.

2.3.2 WSDL

Die WSDL (Webservice Description Language) wird verwendet, um eine plattform- und programmiersprachenunabhängige Beschreibung von Webservices zu gewährleisten. Die

Metasprache WSDL kann Funktionen, Daten und Datentypen beschreiben.

Eine WSDL wird meistens benutzt, um zu überprüfen, welche Operationen vom Server überhaupt angeboten werden und wie sie anzusprechen sind. Vom Entwickler des Webservices sollte zuerst ein XML Schema festgelegt werden, um konkret zu beschreiben, welche Daten empfangen und versendet werden können. Daraufhin wird eine WSDL definiert, um die Schnittstellen zu beschreiben. Durch die Beschreibung wird festgelegt, wie der Web-Service angesprochen wird. Eine Reihe von Hilfsmitteln beherrschen daraufhin sowohl auf Server Seite wie auf Client Seite die Codegenerierung. Diese generiert eine WSDL zu Programmcode beliebiger Programmiersprache (vorausgesetzt, für die Programmiersprache existiert ein solcher Generator). Die Vorteile einer solchen Codegenerierung liegen auf der Hand. Es wird eine Interoperabilität zwischen Client und Server gewährleistet, da sie beide von derselben WSDL ableiten. Weiterhin wird dadurch die Entwicklungszeit stark verkürzt. Der Entwickler muss sich nicht mehr in die Feinheiten der XML Schemata einarbeiten, sondern kann sich ganz auf seine eigentliche Anwendung konzentrieren und die Arbeit zur Entwicklung einer Kommunikationsschicht dem generierten Programmcode überlassen. Diese Vorgehensweise wird als Contract-First-Ansatz bezeichnet.
Da der Server seine definierte WSDL meist auf der zu erreichenden Web Service URL ablegt, muss diese nicht einmal direkt vorliegen. Der Code kann aufgrund dessen einfach direkt mit Angabe der URL erzeugt werden. Obwohl es schon Suchmaschinen für Web Services aller Art gibt, ist ein System, das die gezielt benötigten Web Services sucht und so nutzt, dennoch schwer zu realisieren und wird in der Praxis so auch nicht verwendet. Es werden meist spezielle Anwendungen für spezielle Web Services entwickelt. Der folgende Ausschnitt zeigt den Aufbau mit anschließender Beschreibung einer WSDL.

```
<description targetNamespace="..." ...>
  <types> ... </types>

  <interface>
    <operation name"..." pattern="...">
      <input>...</input>
      <output>...</output>
      <infault>...</infault>
      <outfault>...</outfault>
    </operation>
    <fault>...</fault>
  </interface>

  <binding ...>
    <operation>
      <input>...</input>
      <output>...</output>
    </operation>
    <fault>...</fault>
  </binding>

  <service>
    <endpoint>...</endpoint>
  </service>
</description>
```

Listing 2.5: WSDL-Beispiel

Services werden durch sechs XML-Hauptelemente definiert:

- Datentypen (types): Definieren Datentypen, die zum Austausch der Nachrichten benutzt werden.
- Nachrichten (message): Die abstrakte Definitionen der übertragenen Daten.
- Schnittstellentypen (interface) : Beschreiben Operationen, die vom Web-Service angeboten werden
- Bindung (binding): Bestimmt konkretes Protokoll und Datenformat für die Arbeitsschritte und Nachrichten.
- Ports (endpoint): Spezifiziert eine Adresse für eine Bindung, also eine Kommunikationsschnittstelle.
- Services (service): Fassen eine Menge von verwandten Ports zusammen.

2.4 SSL/TLS

TLS (Transport Layer Security) ursprünglich SSL (Secure Socket Layer) ist ein Verschlüsselungsprotokoll zur sicheren Datenübertragung. TLS ermöglicht nicht nur eine Verschlüsselung der Daten (Vertraulichkeit und Integrität) sondern auch die Authentizität der zu kommunizierenden Punkte. Das Besondere hierbei ist, dass eine bidirektionale Authentifizierung möglich ist. Dies ist für dieses Projekt sehr wichtig. Das Protokoll ist in zwei Schichten unterteilt. TLS kann auf verschiedenen Transportprotokollen, wie z.B. TCP und UPD, arbeiten. Wiederum kann TLS unter verschiedensten Anwendungsprotokollen laufen, wie HTTP, POP3, telnet etc. . TLS teilt sich in zwei Schichten auf, zum Einen das TLS Record Protokoll und zum Anderen der TLS Handshake.

TLS Record Protokoll Das TLS Record Protokoll bietet hierbei die Verschlüsselung (Vertraulichkeit und Intergität). Dies wird durch eine symmetrische Verschlüsselung, die vom TLS Handshake ausgehandelt wurde, erreicht. Es ist auch möglich, dass eine null-encryption (keine Verschlüsselung) verhandelt wird. Hiervon sollte jedoch abgesehen werden. Dies kann evt. genutzt werden, falls die Vertraulichkeit und Integrität der Daten irrelevant sind und nur die Authentizität der Kommunikationspartner wichtig ist. Weiterhin gewährleistet das TLS Record Protokoll die Verlässlichkeit der Verbindung. [3] Dies wird durch Integritätsprüfungen erreicht. Die Intergritätsprüfungen werden mit der Hilfe von Message Authentication Codes (MAC) realisiert. MAC basiert auf dem System, die zu übertragene Nachricht zu hashen und den Hashwert daraufhin mit dem ausgehandelten Schlüssel zu verschlüssen. Dieser Wert wird "MAC-Wert" genannt. Der "MAC-Wert" wird daruafhin seperat von der eigentlichen Nachricht verschickt. Der Empfänger vergleicht daraufhin den "MAC-Wert" mit der eigentlichen Nachricht (Die Nachricht muss er dafür erst hashen). Wenn diese übereinstimmen, ist die erhaltende Nachricht integer. [16]

TLS Handshake Die TLS Handshake Schicht ermöglicht hingegen die bidirektionale Authentifizierung zwischen Client und Server. Weiterhin wird in dieser Schicht das Aushandeln des Verschlüsselungsalgorithmus vollzogen. Dieser Handshake ist die Initiierung einer TLS Verbindung. Um eine Authentifizierung zu erreichen, wird ein asymmetrisches Kryptosystem (Public-Key-Verfahren) verwendet. Als Algorithmus wird hierbei meist auf RSA (Anfangsbuchstaben der Entwickler) oder DSA (Digital Signature Algorithm) zurückgegriffen. Die Authentifizierung eines Endpunktes kann optional sein. Es muss sich jedoch mindestens ein Endpunkt authentifizieren. Bei den meisten Web-Anwendungen ist dies der Server. Weiterhin ist das Aushandeln des gemeinsamen geheimen Schlüssels sicher. Der Schlüssel kann durch Abhören der Leitung nicht berechnet werden. [3]

2.4.1 Truststore und Keystore

Trust- und Keystores haben nicht direkt etwas mit TLS zu tun. Sie sind da, um Zertifikate und Schlüssel sicher zu speichern. Der Keystore selber ist mit einem Passwort geschützt. Ein Keystore kann drei verschiedene Arten von Einträgen beinhalten.

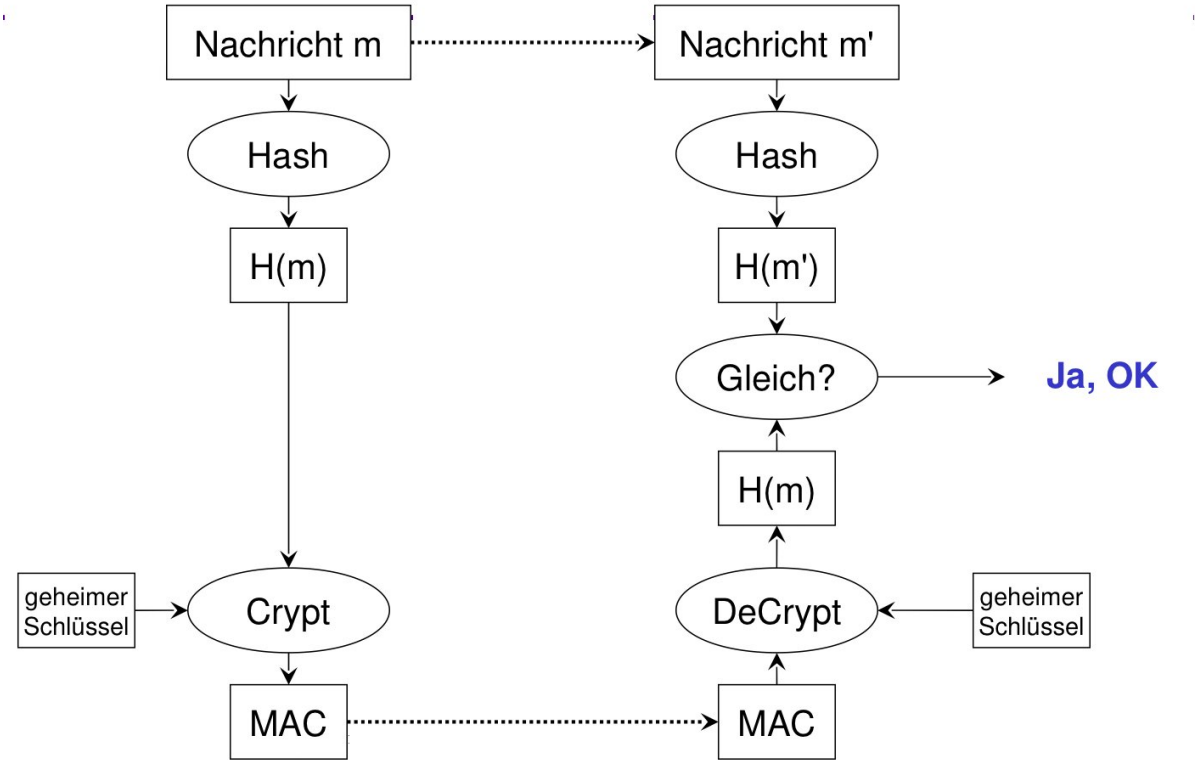

Abbildung 2.2: Ablauf von Message Authentication Code, Quelle: [16]

Ein Private-Key-Entry beinhaltet den eigenen privaten Schlüssel. Dieser ist in einem geschützten Format gespeichert. Weiterhin ist es möglich im Secret-Key-Entry weitere Schlüssel zu speichern. Dieser kann ebenfalls geschützt gespeichert werden. Der dritte Eintrag ist der Trusted-Certificate-Entry, dieser beinhaltet public-key Zertifikate.
Trusted-Certificate-Entrys beinhalten public-key Zertifikate, die vertrauenswürdig sind. [2]

Die Trennung zwischen Truststore und Keystore ist nicht unbedingt notwendig. Es ist auch möglich, alle Einträge (eigener private-key und vertrauenswürdiger Zertifikate) in einem Keystore zu speichern.

3 IF-MAP im TNC Umfeld

In diesem Kapitel wird aufgezeigt, was IF-MAP ist, wie IF-MAP funktioniert, was TNC ist und in welchen Zusammenhang IF-MAP und TNC zueinander stehen.

3.1 TNC

Um einen Client in einem Netzwerk zu authentifizieren, verwendet man den 802.1X Standard. Dieser Standard gewährleistet die Authentifizierung eines Clients, der Zugang zu einem Netzwerk erhalten möchte. Bei dem Standard wird also geprüft, ob der Client die Erlaubnis hat, sich mit diesem Netzwerk zu verbinden.
Die reine Authentifizierung eines Clients kann jedoch unzureichend sein. Wenn ein Client authentifiziert ist, kann nicht davon ausgegangen werden, dass der Client nicht evt. von einem Angreifer ferngesteuert wird. Wenn auf dem Client z.B. ein Trojaner, Virus oder Wurm läuft, kann dies zu einem Sicherheitsproblem im Netzwerk führen. Die Schadsoftware könnte sich dann im Netzwerk verbreiten und so das Netzwerk von innen angreifen. Um Vorgänge wie diese abzuwehren, wurde die Technologie NAC (Network Admission Control) entwickelt. Eine NAC prüft nicht nur, ob der Client berechtigt ist, sich mit dem Netzwerk zu verbinden, sondern prüft auch, ob der Client bestimmte Richtlinien einhält. Diese Richtlinien können z.B. die Aktualität des Virenscanners oder die Aktualität der Betriebssystemupdates beinhalten. Wenn der Client eines dieser Richtlinien nicht einhält, kann die NAC den Client entweder aus dem Netzwerk ausschließen oder ihn unter Quarantäne stellen bis er die Sicherheitsrichtlinien erfüllt.
TNC ist eine von der TCG spezifizierte NAC Lösung. Die Spezifikation ist frei verfügbar.

3.1.1 Architektur

Die Architektur, die in der Abbildung 3.1 zu sehen ist, wird in horizontale und vertikale Schichten aufgeteilt. Folgend wird die Architektur anhand der Schichten erläutert.
Wenn die Abbildung in vertikalen Schichten betrachtet wird, teilt sich die Architektur in die Bestandteile: AR, PEP und PDP. Die Schichten MAP und MAPC werden in den nächsten Kapiteln behandelt. Folgend die Erläuterung dieser Begriffe:

AR:
AR steht für Access Requestor. Ein Access Requestor ist ein Endgerät, welches versucht, in ein durch TNC geschütztes Netz zu gelangen.

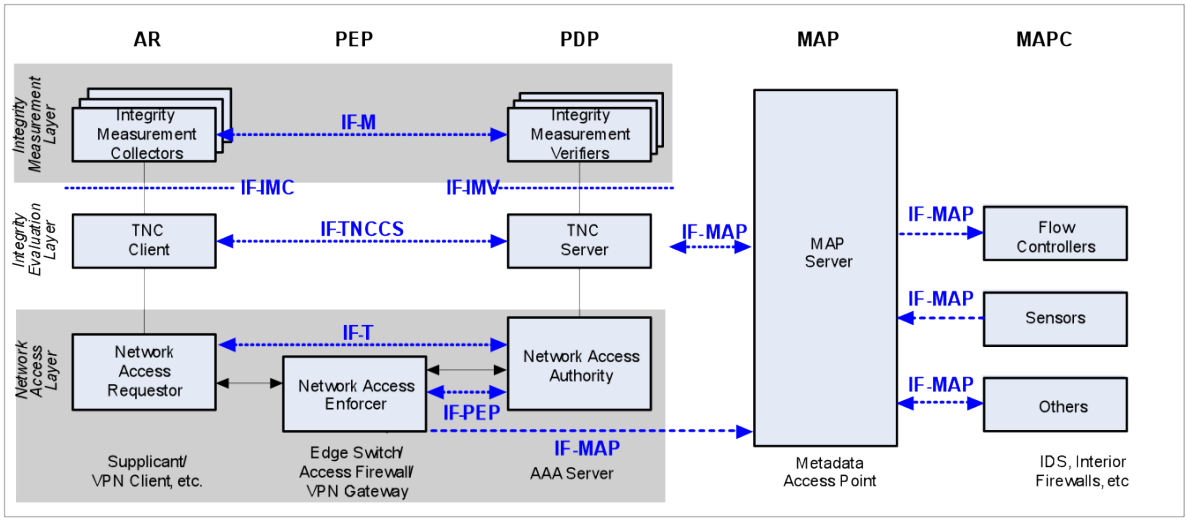

Abbildung 3.1: IF-MAP in der TNC-Architektur, Quelle: [12]

PDP:
PDP steht für Policy Decision Point. Die PDP entscheidet, ob ein AR mit dem Netzwerk verbinden darf oder nicht. Die PDP kommuniziert direkt mit der AR um zu überprüfen, ob der AR die gegebenen Richtlinien erfüllt, um zu diesem Netzwerk zu verbinden.

PEP:
PEP steht für Policy Enforcement Points. Die PEP führt aus, was die PDP entschieden hat. Wenn die PDP entschieden hat, dass der AR nicht in das Netzwerk darf, sperrt die PEP den Zugang für den AR. Ein PEP kann beispielsweise ein VPN Gateway, eine Access Firewall oder ein Switch sein.

Weiterhin teilt sich die Architektur in die folgenden drei horizontalen Schichten auf:

IML:
IML steht für Integrity Measurement Layer. In der IML tauschen AR und PDP Nachrichten über die Messung des Clients aus. Die IMC (Intergrity Mesurement Collectors) sind dabei Bestandteile des AR und die IMV (Integrity Measurement Verifiers) Bestandteile des PDP. Die IMC sammeln dabei Integritätsinformationen vom AR und die IMV verifizieren diese Informationen. [11]

IEL:
IEL steht für Integrity Evaluation Layer. In der IEL wird die Integrität des AR geprüft. Die Integrität des Clients wird mit Hilfe der Ergebnisse der IML evaluiert. Die Komponente TNC-Server entscheidet hierbei, ob der AR den Sicherheitsbestimmungen gerecht wird.

NAL:
NAL steht für Network Access Layer. Die NAL gewährleistet die gewöhnlichen Sicherheits- und Authentifzierungsstandards, wie z.B. 802.1X. Der NAR (Network Access Requestor) ist der authentifizierende Client, der NAE (Network Access Enforcer) stellt hierbei eine Hardwarekomponente dar. Die Hardwarekomponente könnte z.B. ein Switch, eine Firewall oder ein VPN Gateway sein. Die NAE kann den AR physikalisch aus dem Netz ausschließen oder weiterleiten. Dies passiert je nach der Erlaubnis des PDP's. Die NAA (Network Access Authority) authentifiziert den AR.

3.2 Funktionsweise von IF-MAP

Das Konzept IF-MAP beruht darauf, Statusinformationen von Endgeräten und Benutzern zu speichern. Die Arten dieser Statusinformationen können sehr verschieden sein. Beispiele hierfür wären: aktuelle IP- /MAC Adresse, Authentifizierungsstatus, Authorisierungsstatus, Standort-Koordinaten etc. . Die Statusinformationen werden vom MAP-Server (Metadata Access Point - Server) gespeichert. Der MAP-Server baut sich ein Datenmodell aus den erhaltenen Statusinformationen auf. Wenn der MAP-Server die Information bekommen hat, dass ein Benutzer A sich mit einem Endgerät mit der MAC-Adresse AA:23:DD:FF:AB:11 an einem 802.1x Switch authentifiziert hat, speichert er diese Information. Jetzt bekommt das Gerät die IP-Adresse 192.168.1.2 von einem DHCP-Server. Der DHCP-Server teilt dies dem MAP-Server mit. Im MAP-Server sind jetzt die Daten darüber gespeichert, dass ein Benutzer A sich authentifiziert, die IP-Adresse 192.168.1.2 zugewiesen bekommen hat und die MAC-Adresse AA:23:DD:FF:AB:11 besitzt. [12] So baut sich immer weiter ein Graph auf, der viele Statusinformationen zu mehreren Endgeräten beinhaltet. Dieses Modell wird im MAP-Server gespeichert und wird fortlaufend Datenmodell genannt. Der MAP-Server bekommt die Statusinformationen von verbundenen MAP-Clients. In dem eben aufgeführten Beispiel wäre der DHCP-Server und der 802.1X Switch ein MAP-Client. Die MAP-Clients haben verschiedene Möglichkeiten mit einem MAP-Server zu kommunizieren. Ein MAP-Client kann Statusinformationen auf dem Server veröffentlichen, Statusinformationen suchen und auch Statusinformationen abonnieren. Beim Abonnieren bekommt der MAP-Client die angeforderten Informationen, wenn sie verfügbar sind, also wenn ein anderer MAP-Client die abonnierten Statusinformationen veröffentlicht. Das Konzept des IF-MAP Protokolls kann als Ereignisdatenbank mit Abonnement-, Such und Veröffentlichungsfunktion beschrieben werden. Wie viele und welche Informationen die MAP-Clients genau veröffentlichen, hängt von den jeweiligen Implementierungen der MAP-Clients ab. Aufgrund dessen kann der MAP-Server die Korrektheit und die Aktualität der Statusinformationen nicht gewährleisten. Der MAP-Server speichert alle Informationen, die ihm von den MAP-Clients veröffentlicht werden. Die Aktualität und Korrektheit der Daten ist also einzig von den MAP-Clients abhängig. Weiterhin speichert der MAP-Server auch keine veralteten Statusinformationen. Falls eine neue Statusinformation beim MAP-Server eingeht, wird die alte von der neuen überschrieben.

3.2.1 Anwendungsfälle für IF-MAP

Statusinformationen, die ein MAP-Client veröffentlicht, sind Metadaten. Aufgrund dessen werden im Folgenden diese Statusinformationen Metadaten genannt. Die Grundfunktionen für den MAP-Server belaufen sich auf Veröffentlichen, Abbonieren und Suchen von Metadaten. Als Beispiel-MAP-Clients dienen die Komponenten DHCP-Server, Firewall, Sensor und ein mobiles Endgerät. Daraus bilden sich folgende Anwendungsfälle.

Ein Map-Client (DHCP-Server, Sensor, mobiles Endgerät) veröffenlicht Metadaten:

- Der DHCP-Server verteilt eine neue IP-Adresse und veröffentlicht daraufhin diese Metadaten.
- Ein Sensor hat eine einen Portscann von einem Endgerät erkannt und veröffentlicht diese Metadaten.
- Ein mobiles Endgerät veröffentlicht seinen aktuellen Standort.

Ein MAP-Client (mobiles Endgerät) sucht Metadaten zu einem Endgerät auf dem MAP-Server:

- Ein mobiles Endgerät fragt den MAP-Server nach dem Standort eines anderen mobilen Endgerätes ab.

Ein MAP-Client (Firewall) abonniert Metadaten zu einem Endgerät auf dem MAP-Server:

- Die Firewall abonniert Metadaten eines Endgerätes. Verstößt das Endgerät gegen Netzbestimmungen, blockiert die Firewall den Netzverkehr des Endgerätes.

3.3 IF-MAP in der TNC-Architektur

Die folgende Beschreibung geht auf die in Abbildung 3.1 gezeigten Verbindung von IF-MAP an die TNC-Architektur ein.

MAPC:
Ein MAPC (MAP-Client) ist eine Netzwerkkomponente, die IF-MAP nutzt, um mit dem MAP-Server zu kommunizieren. Diese Netzwerkkomponenten nutzen dabei die Funktionalitäten, die im Kapitel 3.2.1 erläutert wurden.

MAP:
Der MAP (Metadata Access Point) ist der zentrale Punkt in der IF-MAP Architektur. Der MAP wird von einem MAP-Server repräsentiert. Alle Daten, die von MAPC's veröffentlicht werden, werden im MAP verwaltet.

Die MAP-Clients können verschiedene Netzwerkkomponenten sein. Auf der Abbildung fungieren z.B IDS (Intrusion Detection System) und Firewall als MAP-Client. Der IDS würde bei erkannter unerlaubter Netzwerkaktivität (filesharing, portscanns etc.) einen Alarm auf dem MAP veröffentlichen. Die Firewall würde diese Metadaten abonnieren und den Angreifer aus dem Netz ausschließen. Aber auch der TNC Server ist ein MAPC. Der TNC Server wird bei der Anfrage eines AR diese Information zum MAP veröffentlichen. Es werden also alle Statusinformationen die Endgeräte betreffend von dem TNC-Server zum MAP veröffentlicht. Weiterhin bezieht der TNC-Server Statusinformationen vom MAP über die aktuelle Situation des Netzwerks und kann diese Informationen in seiner Entscheidung darüber, ob er einem AR den Zugang zum Netz gewährt oder nicht einfließen lassen. Der NAE ist ebenfalls mit dem MAP verbunden und veröffentlicht z.B. die MAC-Adresse des zugelassenen AR's.

3.4 Beispielszenario - IRON

Im Folgenden wird IF-MAP anhand eines Beispiel-Szenarios beschrieben. Dieses Szenario ist im Rahmen des Bachelor-Projekt IRON an der FH Hannover entstanden.

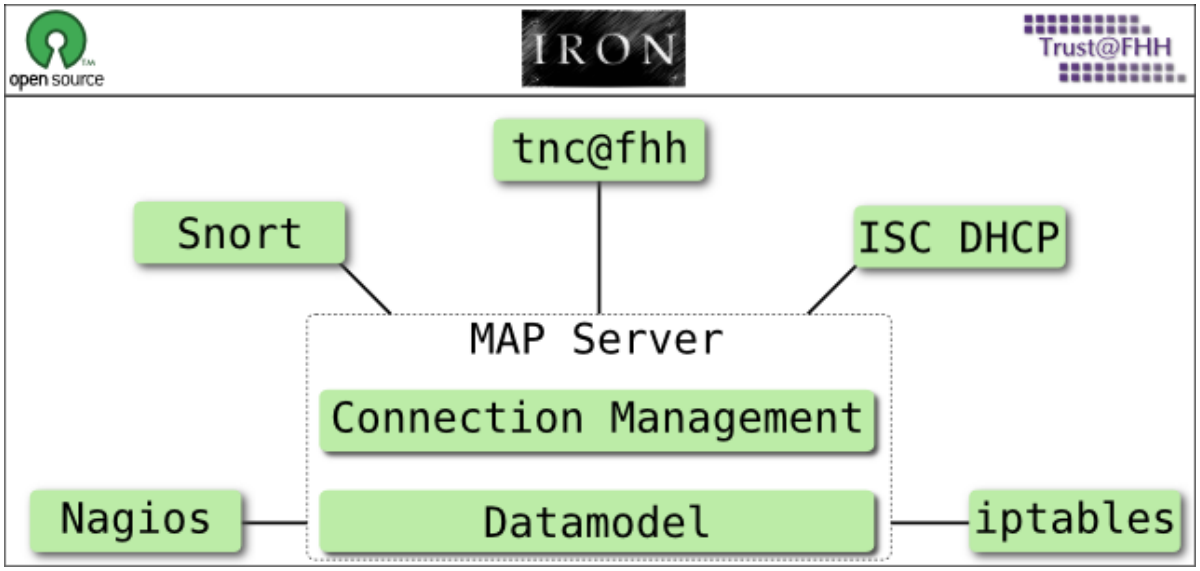

Abbildung 3.2: Architektur IF-MAP im IRON Projekt, Quelle: [7]

In diesem Beispiel agieren Nagios, Snort, tnc@fhh, ISC DHCP und iptables als MAP-Clients. Weiterhin existiert ein MAP-Server, der mit den Clients als Stern Topologie angeordnet ist. In typischen Netzwerken handelt jede Netzwerkkomponente autonom. Es findet keine Kommunikation und vor allem kein Informationsaustausch statt. Das Ziel des IRON Projektes war es, Netzwerkkomponenten auf Metadaten anderer Netzwerkkomponenten reagieren zu lassen.

Die Netzwerkkomponenten werden folgend kurz in ihrer eigentlichen Funktion und ihre Aufgabe im IRON Szenario vorgestellt:

Nagios:
Nagios dient der Überwachung von verschiedenen Netzwerkkomponenten in einer IT-Infrastruktur. Dabei kennt Nagios verschiedene Statusinformationen über die einzelnen Netzwerkkomponenten wie z.B. die angebotenen Dienste (SSH, FTP etc.), verfügbarer Festplattenspeicher, CPU Auslastung etc. . Dies dient dem Administrator des Netzwerkes als Hilfe, falls eine Netzwerkkomponente ausfällt oder bestimmte Dienste nicht mehr erreichbar sind. Die Netzwerkkomponenten werden mit Konfigurationsdateien definiert und konfiguriert.
Nagios erhält Zugang zu der MAP, um sofort festzustellen, welche Netzwerkkomponenten im Netz verfügbar sind. Bei stationären Netzwerkkomponenten ist eine statische Konfiguration leicht erstellbar. Bei mobilen Endgeräten ist dies leider nicht ohne weiteres mit statischen Konfigurationen realisierbar. Der Grund hierfür ist, dass mobile Endgeräte sich oft an verschiedenen Punkten im Netz anmelden (WLAN, LAN Port, VPN). Hierbei macht sich Nagios IF-MAP zu nutze. Nagios erkennt mit IF-MAP dynamisch, welche Netzwerkkomponenten sich im Netz anmelden und fügt diese dann automatisch in die Nagios-Konfiguration ein.

Snort:
Snort ist eine freie Implementierung eines Network Intrusion Detection System (NIDS). Ein NIDS zeichnet alle Pakete im Netzwerk auf, analysiert diese nach verdächtigen Aktivitäten und meldet diese dann. Diese Meldung kann z.B. in Form eines Eintrags in einer Log-Datei geschehen.
Die Motivation, Snort mit IF-MAP zu koppeln ist die, dass andere Netzwerkkomponenten auf die Meldungen von Snort sehr zeitnah reagieren können.

TNC@FHH:
TNC@FHH ist eine Open-Source Implementierung von TNC, die an der Fachhochschule Hannover entwickelt wurde. Die Motivation TNC@FFH an IF-MAP zu binden, ist bereits im Kapitel 3.3 erläutert worden.

ISC DHCP:
ISC DHCP ist eine DHCP Implementierung vom Internet Systems Consortium. ISC DHCP wurde an IF-MAP gebunden, um dem MAP-Server Auskunft über vergebene IP-Adressen zu erteilen. Weiterhin löscht ISC DHCP Metadaten über nicht mehr gültige IP-Adressen auf dem MAP-Server.

3.5 Datenmodell

Das Datenmodell speichert die von MAP-Clients veröffentlichten Metadaten. Durch das Veröffentlichen von Metadaten wird ein solches Datenmodell im MAP-Server aufgebaut. Das Datenmodell ist in Form eines Graphen gebildet. Auf der Abbildung 3.3 ist ein solches Datenmodell abgebildet.

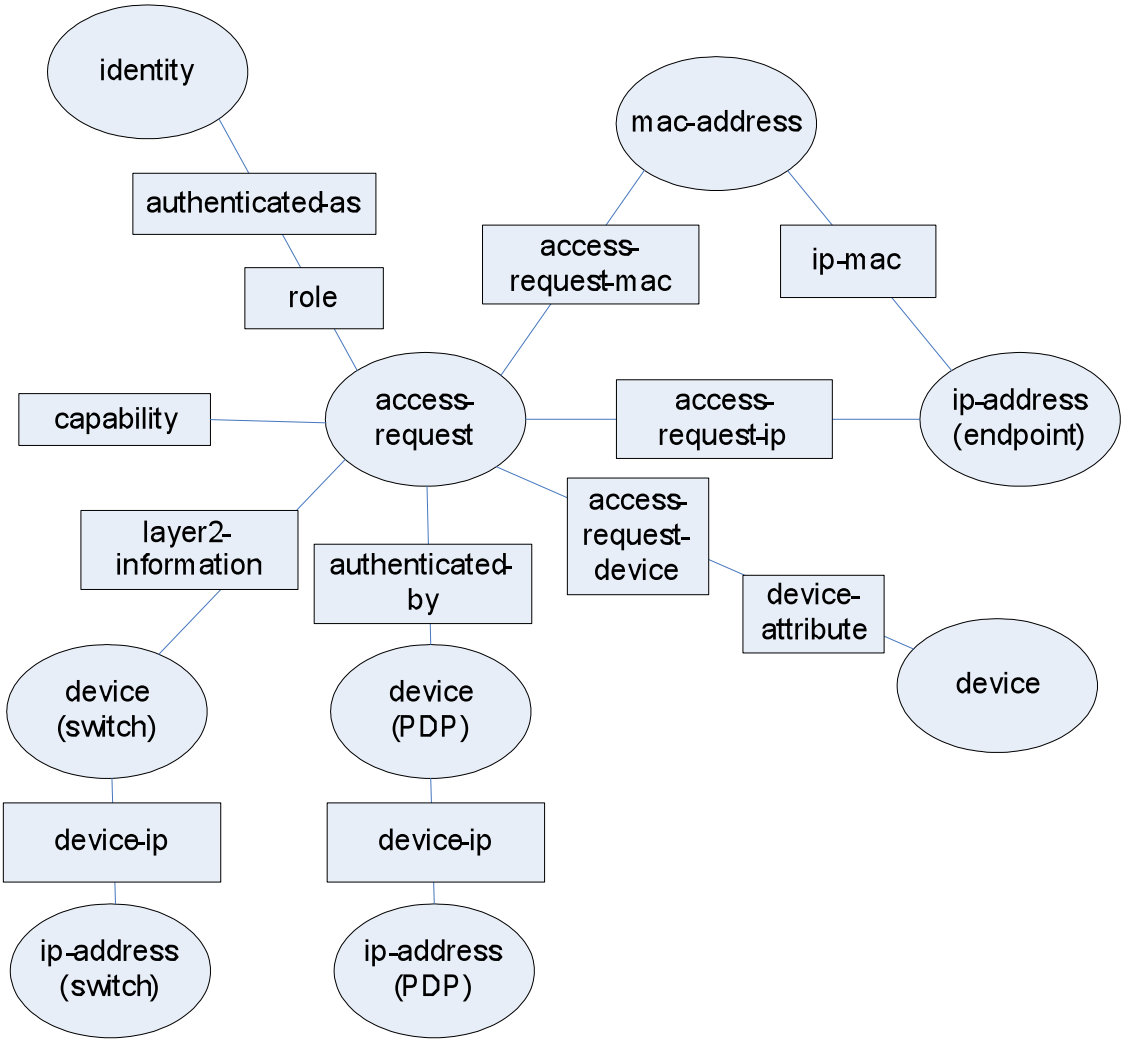

Abbildung 3.3: Datenmodell, Quelle : [12]

Identifer Um die genannten Metadaten einem Gerät zuordnen zu können, müssen Metadaten an Identifier gebunden werden. Ein Identifier kann unter anderem eine IP Adresse oder Mac-Adresse sein. Es können aber auch Identifier auf höherer Abstraktionebene benutzt werden, wie ein Benutzername, eine Email Adresse etc. .
Auf der Abbildung wird der Identifier mit einem Oval repräsentiert.

Link Ein Link beschreibt die Verbindung zwischen zwei Identifiern. Links werden immer von den MAP-Clients hergestellt. Ein Beispiel für eine solche Verbindung wäre, wenn ein DHCP-Server die Verbindung zwischen einer IP-Adresse und einer Mac-Adresse herstellt. Dies wäre ein ip-mac Link.
Auf der Abbildung wird ein Link mit Verbindung zwischen zwei Identifiern angezeigt.

Metadaten Die zu veröffentlichen Metadaten sind frei wählbar. Die Trusted Computing Group hat für das TNC Umfeld eine Spezifikation für Security-Metadaten veröffentlicht. Diese beschreibt Inhalte wie z.B. Layer2 Informationen, access-requests etc. . Es ist jedoch auch möglich, herstellerabhängige Metadaten zu definieren. Metadaten können an Identifier und Links angehängt werden. Sie sind auf der Abbildung als Vierecke gekennzeichnet.

3.6 Kommunikationsmodell

Ein MAP-Client kann bis zu zwei Kanäle zum MAP-Server aufbauen. Wenn der MAP-Client lediglich Daten veröffentlichen möchte, reicht ein SSRC (Synchron Send-Receive Channel). Dieser Kanal ist mindestens erforderlich, um eine Verbindung aufzubauen, Metadaten zu veröffentlichen und Metadaten zu suchen. Der zweite Kanal wird ARC (Asynchron Receive Channel) genannt. Dieser ist für abonnierte Metadaten von Identifiern notwendig. Die Daten vom MAP-Server werden also erst bei Verfügbarkeit über diesen Kanal übermittelt. Die einzelnen Operationen für die jeweiligen Kanäle werden im nachfolgenden Kapitel erläutert.

3.7 Operationen

Operationen zum Verbindungsaufbau: Die Operationen zum Verbindungsaufbau und Verbindungsabbau belaufen sich auf newSession, renewSession, poll und endSession. Verbindungen werden grundsätzlich immer von einem Client zum Server aufgebaut. Je nach Nachrichtentyp ist eine asynchrone sowie eine synchrone Antwort möglich. Die IF-MAP Spezifikation [12] sieht die Verbindungskanäle SSRC (Synchron Send-Receive Channel) und ARC (Asynchron Receive Channel) vor, wobei der ARC nur nötig ist, wenn Metadaten abonniert werden.

newSession Mit der newSession Operation wird eine IF-MAP Session zum Server aufgebaut. Hierbei findet keinerlei Authentifizierung statt. Die Authentifizierung erfolgt mit dem Aufbau der HTTPS Verbindung ausserhalb des IF-MAP Protokolls. Der Server antwortet mit einer eindeutigen Publisher ID und einer ebenfalls eindeutigen Session ID. Die Publisher ID repräsentiert den MAP-Client als solchen. Eine Publisher ID wird nur einmal vergeben. Der Client bekommt bei jedem Verbindungsaufbau dieselbe. Damit der MAP-Server den MAP-Client bei jedem Verbindungsaufbau wieder dieselbe Publisher ID vergeben kann, muss er ihn identifizieren können. Laut IF-MAP Spezifikation [12]

steht dem Entwickler des MAP-Servers frei, wie er dies implementiert. Die Publisher-ID darf lediglich nicht mit Hilfe von Zeit- oder Verbindungsinformationen generiert werden. Eine Session ID wird hingegen bei jedem Verbindungsaufbau neu generiert. Sie repräsentiert ausschließlich die aktive Sitzung.

renewSession Die Spezifikation gibt zwei Möglichkeiten vor, um eine Verbindung zum Server aufrechtzuerhalten. Es kann zum einen für jede Operation ein Socket aufgebaut, die Operation abgeschickt und nach Antwort vom Server wieder abgebaut werden. In diesem Fall lässt der Server die Verbindung 180 Sekunden lang valide. Daraufhin beendet er sie, ohne dies dem Client mitzuteilen. Um dies zu verhindern, muss der Client vor Ablauf der 180 Sekunden eine "renewSession" Operation an den Server schicken. Zum anderen besteht die Möglichkeit, die aufgebaute Verbindung offen zu lassen. Solange diese Verbindung bestehen bleibt, wird der Server die Verbindung nicht abbauen. Der Kanal für ARC Operationen muss mindestens so lange offen bleiben, bis der Server eine Antwort zur Anfrage geschickt hat. Der Server initiiert nie eine Verbindung zum Client.

poll Eine Poll-Operation gehört im Grunde genommen zu den Datenaustausch-Operationen, jedoch beeinflusst die Operation auch das Verbindungsmanagement. Eine Poll-Operation muss immer über einen gesonderten Kanal, den ARC, verschickt werden. Bei einem Poll ist es nicht vorhersehbar, wann der Server auf die Operation antwortet. Der Server antwortet erst bei Verfügbarkeit der abonnierten Metadaten.

endSession Eine endSession Operation beendet die bestehende Verbindung. Der Server antwortet mit einem endSessionResult.

Operationen zum Datenaustausch: Die Operationen zum Datenaustausch innerhalb des IF-MAP Protokolls belaufen sich auf publish, subscribe, search, purgePublish und poll.

Fehlerhafte Anfragen oder Serverfehler werden mit einem ErrorResult beantwortet. Außer poll verlaufen alle aufgelisteten Operationen synchron. Der Client stellt die Anfrage (request) an den Server, dieser antwortet (response) dem Client direkt. Bei Poll ist es nicht vorhersehbar, wann oder ob der Server antwortet. Wovon dies abhängt, wird bei der folgenden detaillierteren Beschreibung deutlich.

publish Publish veröffentlicht Metadaten auf den MAP-Server. Es gibt drei verschiedene Möglichkeiten ein Publish zu verschicken:

- Ein Publish-Update veröffentlicht Metadaten auf den Server. Sobald ein Client diese abonniert, werden sie dem Client veröffentlicht. Die Metadaten bleiben im Map-Server gültig, solange dieser läuft oder die Metadaten gelöscht werden.
- Ein Publish-Notify funktioniert wie ein Update, nur dass die Metadaten nicht gespeichert bleiben. Die Daten werden nur an die Clients veröffentlicht, die genau in diesem Augenblick den zugehörigen Identifier abonnieren.

- Ein Publish-Delete löscht alle Metadaten, die mit dem Identifier zusammenhängen.

subscribe Mit Subscribe ist eine Abonnierung von Metadaten möglich. Die Abonnierung läuft über einen Identifier. Es werden alle Metadaten abonniert, die diesem Identifier zugeordnet sind. Weiterhin kann ein Filter gesetzt werden, um ungewünschte Metadaten auszufiltern. Wird dieser Filter nicht gesetzt, werden alle Metadaten, die dem Identifier zugeordnet sind, abonniert.

poll Mit einem poll kann abgefragt werden, ob neue Daten für die abonnierten Identifier vorhanden sind. Ist dies der Fall, wird dieser Operation direkt mit einem Poll Result geantwortet. Ist dies nicht der Fall, antwortet der Server erst beim Eintreffen neuer Daten (von anderen Clients über publish).

search Mit einer search Anfrage ist es möglich, direkt Metadaten für einen Identifier abzufragen. Der Server beantwortet diese Anfrage direkt, also synchron. Der Unterschied zum Subscribe ist, dass eine search Anfrage eine einmalige Anfrage ist. Es werden keine Identifier abonniert.

purgePublish Mit einer purgePublish Operation ist es möglich, eine Anfrage zur Löschung der Metadaten eines publishers zu stellen. Ob diese Daten dann gelöscht werden, hängt von der Konfiguration des Servers ab. Wenn die Daten nicht gelöscht werden, teilt der Server dies dem Client mit einem "AccessDenied" mit.

3.8 Binding für IF-MAP

Die IF-MAP Operationen werden als Nutzdaten im SOAP-Body transportiert. Es ist ein XML-Schema von der TCG veröffentlicht, die den Aufbau der IF-MAP Anfragen spezifiziert. Zu den von der TCG entwickelten Metadaten existiert ebenfalls ein XML-Schema. Für genauere Infos zum Aufbau dieser Metadaten wird hier auf die jeweilige Spezifikation TNC IF MAP Metadata for Network Security [13] verwiesen. Weiterhin ist eine WSDL veröffentlicht, die die Schnittstellen zum Server beschreibt.

4 Mobile Endgeräte im IF-MAP Umfeld

Im Kapitel 3.2 wird beschrieben, dass IF-MAP zum Austausch von Metadaten benutzt wird. Diese Metadaten beziehen sich auf so genannte Identifier. Mit diesen Identifiern wiederum können Rückschlüsse auf das zugehörige Endgerät gezogen werden.
Im IRON-Szenario3.4 fällt auf, dass bis auf TNC@FHH alle MAP-Clients entweder nur Metadaten veröffentlichen oder nur abonnieren. Dabei wird deutlich, dass die Metadaten, die veröffentlicht werden, immer zu Endgeräten in der Umgebung der MAP-Clients gehören. Beispiele hierfür sind:

- Der DHCP-Server veröffentlicht Metadaten über Endgeräte, die eine DHCP-Anfrage starten.
- Snort veröffentlich Metadaten über Endgeräte, die versuchen untersagte Netzwerkaktivitäten zu starten.

Es wäre jedoch durchaus denkbar, dass ein Android MAP-Client Metadaten über sich selber veröffentlicht. Zusammengefasst bestehen drei denkbare Verwendungen, um einen MAP-Client für Android zu nutzen:

- Die Veröffentlichung von Metadaten in Bezug auf Endgeräte in der Umgebung
- Die Veröffentlichung von Metadaten in Bezug auf das Androidphone
- Die Abonnierung von Metadaten in Bezug auf Netzwerkendgeräte

Folgend werden Beispiele für die jeweiligen Verwendungen aufgeführt.

Veröffentlichung von Metadaten in Bezug auf Endgeräte in der Umgebung Es existieren Anwendungen für die Netzwerkanalyse (z.B. WLAN Analyse) auf Android. Das Android Endgerät könnte bei der WLAN Analyse Informationen wie die SSID, IP-Adresse, MAC-Adresse, Hersteller etc. von den jeweiligen WLAN Access-Points speichern. Da viele Android Endgeräte bereits GPS-Empfänger besitzen, könnte der Standort ebenfalls als Metadaten hinzugezogen werden. Diese Metadaten würde der MAP-Client auf einen MAP-Server veröffentlichen. Ein anderer MAP-Client kann mit den Metadaten, die von den MAP-Clients veröffentlicht wurden, eine Karte mit verfügbaren WLAN Access-Points erstellen. Diese Karte könnte durch mehrerer Android MAP-Clients stetig erweitert und aktualisiert werden.

Veröffentlichung von Metadaten in Bezug auf das Androidphone In einer aktuell bearbeitenden Master-Thesis [15] wird ein System entwickelt, das eine Integritätsmessung und -prüfung auf einem Android Engerät ermöglicht. Dieses System soll Manipulationen an Hard- und Software erkennen und das Endgerät aus dem Netzwerk ausschließen.
Angenommen, ein Unternehmen hat eine Anwendung XY entwickelt, die nur an bestimmten Orten benutzt werden darf. Um die Anwendung XY nutzen zu können, ist es nötig, auf unternehmensinterne Netzwerkressourcen(z.B. eine Datenbank) zuzugreifen. Diese Netzwerkressourcen sollen jedoch nur freigegeben werden, wenn das mobile Endgerät sich an bestimmten Orten aufhält. Das mobile Endgerät, auf dem die Anwendung XY läuft, ist durch Integritätsmessungen geschützt vor Manipulationen. Durch die Anbindung von Anwendung XY an einen MAP-Client wäre es möglich, bei jedem Start der der Anwendung eine aktuelle Standortinformation über IF-MAP zu veröffentlichen. Wiederum ist ein anderer MAP-Client zuständig für die Freigabe von Netzwerkressourcen. Dieser MAP-Client abonniert die Standortinformationen der jeweiligen Androidphones und entscheidet anhand dieser über die Zugriffserlaubnis der Androidphones auf die Netzwerkressourcen.

Die Abonnierung von Metadaten in Bezug auf Netzwerkendgeräte Weiterhin könnte ein MAP-Client auch eine sinnvolle Rolle im IRON-Szenario einnehmen. Es wäre denkbar, den MAP-Client als Monitoring-Anwendung zu nutzen. Mit dieser Anwendung wäre es möglich, wichtige Informationen über Endgeräte im Netzwerk abzurufen. Ein Netzwerkadministrator hätte an einer solchen Anwendung Bedarf. Beispielsweise könnte damit der Netzwerkadministrator umgehend über ein Sicherheitsproblem benachrichtigt werden.

Im folgenden Kapitel wird ein weiteres Szenario ausgearbeitet, in dem der Android MAP-Client als subscriber agiert.

4.1 Objekt -und Szenenveränderung

Es gibt an belebten Orten immer wieder Probleme mit besitzlosen Gegenständen. Dabei hat die Vergangenheit gezeigt, dass diese Tatsache immense Gefahren an so belebten Orten wie Flughäfen, Bahnhöfen oder großen Einkaufszentren mit sich bringt. Wichtig in einer solchen Situation ist die schnelle Erkennung solcher Gegenstände. Eine Erkennung alleine durch Kameras oder Hinweise von Menschen kann unzureichend sein. Das Ziel ist also eine Reduzierung dieser Reaktionszeit. Vermehrt sind in letzter Zeit Algorithmen zur Erkennung von Objekt- und Szenenveränderungen konzipiert worden. Ebenso zur Objektzählung oder Gesichtserkennung.
Wenn eine Person einen Koffer in sicherheitsrelevanten Orten einfach stehen lässt, kann die Kamera dies erkennen. Weiterhin wäre eine evt. Gesichtserkennung dieser Person möglich. Diese Daten müssen jedoch immer noch verteilt und darauf reagiert werden. Für diese Datenverteilung eignet sich IF-MAP optimal. Es wird von einer Anbindung

einer Kamera mit Objekt -und Szenenveränderungserkennung zu dem IF-MAP Protokoll ausgegangen. Für eine Anbindung an eine solche Kamera gibt es mehrere Möglichkeiten, welche abhängig von der genutzten Kamera sind. Wenn die Kamera eine offene API hat, eignet sich diese hervorragend, um die Kamera anzusprechen. Um den Bedarf an ein solches System zu unterstreichen, sei hier erwähnt, dass viele PC-Karten zum verarbeiten von Videomaterial zu erwerben sind, die eine Erkennung von Objekt- und Szenenveränderung realisieren. Viele bieten reine proprietäre Software zur Auswertung der Kameraergebnisse an. Manche bieten jedoch auch ein SDK an, um die verarbeiteten Kamerabilder anzusprechen. So z.B. die Modelle LE3000, LE6480E und LE5000 von der Marke AEON. Inwieweit die SDK den Zugriff auf diese Aufnahme-Karten zulässt, kann in dieser Arbeit nicht untersucht werden, da das SDK nur mit dem Erwerb einer dieser PC-Karten verfügbar ist. Die Kameras für dieses Szenario müssten also entweder auf eine solche PC-Karte zurückgreifen oder es müsste eine komplett eigene Verarbeitung der Kamerabilder entwickelt werden.

Abgesehen davon würde das System so aussehen, dass ein MAP Client diese Events von der Kamera erhält, in eine IF-MAP konforme Weise verarbeitet und an einen MAP-Server "published". Dieser Client würde also als publisher fungieren. An diesem Punkt zeigt IF-MAP seine größte Stärke. Die Verteilung vom MAP-Server aus, ist komplett variabel. Subscriber dieser Daten können sehr vielfältig und von den verschiedensten Systemen aus möglich sein.

Mögliche Beispiele wären:

- Sicherheitspersonal mit Android-basierten Endgeräten (z.B. einem Mobilfunkgerät), bekommen bei besitzerlosen Koffern sofort eine Nachricht.
- Eine Sicherheitszentrale mit einem IF-MAP Client, schaltet sofort auf die alarmierende Kamera.
- Ein IF-MAP Client, der an ein Persistenzsystem gebunden ist, das solche Ereignisse zur Beweissicherung aufzeichnet.

Die subscriber sind dabei auch völlig variabel einsetzbar. Es muss für einen weiteren subscriber lediglich die Zugangsvoraussetzung eingestellt werden. Eine Einstellung bezüglich der Datenverteilung muss am Server nicht vorgenommen werden.
Wie im Bild 4.1 wertet die Kameraverarbeitung die Bilder der Netzwerkkameras aus. Sobald eine Szene auftritt, die einen Alarm rechtfertigt, veröffentlicht die Kameraverarbeitung diese Metadaten. Der Identifier, an den diese Metadaten angebunden sind, ist in dem Fall die Kamera selbst. Somit ist auch die Identifizierung möglich, wo das Ereignis statt findet. Das Persistenzsystem, die Sicherheitszentrale und das Androidphone abonnieren Metadaten von den jeweiligen Kameras. Sobald die Kameraverarbeitung einen Alarm veröffentlicht, bekommen die eben genannten subscriber diesen Alarm durch die Metadaten mitgeteilt.
Hierbei reagiert jeder subscriber individuell. Das Persistenzsystem würde diese Daten

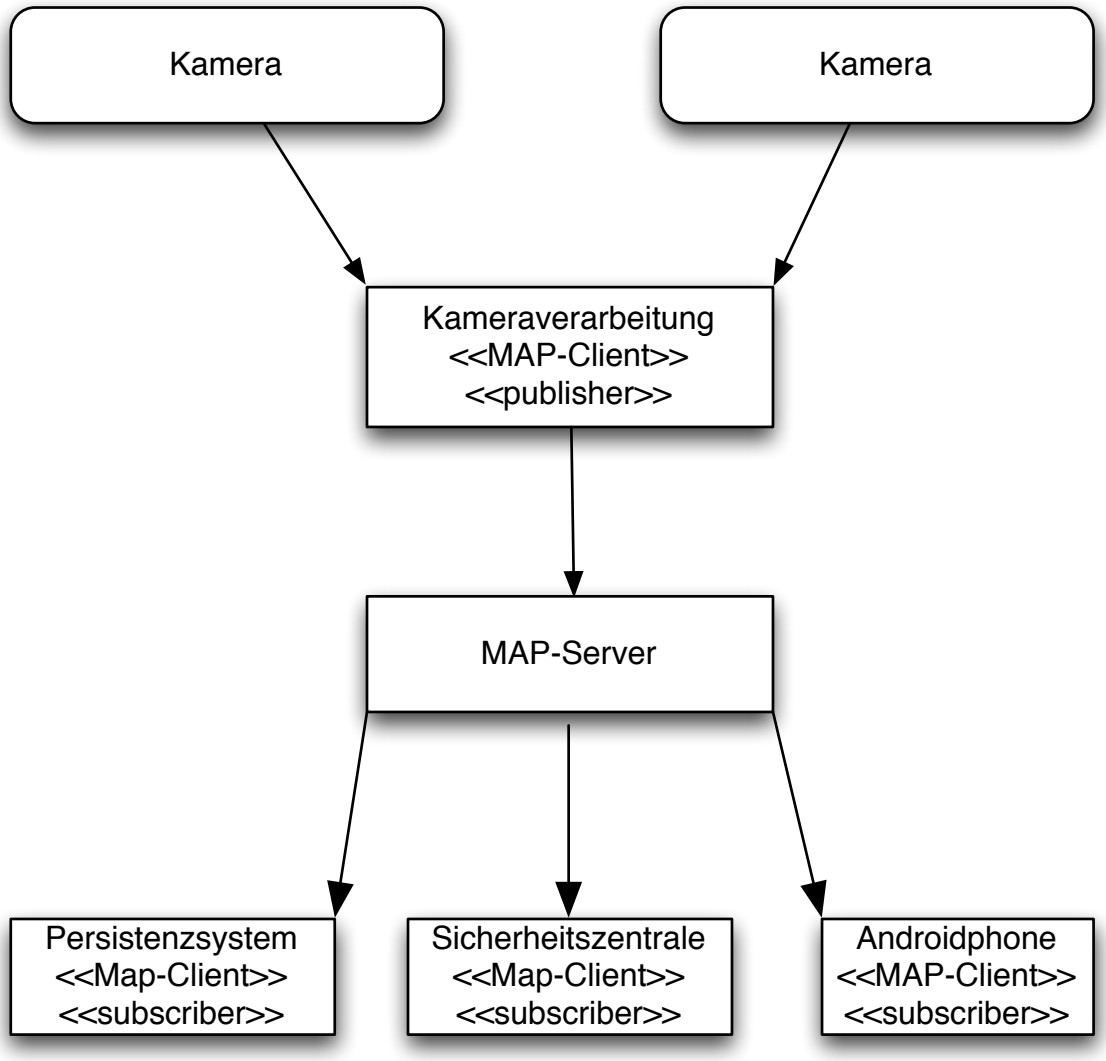

Abbildung 4.1: Architekturmodell - Szenen- und Objektveränderung

evt. zur Beweissicherung speichern. Die Sicherheitszentrale bekommt eine sofortige Schaltung auf die jeweilige Kamera. Die Androidphones würden direkt eine Mitteilung einblenden, wo der Alarm ausgelöst wurde und welche Art von Alarm es ist.

5 Anforderungsanalyse

In diesem Kapitel wird beschrieben, welche Prozesse erforderlich sind, um einen MAP-Client in die Android Umgebung zu integrieren. Weiterhin wird beschrieben, welche Funktionen dieser MAP-Client haben soll.

5.1 Systembeschreibung

5.1.1 Ausgangsbeschreibung

Eine große Hilfe bei dieser Arbeit war die Ausgangslage. Das Projekt IRON, welches in Kapitel 3.4 vorgestellt wurde, war zu Beginn dieser Arbeit im letzten Drittel seiner Bearbeitungszeit. Dies hatte den Vorteil, dass ein ständiges Testen in der IRON Projektumgebung möglich war. Eine Vorarbeit speziell für die Arbeit war jedoch nicht gegeben. Weiterhin gab es an Informationsquellen lediglich die IF-MAP Spezifikation [12] und diverse Literatur zur Android-Entwicklung [6] [17] [10].

5.1.2 Ziele und Erfolgskriterien

Das Ziel dieser Arbeit ist es, einen funktionsfähigen MAP-Client für die Android-Plattform zu implementieren. Dieser MAP-Client soll in der Lage sein, spezifikationskonform über das IF-MAP Protokoll mit einem MAP-Server zu kommunizieren. Die vorgestellten Szenarien aus Kapitel 4 verdeutlichen, dass ein MAP-Client auf Android viele Einsatzmöglichkeiten haben kann. Der zu entwickelnde MAP-Client soll nicht auf ein spezielles Szenario ausgelegt werden. Vielmehr soll das Ergebnis dieser Arbeit ein universeller zu bedienender MAP-Client werden, der für jedes erdenkliche Szenario genutzt werden kann. Das Ergebnis dieser Arbeit richtet sich an Entwickler, die eine Kommunikation über IF-MAP anstreben. Das Produkt dieser Arbeit muss also Schnittstellen anbieten, damit Anwendungsentwicklern eine Möglichkeit zur Benutzung dieser Komponente gegeben wird. Der MAP-Client dient als Kommunikationsschicht zwischen einer beliebigen Anwendung und einem MAP-Server. Der MAP-Client soll für jedes erdenkliche Szenario genutzt werden können. Dabei ist es auch wichtig, die Nutzung von beliebigen Metadaten zu erlauben. Diese Möglichkeit erlaubt es z.B. auch herstellerabhängige Metadaten zu verwenden.

5.2 Anforderungen

Folgend werden Anforderungen aufgelistet, die ein MAP-Client auf Android erfüllen muss, um den Zielen und Erfolgskriterien im Kapitel 5.1.2 gerecht zu werden. Die Anforderungen werden in funktionale, nichtfunktionale und technische unterteilt.

5.2.1 Funktionale Anforderungen

Die funktionalen Anforderungen spiegeln wieder, welche fachlichen Dienste vom System bereitgestellt werden können. [9] Dabei sind technische Details erstmal nicht von Belang.

Veröffentlichen von Metadaten: Der IF-MAP Client muss die Möglichkeit haben, Metadaten zu veröffentlichen.

Abonnieren von Metadaten: Es muss die Möglichkeit geben, Metadaten zu einem Endgerät oder Benutzer zu abonnieren.

Suchen von Metadaten: Das Suchen von Metadaten zu einem Endgerät oder Benutzer muss möglich sein.

Benutzung von beliebigen Metadaten: Der IF-MAP Client muss die Einbindung beliebiger Metadaten erlauben. Dies ist nötig, damit immer die passenden Metadaten für jegliche Szenarien entwickelt und diese dann auch mit diesem IF-MAP Client benutzt werden können.

Schnittstelle für Entwickler: Es muss eine Schnittstelle für Entwickler vorhanden sein. Mit dieser Schnittstelle ist es möglich, den IF-MAP Client zu verwenden. Diese Schnittstelle muss auch die einzige Möglichkeit sein, den IF-MAP Client zu bedienen. Hierbei sei nochmals erwähnt, dass der zu entwickelnde Client keine eigenständige Anwendung ist. Der Client ist eine Kommunikationsschicht, die von beliebigen Anwendungen zur Kommunikation mit IF-MAP Komponenten verwendet werden kann.

5.2.2 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen beschreiben Qualitätsansprüche.

Verschlüsselte Verbindung: Alle Verbindungen, die von dem IF-MAP Client ausgehen müssen durchgehend verschlüsselt sein. Es darf keine Möglichkeit geben, durch Mitschneiden des Datenverkehrs auf den Inhalt schließen zu können.

Authentifizierung: Bei jedem Verbindungsaufbau zu einen IF-MAP Server muss sich der Client authentifizieren. Auch der IF-MAP Server muss sich vor dem Client authentifizieren. Es muss eine komfortable Möglichkeit geben, einem MAP-Server die Authentifizierung zu erlauben bzw. zu verbieten. Diese Einstellmöglichkeit muss es getrennt vom eigentlichen Programmcode geben.

Resistent gegen Verbindungsabbrüche: Weiterhin ist mit einer begrenzten Bandbreite und mit Netzabbrüchen zu rechnen. Es sollte also vermieden werden, viele Daten zu übertragen. Auch ein ständiger Netzauf- und abbau kann bei der Anwendung zu Performanceproblemen führen. Der Client muss sinnvoll mit Verbindungsabbrüchen umgehen können.

Ressourcensparend und leichtgewichtig: Da Android meist auf mobilen Endgeräten läuft, müssen einige Besonderheiten beachtet werden. Auf mobilen Endgeräten muss immer mit Ressourcenknappheit gerechnet werden. Dies bezieht sich nicht nur auf den Speicher, sondern auch auf die CPU. Aufgrund dessen sollten leichtgewichtige Libraries eingesetzt werden.

Logische und ergonomische Schnittstelle für Entwickler: Der MAP-Client muss eine logische und einfach zu benutzende Schnittstelle für Entwickler bieten. Die soll durch eine einfache Dokumentation (z.B. JavaDoc) beschrieben und ohne weitere Literatur benutzbar sein.

5.2.3 Technische Anforderungen

Android als Betriebssystem Die Anwendung muss auf dem Android Betriebssystem lauffähig sein.

SOAP als Kommunikationsprotokoll Als Kommunikationsprotokoll ist SOAP zu verwenden. Dies muss verwendet werden, weil es die IF-MAP Spezifikation so vorsieht. Eine Implementierung mit einem anderen Kommunikationsprotokoll würde die Kommunikation mit einem MAP-Server nicht ermöglichen.

Interoperabilität Es müssen alle Standards aus der IF-MAP Spezifikation eingehalten werden, um die Interoperabilität zwischen MAP-Client und MAP-Server zu gewährleisten. Bei der Entwicklung muss auf die neuste IF-MAP Spezifikation zurückgegriffen werden.

6 Analyse - Android

In diesem Kapitel wird auf die Android Plattform eingegangen. Es wird ein Überblick zum Aufbau der Android-Architektur und zu den Besonderheiten bei der Anwendungsentwicklung auf der Android-Plattform gegeben. Weiterhin werden Möglichkeiten der Implementierung vorgestellt, analysiert und passend für diese Arbeit ausgewählt.

6.1 Android Plattform

6.1.1 Allgemein

Android ist ein Betriebssystem, das für mobile Endgeräte entwickelt wurde. Ein Linux-Kernel 2.6 bietet derzeit die Basis für das Betriebssystem. Die Architektur besteht aus dem Kernel (zuständig für Speicherverwaltung, Prozessverwaltung und Netzwerkkommunikation) und der Laufzeitumgebung Dalvik. Das erste release fand am 21.Oktober 2008 statt. [1] Google teilte am 23.Juni 2010 mit, dass 160.000 Android Mobiltelefone pro Tag aktiviert werden. Um die stetige Verbreitung noch weiter zu verdeutlichen, sei noch gesagt, dass es letzten Monat 100.000 und im Februar "nur" 60.000 pro Tag waren. [14] Diese Verbreitung verdeutlicht die Nachfrage und Attraktivität von Android.

6.1.2 Android Architektur

Die Android Architektur ist, wie auf der Abbildung 6.1 zu sehen, in verschiedene Schichten aufgeteilt. Folgend werden die einzelnen Schichten kurz erläutert, um einen Überblick zu Androids Architektur zu bieten.

Application Die einzelnen Programme wie Phone, Contacts etc., die auf der Schicht abgebildet sind, sind Anwendungen, die zur Grundausstattung von Android gehören. Auf dem Bild sind lediglich die wichtigsten aufgeführt.

Application Framework Die Application Framework Schicht beinhaltet Frameworks, die für jeden Android Anwendungsentwickler zugänglich sind. Als Beispiel sei hier View System genannt, das es leicht ermöglicht, grafische Oberflächen für Android zu erzeugen.

Libraries Die auf der Abbildung aufgeführten Libraries sind verschiedene C/C++ Libraries. Angesprochen werden sie über die eben erläuterten Application Frameworks. Diese Libraries sind ebenfalls von vornherein installiert.

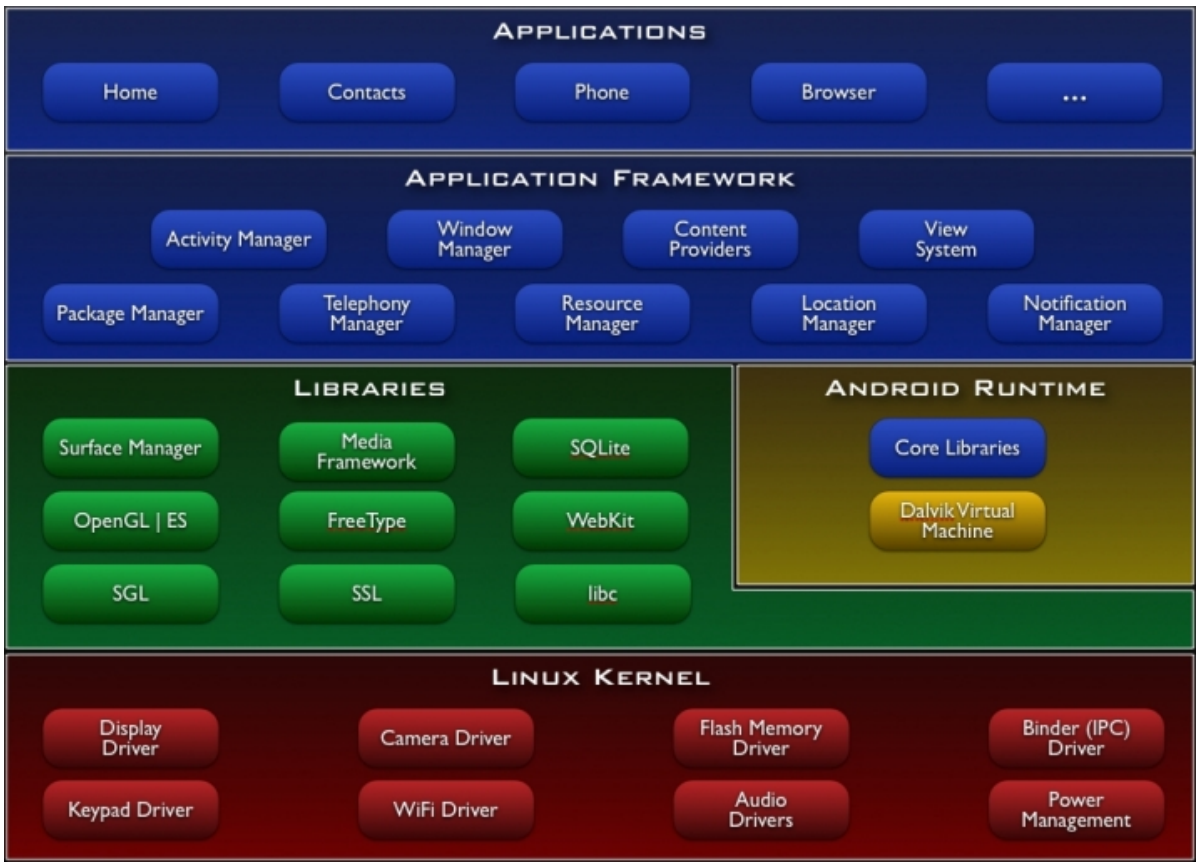

Abbildung 6.1: Android Architektur, Quelle: [5]

Android Runtime Die Android Runtime beinhaltet die Dalvik VM und die Java Core Libraries. Es wurden bei den Core Libraries einige Libraries weggelassen. Einige wurden weggelassen, weil sie bei Android keine Verwendung finden (z.B. Swing und AWT) und manche, wie z.B. JAXB-RI, wurden aus Performanzgründen weggelassen. Die Möglichkeit, ausgeschlossene Libraries wieder zu den Core-Libraries hinzuzufügen, wird vom Compiler unterbunden. Es ist jedoch ohnehin nicht zu empfehlen, solche Libraries wieder einzufügen, da sie gerade wegen den Performanzgründen ausgeschlossen wurden.

Linux Kernel Android basiert auf einen Linux 2.6 Kernel. Dieser Kernel arbeitet als Schicht zwischen der Hardware und dem restlichen Software Stack. [5]

6.1.3 Dalvik Laufzeitumgebung

Die Dalvik VM ist eine von Dan Bornstein entwickelte Java Virtual Machine. Sie ist entwickelt worden, um anspruchsvolle Java Anwendungen auf leistungsschwacher Hardware laufen zu lassen. Die Mindestvoraussetzung der Dalvik VM liegt bei einer CPU mit 500 MHz, 64 MB RAM Speicher und ein Linux Kernel, der ohne Swap auskommt. [10] Jede Android Anwendung bekommt ihren eigenen Prozess auf Betriebssystemebene und

somit auch seine eigene Dalvik Instanz. Weiterhin unterscheidet sich die Dalvik VM von den meisten JVM's darin, dass sie nicht als Kellerautomat sondern als Registermaschine arbeitet.

Eine Registermaschine bezieht ihre Operanden und Bytecodes aus virtuellen Registern. Die DalvikVM wurde so konzipiert, dass sie sich der ARM Prozessorarchitektur weitestgehend annähert. Es wird eine Abbildung von virtuellen zu Prozessorregistern erstellt. [10, S. 8]
Anwendungen werden im Dex Format, nicht im JAR Format, gespeichert. In einem Jar Container werden Class Dateien umkomprimiert angelegt. Jede dieser Class Dateien hat einen "heterogeneous constant pool", siehe Bild 6.2. Dieser "constant pool" beinhaltet Bezeichnungen von Methoden, Klassen etc. Diese Bezeichnungen werden in jeder Class neu beschrieben. Im Dex Format werden im Gegensatz solche Definitionen nur einmal für alle beinhaltenden Klassen getroffen. Dies zieht eine Speicherersparnis nach sich. Diese Speicherersparnis wirkt sich sehr positiv auf die begrenzten Ressourcen aus, die auf einem mobilen Endgerät bestehen. Das Bild 6.2 zeigt den Unterschied zwischen einem JAR und einem DEX Format deutlich auf.
[10, S. 8-9]

6.2 Android-Manifest

Die Android-Manifest Datei ist bei einer Anwendung von besonderer Bedeutung. Sie befindet sich in der Ordnerstruktur im Root. Die Android-Manifest Datei muss vom Entwickler der Anwendung geschrieben werden. In der Manifest Datei werden alle Komponenten der Anwendung und ihre Fähigkeiten beschrieben. Das Android-Manifest speichert Informationen über den Aufbau der Anwendung. [6] Diese ist mit XML beschrieben. Das Android-Manifest ist ein fester Bestandteil einer Anwendung. Es wird unter anderem der Einstiegspunkt zu einer Anwendung beschrieben. In den folgenden Kapiteln wird verdeutlicht, aus welchen Komponenten sich eine Android Anwendung zusammensetzt. Alle diese Komponenten müssen in dem Android Manifest beschrieben werden. Weiterhin werden Berechtigungen für die Anwendung beschrieben, denen der Benutzer beim Installieren der Anwendung zustimmen muss. Es kann als eine Art Deployment Deskriptor beschrieben werden.

6.3 Intents

Android Anwendungen sind vom Konzept her komponentenbasiert. Anwendungen sind in sich lose gekoppelt. Ausserdem besteht sehr oft eine Kommunikation zwischen verschiedenen Anwendungen sowie zwischen Anwendung und Betriebssystem selber. Intents sorgen für die Kommunikation zwischen diesen Komponenten. Sie stellen Verbindungen zwischen diesen her. Es bestehen zwei verschiedene Arten von Intents. Zum Einen die impliziten und zum Anderen die expliziteren Intents. Ein Intent wird immer von der

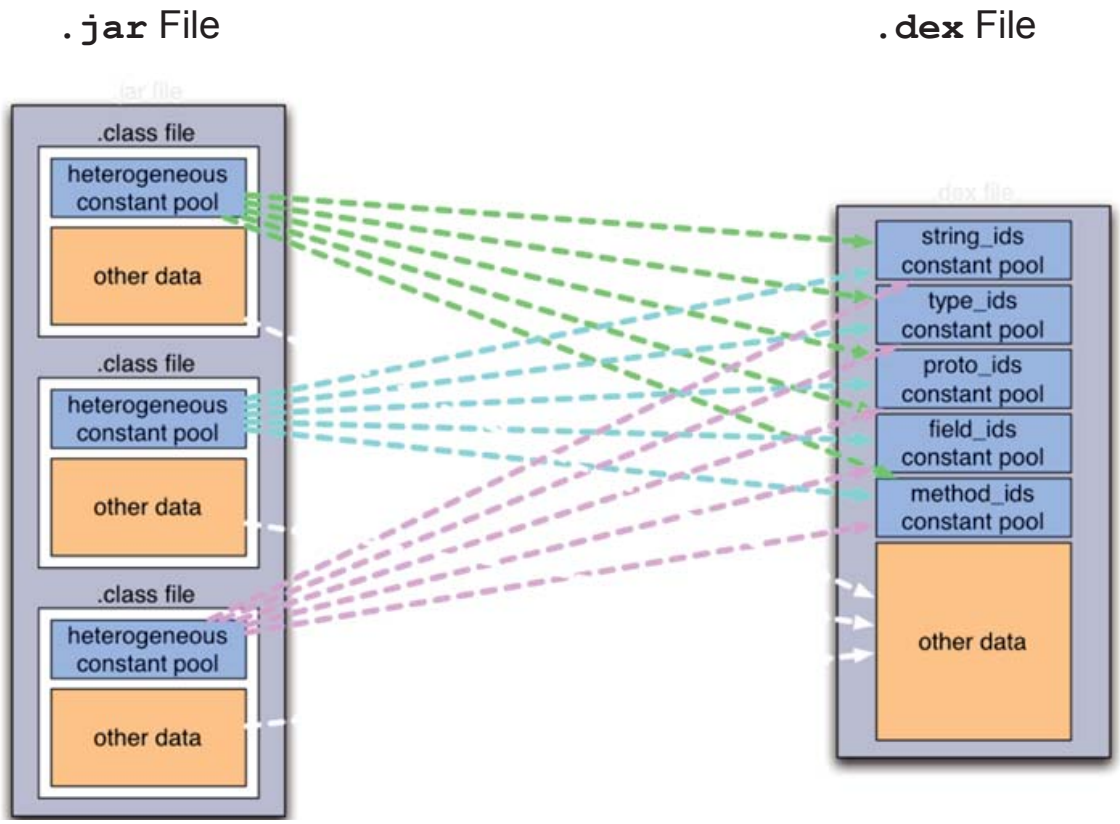

Abbildung 6.2: Unterschied JAR / DEX Headersegment, Quelle: [10, S. 9]

Komponente erstellt, die eine andere Komponente aufruft. Die aufrufende Komponente kennt den Aufrufer nicht und kann auch nicht individuell auf diesen reagieren. Die aufgerufene Komponente stellt lediglich Schnittstellen bereit, über die entweder Informationen abgerufen oder übergeben werden können.

6.3.1 Explizite Intents

Ein explizites Intent wird bei Verbindungen genutzt, wenn zur Programmierzeit die Verbindungskomponente bereits bekannt ist. Die Kopplung über ein explizites Intent wird oft bei Verbindungen innerhalb der Anwendung benutzt. Diese Anwendungen sind lose gekoppelt aufgebaut und werden durch diese Intents verbunden. Der Vorteil hierbei ist, dass der Austausch einer Komponente ohne Änderung im Programmcode vollzogen werden kann. Es muss lediglich der Intent ausgestauscht werden. Nachfolgend ein Codebeispiel, wie solch ein Intent erstellt wird.

```
Intent intent = new Intent
        (this, Mapclient.class);
```

Dem Intent werden hierbei die beiden Kompontenten im Konstruktor übergeben. Diese Möglichkeit der Verbindung stellt eine Unabhängigkeit der zu verbindenden Komponente dar. In der Komponente, hier im Beispiel "Mapclient.class", muss bei einer Verbindung mit einer anderen Komponente keinerlei Änderungen im Programmcode vorgenommen werden.

6.3.2 Implizite Intents

Bei impliziten Intents hingegen, ist die Komponente zu der verbunden wird, nicht zur Programmierzeit bekannt. Diese Art der Adressierung wird meist verwendet, um programmübergreifende Verbindungen zu gewährleisten. Es kann oft nicht sichergestellt werden, ob die Empfängerkomponente überhaupt auf dem Gerät installiert ist.

```
Intent intent = new Intent
        (Intent.ACTION_DIAL, Uri.parse("tel:(0228)1234567"));
startActivity(intent);
```

Listing 6.1: Quelle: [6]

In diesem Beispiel wird ein Intent erstellt, um einen Rufaufbau zu realisieren. Solch ein Intent kann beispielsweise von einer Adressbuch-Anwendung erstellt werden. Intent.ACTION DIAL dient hierbei als Intent-Bezeichner. Die Zielkomponente bestimmt selber, auf welche Bezeichner sie reagieren möchte. In diesem Beispiel würde die Anwendung reagieren, die für das Wählen von Rufnummern zuständig ist.

6.4 Activity und Service

Bei der Anwendungsentwicklung auf Android wird zwischen Activity und Service unterschieden. Im Folgenden wird darauf eingegangen, welche Rolle eine Activity und ein Service bei der Anwendungsentwicklung auf Android einnimmt. Weiterhin werden verschieden Arten von Services vorgestellt und darauf eingegangen, welcher dieser Services sinnvoll für dieses Projekt ist.

6.5 Activity

Eine Activity bereitet Daten auf und präsentiert sie auf dem Gerät. In einer Activity sollte möglichst keine Programmlogik vorhanden sein. Jede Anwendung besitzt eine Activity, die beim Start der Anwendung instanziert wird. Die Activty benutzt sogenannte Views, um Daten darzustellen. Views sind XML Dateien, in denen die einzelnen GUI-Komponenten deklarativ beschrieben werden. Es können, wie bei Java-Swing, verschiedene Layouts, wie z.B. Grid-Layout, LinearLayout etc. definiert werden. In der Beschreibung der View werden auch Elemente wie Buttons, Textfelder, Listen etc definiert. Es ist jedoch auch möglich, eine View im Programmcode zu definieren. Dieses würde jedoch die Wartbarkeit stark behindern. Um die View zu ändern bedarf es bei der deklarativen

Methode nur einer Änderung in der XMl Datei. Bei einer Gestaltung im Programmcode hingegen, muss die Stelle der Definition im Code gefunden, verstanden und daraufhin geändert werden. Wenn zusätzlich noch Abhängigkeiten zu anderen Code-Abschnitten oder Klassen bestehen, wird die Wartbarkeit noch viel weiter eingeschränkt. Ein weiter Vorteil bei der Definition von Views in XML-Dateien ist die Verweisung von Textbausteinen zu der Datei String.XML. In der String.XML ist es möglich, Texte, die in Views verwendet werden, zu definieren. Diese werden dann anhand ihres frei wählbaren Namens in der View aufgerufen. Dieses ermöglicht eine leichte Änderung der Programmsprache. Bei der Änderung der Sprache würde ein Austauschen der String.XML ausreichen.

6.6 Services

Jede Anwendung, die am Android Endgerät gestartet wird, ist ein Prozess. Jeder dieser Prozesse hat auch eine Process ID, kurz PID. Der gestartete Prozess hat immer einen so genannten UI-Thread (User Interface Thread). Dieser Thread sorgt für die Darstellung und kümmert sich um User Ein- und Ausgaben. Die in der Android-Manifest angegebene Start-Activity läuft in dem UI-Thread. Um länger laufende Arbeit zu verrichten, sollte in diesem Fall immer ein neuer Thread gestartet werden. Falls die Anwendung nämlich fünf Sekunden nach Benutzereingabe nicht reagiert, bekommt der Benutzer eine ANR (Application not responding) Meldung angezeigt. Ein Thread überdauert nie einen Prozess. Wenn dieser Thread jedoch länger als die eigentliche Anwendung laufen soll, muss hierfür ein eigener Prozess gestartet werden. Dies ist meist bei Prozessen wichtig, die von verschiedenen Anwendungen benutzt werden. Beispielsweise ist es sinnvoll, abspielende Musik in einem eigenen Prozess laufen zu lassen. Somit können verschiedene Anwendungen darauf zugreifen, der Musikprozess jedoch ist nicht abhängig von deren Lebenszyklus. [6]
Nochmal zusammengefasst betrachtet, benutzt man also für Hintergrundoperationen, die von keiner anderen Anwendung abhängig sind, eigene Prozesse und für Operationen, die abhängig von der gestarteten Anwendung sein können, aber eine lange Rechenzeit benötigen, Threads.
In Android wird dies mit den schon erwähnten Services realisiert.

6.6.1 Local Service

Ein Local Service läuft immer im gleichen Prozess wie die Anwendung, die ihn gestartet hat. Sein Lebenszyklus ist höchstens so lang, wie der der Anwendung selbst. Ein Local Service wird jedoch nicht nur verwendet, um einen neuen Thread zu starten. Dafür könnte man auch einen einfachen Java Thread nutzen. Man muss einen Service als eigenständigen Programmbaustein betrachten. Ein Service kann und sollte auch komplett unabhängig und eigentständig von einer Activity implementiert werden. Durch dieses Konzept ensteht eine lose Kopplung zwischen Logik- und Präsentationsschicht.

Für die Kommunikation zwischen einer Activity und einem Local Service gibt es kla-

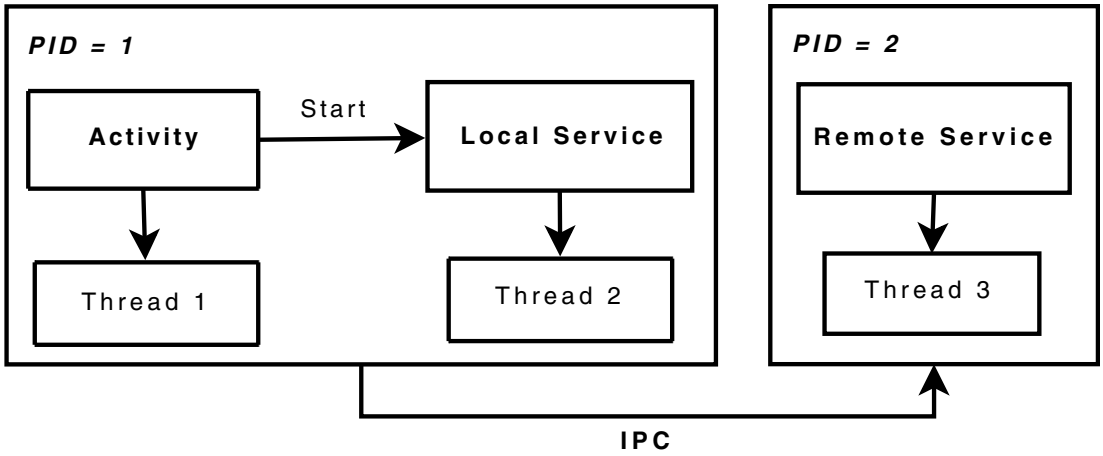

Abbildung 6.3: Vergleich: Local Service vs. Remote Service, Quelle: [6]

re Richtlinien. Dies hat den großen Vorteil, dass keine eigene Schichtentrennung oder Fassadepattern realisiert werden muss. Dies wiederum führt dazu, dass der Entwickler einer Activity sofort weiß, wie er einen Service anzusprechen hat, ohne sich in die Details des Services einarbeiten zu müssen. Um einen Local Service zu erstellen, wird wie immer der Eintrag des Local Service in der Android Manifest erstellt:

```
<service android:name=".package.NameDesService" />
```

Daraufhin muss ein sinnvoller Einstiegspunkt für den Service gefunden werden. Meist wird hierfür einfach eine neue Klasse erzeugt, die als Verbindung zwischen der Activity und dem Service besteht. Diese selbst erzeugte Klasse muss von der Klasse "Service" ableiten. Daraufhin müssen folgende Methoden implementiert werden:

- onCreate()
- onDestroy()
- onBind(Intent intent)

Die Methode onCreate() wird beim Erzeugen des Services aufgerufen. Sie kann als eine Art Konstruktor des Services gesehen werden. Alle Aufgaben, die bei der Benutzung des Services erledigt werden müssen, sollten in dieser Methode passieren. Die Methode onDestroy() wird bei Beendigung des Services aufgerufen. Hier können beispielsweise offene Sockets oder Netzwerkwerksessions ordungsgemäß geschlossen werden. Eine besondere Rolle spielt die onBind() Methode. Sie wird bei der Bindung einer Activity an einen Service aufgerufen.

Wie dies funktioniert, wird anhand eines Code-Beispiels erklärt:

```
public class EigenerLocalService extends Service {

        private final EigenerLocalBinder mLocalBinder = new EigenerLocalBinder();

    public class EigenerLocalBinder extends Binder {
        public EigenerLocalService getService () {
                return EigenerLocalService.this;
        }

                public ObjektFuerActivity getDaten () {
                        return new ObjetFuerActivity();
                }
    }

        @Override
        public IBinder onBind(Intent arg0) {
                return mLocalBinder;
        }

        @Override
        public void onCreate() {}

        @Override
        public void onDestroy() {}
}
```

Listing 6.2: Beispielimplementierung eines Local Service

Wie bereits erwähnt, wird bei der Verbindung zum Service die Methode onBind() aufgerufen. Diese enthält als Rückgabewert ein Objekt vom Typ IBinder. Als innere Klasse wurde hierbei die "EigenerLocalBinder" Klasse definiert, die von Binder ableitet. In dieser Klasse können die Schnittstellen für die verbunde Activity definiert werden. Man kann also genau bestimmen, welche Objekte der Activity angeboten werden und welche vor der Activity gekapselt werden.

Auf der Activity Seite wird der Service folgendermaßen aufgerufen:

```
...
private EigenerLocalService localService;
private EigenerLocalBinder localBinders;

        private ServiceConnection localServiceConnection =
                new ServiceConnection() {
                        @Override
                        public void onServiceDisconnected(ComponentName arg0) {}

                        @Override
                        public void onServiceConnected(ComponentName arg0, IBinder arg1) {
                                EigenerLocalBinder = (EigenerLocalBinder) arg1;
                                localService = ((EigenerLocalBinder) arg1).getService();
                        }
        };

        private void connectService () {
                Intent intent = new Intent (getApplicationContext(), localService.class);
                bindService(intent, localServiceConnection, Context.BIND_AUTO_CREATE);
        }
...
```

Listing 6.3: Beispielimplementierung eines Local Service

Mit dem Aufruf der connectService() Methode wird ein Intent zwischen der Activity (getApplicationContext ()) und dem Local Service (localService.class) hergestellt. Die bindService Methode verbindet letztendlich mit dem LocalService. Das Objekt localServiceConnection vom Typ ServiceConnection repräsentiert hierbei die Verbindung zum Local Service. Wenn die Verbindung hergestellt ist, wird in dem Objekt die onServiceConnected() Methode aufgerufen und der Binder des Local Service übergeben, worüber dann die freigegebenen Schnittstellen benutzt werden können.

6.6.2 Remote Service

Der Remote Service läuft immer in einem eigenen Prozess. Sein Lebenszyklus überdauert die Anwendung von der er gestartet wird. Die Kommunikation mit einem RemoteService gestaltet sich schwieriger im Vergleich zum Local Service. Der Unterschied besteht primär darin, dass eine Activity mit einem Local Service auf den gleichen Speicherbereich zugreift. Es kann bei einem Local Service also auch mit direkten Referenzen gearbeitet werden. Da ein Remote Service jedoch als eigener Prozess läuft, hat dieser auch seinen eigenen Speicherbereich. Um also mit einem Remote Service kommunizieren zu können, muss ein prozessübergreifender Datenaustausch stattfinden. Dieser findet mit Interprozesskommunikation (engl. inter-process communication IPC) auf Betriebssystemebene statt, siehe Abbildung 6.3. Beim Datenaustausch werden alle Objekte im Zuge der Serialisierung auf primitive Datentypen zerlegt und daraufhin zum Service geschickt. Für die De- bzw. Serialisierung stellt Google eine Variante zur Code Generierung zur Verfügung. Der generierte Code übernimmt die Serialisierung. Daraufhin ist es möglich, Methoden über Interprozesskommunikation aufzurufen. Für die Code Generierung müssen erstmal die Schnittstellen für den Remote Service definiert werden. Dies wird mit Hilfe der IPS (Interface Description Language) gemacht. IPS ist eine deklarative Sprache zur Beschreibung von Schnittstellen einer Software Komponente. [4] Vergleichbar ist diese mit der in Kapitel 2.3.2 beschriebenen Sprache WSDL. Android benutzt hierfür jedoch eine leicht abgeänderte Version von IDL, genannt AIDL (Android - IDL). Folgend ein Beispiel:

```
interface OwnRemoteService {
        void getPerson(inout Person person);
        void setPerson(in Person person);
        void setAlter(int alter);
}
```

Listing 6.4: AIDL Beispiel

Der Aufbau einer AIDL ähnelt sehr stark der eines Java Interfaces. Lediglich die Bezeichner "inout" und "in" sind in der Java Sprache nicht vorhanden. Es existieren insgesamt drei solcher Bezeichner:

- **in** bezeichnet, dass der Parameter nur zur Übergabe vorhanden ist. Die Variable kann beim Empfänger beliebig verändert werden. Der Absender bekommt die Veränderungen nicht mitgeteilt.

- **inout** simuliert eine Art "Call by Reference". Wird das Objekt beim Empfänger verändert, verändert sich das Objekt auch beim Absender. Auch hier wird über IPC kommuniziert. Es wird bei jeder Änderung eine De- und Serialisierung vorgenommen, um die veränderten Daten zwischen den Prozessen auszutauschen. Inout ist aufgrund dessen sehr rechenintensiv und sollte nur benutzt werden wenn es wirklich benötigt wird.
- **out** bezeichnet eine Variable, die nur beim Absender benötig wird. Es wird lediglich der Typ angegeben. Das Objekt wird beim Empfänger erzeugt und zurückgegeben. Der Unterschied zum inout ist hierbei, dass wenn der Absender das Objekt im Nachhinein ändert, die Änderung nicht übertragen wird. Dies wird für kopierte Rückgabewerte benutzt.

6.6.3 Bewertung

Der Hauptgrund für die Implementierung einer Anwendung als Remote Service ist, dass die Anwendung von verschiedenen Activities benutzt werden kann. Der Remote Service ist also unabhängig von den steuernden Activities. Dies ist bei einem Musikabspielprogramm oder einer Anwendung, die Kontakte speichert, wichtig. Die Anwendung die Adressen speichert, kann z.B. von einer Telefon Anwendung, einer Navigationssoftware oder auch von einer Anwendung zum SMS Versand benutzt werden.
Im Bezug auf das entworfene Szenario aus Kapitel 4.1 wird der MAP-Client für das Mobilfunkgerät des Sicherheitspersonals benutzt, um Daten von der Kameraverarbeitung zu empfangen. Wenn ein Ereignis auftritt, das die Aufmerksamkeit des Sicherheitspersonals verlangt, macht das Mobilfunkgerät z.B. mit einer Push Notifikation auf sich aufmerksam. Daraufhin öffnet sich eine grafische Oberfläche, die den Grund des Alarms erläutert.
Für das Szenario wäre also mehr als eine Activity, die auf den MAP-Client zugreift, nicht notwendig. Aber auch abgesehen von dem Szenario erscheint eine Steuerung und Anzeige von mehreren Activities auf den MAP-Client nicht sinnvoll. Eine Anwendung, die das MAP-Protokoll zur Kommunikation mit einem MAP-Server benutzt, hat aus Erfahrung einer dieser zwei verschiedene Verwendungen:

1. Ereignisse verarbeiten, relevante Daten auswählen und diese über das MAP-Protokoll propagieren.
2. Ereignisse empfangen, die andere MAP-Clients propagiert haben, und auf diese Ereignisse reagieren.

Folgend werden anhand des Szenarios aus Kapitel 4.1 und des IRON-Szenarios aus Kapitel 3.4 die MAP-Clients nach diesen Verwendungen kategorisiert.

Szanrio: Objekt- und Szenenveränderung

- Kameraverarbeitung verfährt nach Typ 1. Die Kameraverarbeitung bemerkt sicherheitskritische Vorgänge (z.B. stehen gelassener Koffer) und propagiert diesen Alarm zum MAP-Server.

- Persitenzsystem verfährt nach Typ 2. Das Persistenzsystem speichert Ereignisse, die die Kameraverarbeitung veröffentlich hat, zur Beweissicherung.
- Sicherheitszentrale verfährt nach Typ 2. Die Sicherheitszentrale bekommt Meldung über Art und Ort der Meldung.
- Androidphone von Sicherheitspersonal verfährt nach Typ 2. Siehe Sicherheitszentrale.

Szenario: IRON

- Nagios verfährt nach Typ 2. Nagios empfängt neu angemeldete Clients und nimmt diese in seiner Konfiguration auf.
- Snort verfährt nach Typ 1. Snort erkennt Angriffe im Netz und veröffentlicht diese zum MAP-Server.
- TNC@FHH verfährt nach Typ 1 und Typ 2. TNC veröffentlicht Authentifizierungsanfragen von Endgeräten und abboniert Metadaten, um diese im Entscheidungsprozess bei Zulassung von Clients mitwirken zu lassen.
- DHCP verfährt nach Typ 1. Bei der Vergabe von IP-Adressen veröffentlich DHCP diese Vergabe.
- Iptables verfährt nach Typ 2. Iptables empfängt Verstöße, die von Snort oder anderen Sicherheitskomponenten im Netzwerk veröffentlicht wurden und sperrt diese ggf.

Diese Darstellung soll deutlich machen, dass Anwendungen (z.B. Snort, DHCP, Kameraverarbeitung ...) das MAP-Protokoll als Kommunikationsschicht verwenden, um ihre aufgezeichneten Metadaten zu veröffentlichen oder die von anderen aufgezeichneten Metadaten zu empfangen. Hierbei verfolgen die MAP-Clients immer eine klar umrissene Aufgabe.
Bei der Entscheidung zwischen einem Local und einem Remote Service stellt sich hauptsächlich die Frage, ob der Service (Map-Client) von mehr als einer Activity gesteuert wird. Die aktuellen Erfahrungen zeigen, dass die Anwendung, die als MAP-Client fungiert, eine klare Aufgabe besitzt und die Daten, die der MAP-Client empfängt, nicht verschiedenen Anwendungen von Nutzen sind. Bezogen auf das Szenario wäre folgendes Beispiel repräsentativ.
Das Androidphone des Sicherheismitarbeiters abonniert Metadaten von einer Kamera. Die Kameraverwaltung erkennt eine gefährliche Situation und propagiert diese. Daraufhin empfängt der MAP-Client auf dem Androidphone diese Metadaten. Die Anwendung auf dem Androidphone die als MAP-Client fungiert, verarbeitet die empfangenen Daten und bereitet sie für die Anzeige auf.
Eine andere Anwendung hätte keine Verwendung für die abonnierten Daten. Aufgrund dessen wird die Implementierung des MAP-Clients auf Android als Local Service implementiert.

7 Konzept für die Implementierung eines IF-MAP Cients unter Android

In diesem Kapitel werden Architekturentscheidungen erläutert. Weiterhin werden Architekturmodell und Klassendiagramm vorgestellt und erläutert. Daraufhin wird auf das Design des IF-MAP Kommunikationsmodell eingegangen.

7.1 Library oder Framework?

Die Grundfrage für den Aufbau des MAP-Clients ist, welche Art von Implementierung gewählt wird. Möglichkeiten, die sich ergeben sind: ein Framework, eine API oder eine Library. Um dies zu beantworten, wird zunächst deutlich gemacht, worin sich die genannten Möglichkeiten unterscheiden.

Framework heißt übersetzt "Rahmenstruktur". Mit dieser Übersetzung lässt sich gut erklären, was ein Framework ist. Ein Framework gibt eine Struktur vor. Diese Struktur beschreibt bestimmte Standardabläufe. Zu diesen Standardabläufen ist es jedoch möglich, eigene Komponenten einzupflegen. Das Framework wird bei der richtigen Stelle die eigens eingefügten Komponenten aufrufen. Zu diesem Konzept fällt meist der Satz: "Don't call the framework, the framework calls you!". Dieser Satz beschreibt das Konzept sehr treffend, denn der Entwickler ruft grundsätzlich keine Schnittstellen eines Frameworks auf.

Eine Library bietet für einen bestimmten Einsatzzweck "Werkzeuge" an. Diese Werkzeuge können frei benutzt werden. Eine Library gibt keinen Programmfluss vor. Die Library ist lediglich eine Hilfe für den Anwendungsentwickler. Durch Libraries müssen oft auftretende Aufgaben nicht selbst implementiert werden.

Für ein Framework ist IF-MAP zu flexibel. Die Wahl, welche Operation als nächstes getätigt oder wann die Verbindung abgebaut wird, hängt vom Einsatzszenario ab. Dieses wäre nicht mit einzelnen Hot-Spots abzudecken.
Dieses Projekt wird als Library implementiert. Die Entscheidung ist auf folgendes zurückzuführen: Der Ablauf der einzelnen Operationen ist durch eine Spezifikation festgelegt. Dieser Ablauf wird also nur von den jeweiligen Eingabedaten beeinflusst. Genauer genommen resultieren die Antworten vom Server aus den Anfragen vom Client. Für den Benutzer dieser Library ist es also lediglich wichtig, Daten mit Hilfe von Operationen zu verschicken und daraufhin eine Antwort in Form eines Objektes zu erhalten. Dieses

Verfahren ist mit einer Library zu gewährleisten.

7.1.1 Architekturanforderungen

Die API soll selber in sinnvolle Schichten aufgeteilt sein. Diese Schichten sollen lose voneinander gekoppelt sein und eine starke Kohärenz aufweisen. Da dieses Projekt nicht ohne die Hilfe von Libraries für die Verarbeitung von SOAP-Nachrichten auskommen wird, wird dies z.B. eine sinnvolle Schicht sein. Da nicht alle Libraries eine Kompatibilität mit Android aufweisen, ist eine lose Kopplung hierbei sehr wichtig, um einen nachträglichen Austausch oder Wartung dieser Komponente zu ermöglichen.

7.2 Nachrichtenverarbeitung

Da der Nachrichtenaustausch zwischen MAP-Client und MAP-Server über SOAP-Nachrichten stattfindet und in der MAP-Client Anwendung nur mit Java-Objekten gearbeitet wird, muss ein Weg gefunden werden, Java-Objekte in SOAP-Objekte zu transformieren. Um Java-Objekte zu SOAP-Nachrichten und umgekehrt zu transformieren, bedarf es einer geeigneten Library. Dies ist mit vielen Technologien zu erreichen. Dabei muss entschieden werden, wie viel Arbeit der Library überlassen und wie viel selbst implementiert wird. Die Abstufung ist hierbei wie folgt gewählt: Es kann ein einfacher XML-Parser, ein XML-Binder oder eine vollständige SOAP-Libary verwendet werden. Auf den Mapclient bezogen, müsste man mit den genannten Verfahren wie folgt vorgehen:

7.2.1 Geeignete Verfahren

XML-Parser Mit einem XML-Parser ist es möglich, durch die XML-Nachricht zu iterieren. Es wird also jedes Element, Attribut und jeder Wert einzeln überprüft. Bei der Überprüfung muss die geeignete Klasse gefunden und dementsprechend auf diese Klasse abgebildet werden. Ausserdem müssten viele Einzelfälle wie inline Elemente, optionale Elemente etc. beachtet werden.

XML-Binder Ein XML-Binder übernimmt das Parsen. Es ist lediglich nötig, dem Binder mitzuteilen, welche Klassen und seine Attribute äquivalent zum XML-Schema sind. Dies kann z.B. durch Annotations oder Konfigurationsdateien geschehen. Daraufhin wird dem XML-Binder das Objekt übergeben, dass dem Root-Knoten der XML-Nachricht entspricht. Der Binder transformiert dies dann in eine XML-Nachricht.

SOAP-Libary Zuletzt gibt es noch die Möglichkeit, eine volle SOAP-Library zu verwenden. Solch eine Library setzt noch eine Abstraktionsebene höher an als ein XML-Binder.

Eine typische SOAP-Libary übernimmt das Session-Handling zum Server, verarbeitet HTTP Header, XML-Nachrichten, SOAP spezifische Daten und realisiert Sicherheitskonzepte wie WS-Security.

7.2.2 Bewertung

Aufgrund des großen Umfang der IF-MAP Spezifikation und des daraus resultierenden großen XML-Schema wäre es mit großem Aufwand verbunden, die XML Nachrichten mit einem XML-Parser zu verarbeiten. Die Entwicklung mit solch einem Parser wäre also mit nicht vertretbarem Aufwand zu leisten.

Die einzige verfügbare Open-Source SOAP-Libary ist eine für Android angepasste Version von Ksoap2. Demnach trifft die folgende Argumentationskette lediglich auf Ksoap2 zu. Bei der Benutzung einer SOAP-Library übergibt man die Kontrolle der Verbindungen zum Server komplett der Library. Bei diesem Projekt ist dies jedoch nicht gewollt. IF-MAP bietet keinen klassischen synchronen Web-Service an. Speziell wird dies bei dem Aufbau vom Asynchronen Channel deutlich. Bei solch einem Channel muss der Client diese Verbindung "offen" lassen, bis der Map-Server auf die Anfrage antwortet. Solche Einstellungen sind bei Ksoap2 jedoch nicht möglich. Weiterhin gibt es Probleme beim Zugriff von XML Complex Types auf Attribute. Dies ist bei Ksoap2 auf diese Weise leider nicht vorgesehen. Da dass für dieses Projekt jedoch unabdingbar ist, scheidet Ksoap2 spätestens hier aus.

Als einzige Alternative ist das XML-Binding geblieben. Der Vorteil, der hier zu nennen ist, ist dass wir die Kontrolle vom Session-Handling in der Anwendung behalten und den Vorteil der XML-Parsing Abstraktion zusätzlich verwenden. Die Wahl des XML-Binders ist ebenfalls stark eingeschränkt. Die Standard XML-Binding Library im Java Development Kit 6.0 ist "JAXB RI". Leider wurde dies jedoch in der von Google angepassten Android SDK aus dem Core-Package entfernt. Ein Hinzufügen von Klassen in das Java Core-Package gestattet Android wiederum nicht.

Die einzigen derzeit erhältlichen Binding-Libaries, die mit Android kompatibel sind, sind Simple XML und XStream. Die Nachteile bei XStream sind zum Einen die Größe: Mit 8 MB ist XStream weitaus schwergewichtiger als Simple XML. Zum Anderen wurde XStream seit Dezember 2008 nicht weiterentwickelt. Simple XML hingegen wird derzeit durchgehend weiterentwickelt und ist mit 200 KB sehr leichtgewichtig.

7.2.3 Simple XML

Die Verwendung von Simple XML beschränkt sich lediglich auf das Setzen von Annotiatons in den zu transformierenden Java Klassen.

Hier ein Auszug solch einer Klasse:

```
@Element(name="publish")
public class PublishRequestType {
    @ElementList(inline=true, required=false, entry="notify")
    protected ArrayList<NotifyType> notifyList;
    @ElementList(inline=true, required=false, entry="update")
    protected ArrayList<UpdateType> updateList;
    @Attribute(name="validation")
    protected String validation;
    @Attribute(name = "session-id", required = true)
    protected String sessionId;
```

Listing 7.1: Simple XML - Beispielcode

Mit @Element wird ein XML Element deklariert. Im genannten Beispiel wäre das die Klasse selbst. @ElementList wird als Liste von Elementen deklariert, inline bewirkt, dass nur die Objekte in der Liste als Elemente verarbeitet werden. Das Klassenattribut selber wird dann nicht als Element verarbeitet. @Attribute deklariert XML Attribute.

Es muss daraufhin nur noch das Objekt, das das Root Element repräsentiert, zur Serialisierungsklasse von Simple XML übergeben werden.

7.3 Architektur

Der Map-Client ist als Android Local-Service implementiert. Von einer Activity ist dieser also vollständig gelöst. Der Vorteil dieser Trennung ist offensichtlich. Es kann jede beliebige Actvity auf den Service aufsetzen. Weiterhin ist der Local Service lose von der Library SimpleXML getrennt. Dies gewährleistet ein leichtes Austauschen der Library. Da nicht gewährleistet werden kann, dass eine Library wie Simple XML weiterentwickelt wird, ist eine lose Kopplung hier für die Zukunft sehr sinnvoll.

Die Architektur des Local Service ist in zwei Schichten aufgeteilt. Die Kommunikationsschicht ist für folgende Aufgaben entwickelt:

- Aufbau eines SSL-Sockets zum MAP-Server
- Erzeugen und Transformieren von Objekten für die Marshalling-Schicht
- Übergabe von Funktionen an eine Activity
- Kontrolle über den synchronen und asynchronen Socket

Die Marshalling-Schicht übernimmt folgende Aufgaben:

- Aufrufen von Marshallfunktionen der SimpleXML-Libary
- Verarbeitung von HTTP-Headern

Die Kommunikation zwischen Local Service und Activity geschieht über ein Binderobjekt, das vom Local Service angeboten wird.

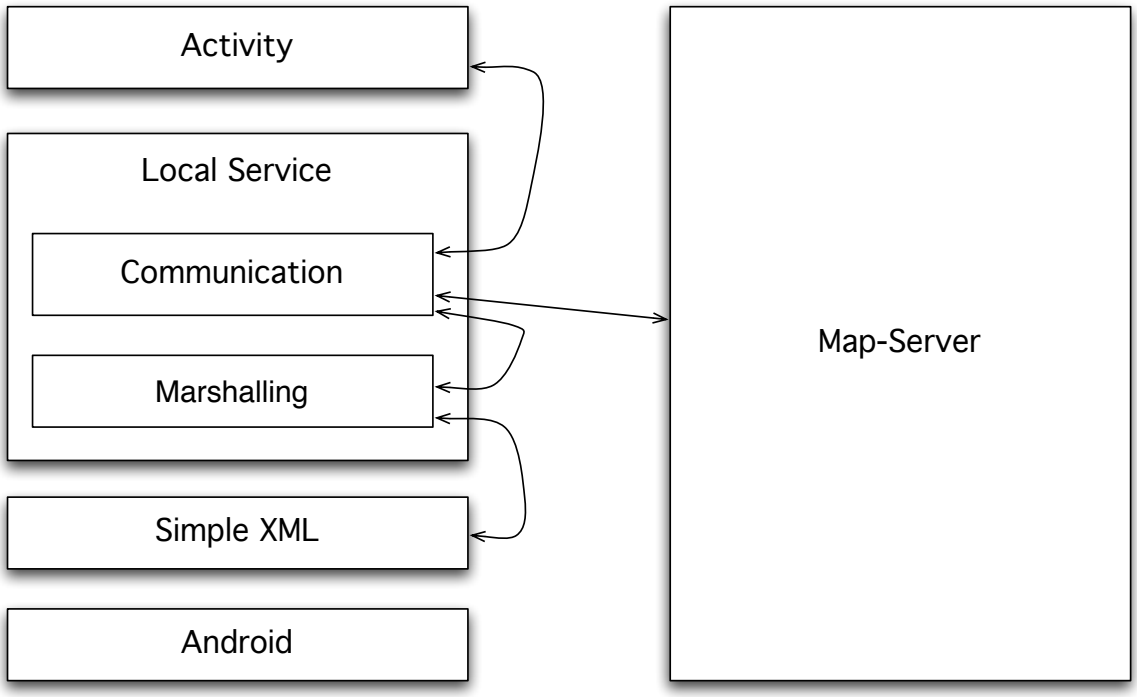

Abbildung 7.1: Architekturmodell

7.4 Klassendiagramm

Die Kommunikation zwischen einer Activity und dem Local Service findet ausschließlich über den ServiceHandler statt. Beim Verbinden zwischen Activity und Local Service wird automatisch die überlagerte Methode onBind aufgerufen. Hierbei wird ein MapclientBinder zurückgegegeben, der den Zugang zum ServiceHandler gewährleistet. Über den ServiceHandler können Operationen zum Mapserver aufgerufen werden. Lediglich Poll benötigt ein Objekt, das Callback als Callback-Handler implementiert .

Die Transformklasse realisiert die schon erwähnte lose Kopplung zwischen dem Service und der Verbindung zur Simple XML Libary. Um die mit Simple XML Anntotiations verarbeiteten Klassen nicht im gesamten Service sowie in den Activitys verwenden zu müssen, verarbeitet die Klasse Transform diese in eigene Objekte. Dies hat zwei entscheidende Vorteile: Zum Einen sind die transformierten Klassen nicht abhängig von Änderungen im XML Schema oder gar einem kompletten Austausch der XML-Binder Library. Zum Anderen ist die Unabhängigkeit von Veränderungen in der IF-MAP Spezifikation ein großer Vorteil. Als Beispiel kann hier Änderung eines Attributnamen angeführt werden. Es wurde in der XML-Schmea ifmap-base-2.0v16 zu ifmap-base-2.0v17 das Attribut publisher-id zu ifmap-publisher-id. Bei solch einer Änderung müsste nur die Annotation in den Simple XML spezifischen Klasse angepasst werden. Wenn die Veränderung jedoch viel weitreichender ist, reicht eine Anpassung in den Simple XML spezifischen Klassen nicht mehr aus.

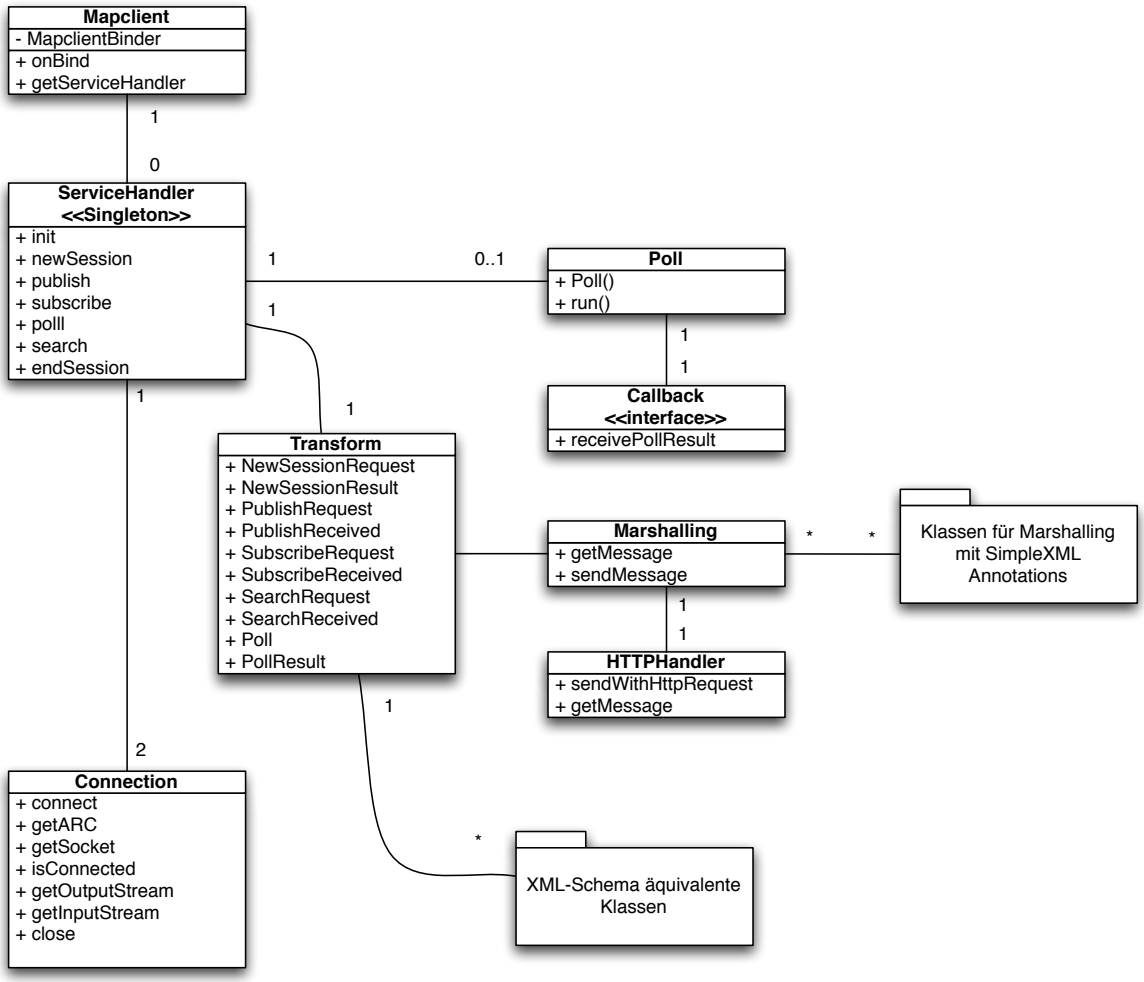

Abbildung 7.2: Klassendiagramm

Die Marshalling Klasse sorgt für die nötigen Aufrufe in der Simple XML Library zum Marshallen sowie Unmarshallen. Weiterhin werden mit dem HTTPHandler die nötigen HTTP Header verabeitet und generiert.

7.5 Design der Operationen

Die möglichen Operationen teilen sich in sieben synchrone und einen asynchronen Aufruf auf. Folgend werden verschiedene Möglichkeiten zur Implementierung vorgestellt.

7.5.1 Callbackhandler

Für jede Operation wird ein neuer Thread gestartet, der den jeweiligen Service aufruft und auf eine Antwort wartet. Anschließend wird mit der Antwort von dem Server eine

Callbackmethode aufgerufen. Je nachdem, was für eine Antwort vom Server kommt, wird eine bestimmte Methode aufgerufen. Da der Server auch eine Fehlerantwort zurückgeben kann, ist es nicht klar, welche Callback-Methode aufgerufen wird. Die Callbackmethoden müssen dann je nach Anspruch ausgefüllt werden. Da keine bestimmte Antwort von dem Server kommt, ist das Zurückkommen der Methoden bei dieser Möglichkeit vorteilhaft gelöst. Weiterhin wäre es dem Aufrufenden möglich, parallel anderweitige Befehle auszuführen.

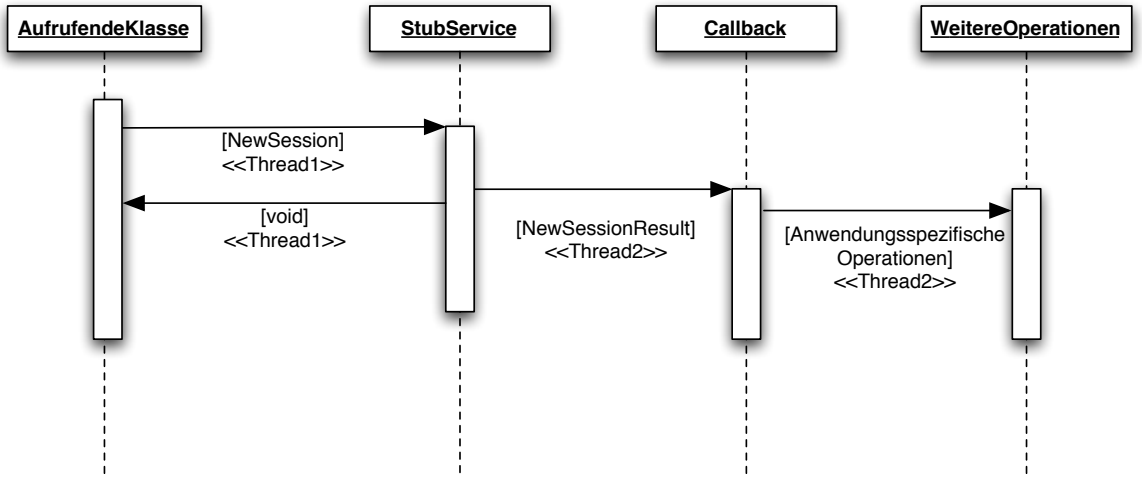

Abbildung 7.3: Architekturablauf - Callbackhandler

7.5.2 Callbackhandler ohne Threads

Weiterhin wäre eine denkbare Möglichkeit, zwar einen zuletzt genannten Callbackhandler zu implementieren, für die jeweiligen Operationen aber keinen neuen Thread zu erzeugen. Der Vorteil bei der Verarbeitung von verschiedenen Antworten bleibt hier sicherlich bestehen, jedoch würde man Einbußen bei der Logik des Programms machen. Die Methode, mit der die Operation aufgerufen wird, würde als gleicher Thread die Callbackmethode aufrufen. Dies ist jedoch für ein typisches Callback-Handling nicht üblich.

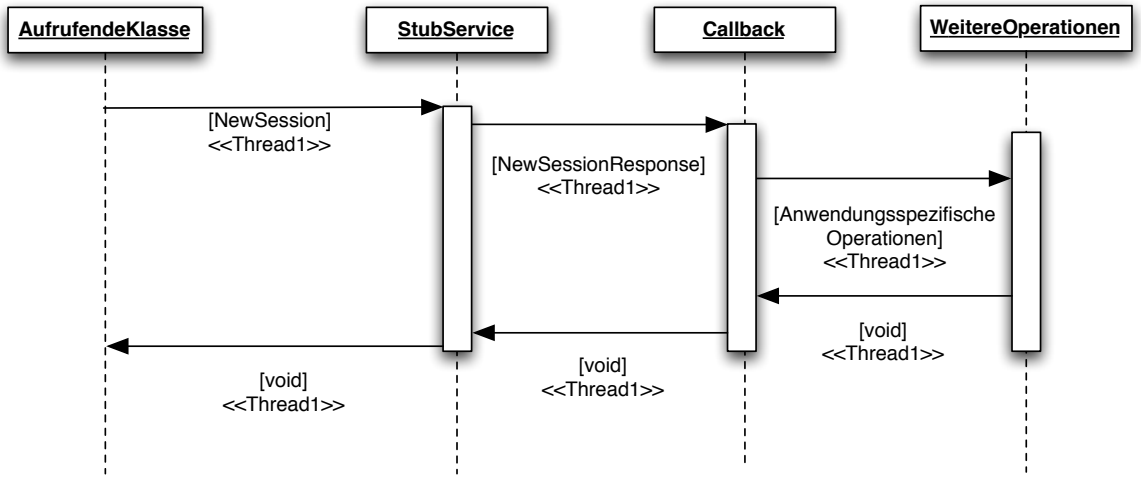

Abbildung 7.4: Architekturablauf - Callbackhandler ohne eigene Threads

7.5.3 Originale Abbildung

Eine weitere Möglichkeit wäre es, die synchronen Aufrufe mit normalen Rückgabeparametern zu realisieren. Hierbei könnten die alternativen Fehlerantworten im Exception-Handling abgefangen und bearbeitet werden. Poll als einzige asynchrone Operation würde weiterhin mit einem Callbackhandler realisiert werden.

7.5.4 Bewertung

Dieses Projekt wird als Local-Service implementiert und soll möglichst "einfache" und für den Benutzer möglichst "logisch" nachvollziehbare Schnittstellen bieten. Aufgrund dessen wird in diesem Projekt für den Android-Client eine originale Abbildung implementiert. Weiterhin ist mit dem Aufrufen der genannten synchronen Operationen auf dem Server nicht mit Wartezeit zu rechnen. Deswegen wäre ein extra Thread hierfür nicht nötig.

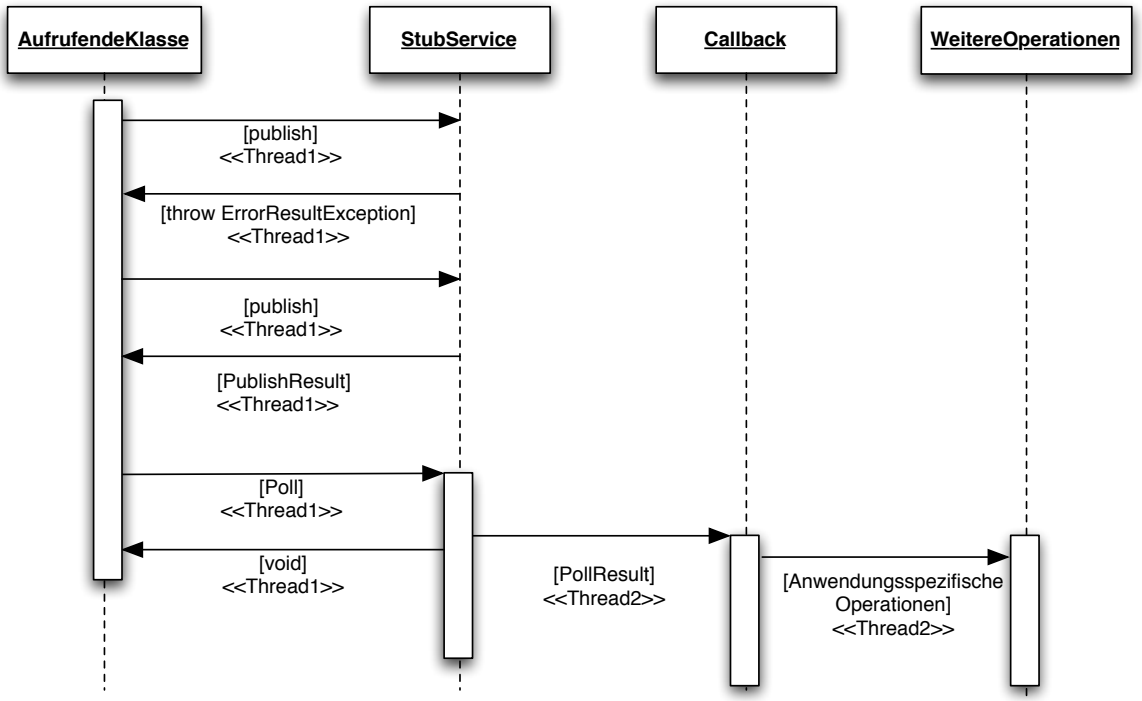

Abbildung 7.5: Architekturablauf - Originale Abbildung

7.6 Bewertung des Konzepts

Im Bezug zu den in Kapitel 5.2 genannten Anforderungen wird folgend eine Übersicht darüber erstellt, welche der Anforderungen durch das Konzept erfüllt werden. Die restlichen Anforderungen müssen durch die Implementierung erfüllt werden.

Nichtfunktionale Anforderungen

- Verschlüsselte Verbindung: Die Architektur in Kapitel 7.3 beschreibt einen Verbindungsaufbau vom MAP-Client zum MAP-Server über einen SSL-Socket.
- Ressourcensparend und leichtgewichtig: Es wurde im Kapitel 7.2.2 bei der Wahl einer geeigneten XML-Library darauf geachtet, dass die Library möglichst leichtgewichtig ist.
- Logische und ergonomische Schnittstelle für Entwickler: Durch die Entscheidung, den MAP-Client als Local-Service und in Form einer API zu implementieren, wird dem benutzenden Entwickler eine leicht zugängliche und logische Schnittstelle geboten.

Technische Anforderungen

- Android als Betriebssystem: Im Kapitel 7.2.2 wurde auf eine Kompatibilität der XML-Library mit Android geachtet. Weiterhin wurde das Konzept für die Implementierung des MAP-Client auf die Android-Plattform ausgerichtet, wie im Kapitel 6.6.3 ersichtlich.
- SOAP als Kommunikationsprotokoll: Es wurde im Kapitel 7.2 festgelegt, wie Java-Objekte in SOAP-Nachrichten transformiert werden. Dadurch ist die Kommunikation über das SOAP-Protokoll gewährleistet.

8 Realisierung der IF-MAP API

8.1 Entwicklungsumgebung

Um Anwendungen für Android zu implementieren, kann nicht die Java Development Kit von Sun benutzt werden. Es muss das Android SDK (Software Development Kit) von Google verwendet werden. Um die Anwendung im Entwicklungsprozess zu testen, benötigt man einen Android-Emulator. Dieser ist, wie das Android SDK, auf der offiziellen Entwickler-Seite von Android zu beziehen. Um jedoch im Entwicklungsprozess die Software zu testen, müsste bei dieser Variante jedes mal wieder das Projekt kompiliert und auf den Emulator geladen werden. Außerdem wäre die Möglichkeit, eine Software zu debuggen, nicht gegeben. Da dies jedoch unproduktiv und aufwendig ist, wurde von Google das ADT Plugin (Android Development Tools) veröffentlicht. Dieses Plugin ermöglicht die volle Integration des Android Entwicklungsprozesses in die Eclipse IDE. ADT unterstützt ein automatisches Laden der Software auf den integrierten Emulator. Weiterhin ist eine volle Debugging-Umgebung gegeben, die eine direkte Verbindung mit den Emualtor aufbaut. Das ADT Plugin kann ebenfalls auf der offiziellen Entwicklerhompage von Android bezogen werden.

8.2 Verbindungen

Im Kapitel 3.7 wurde beschrieben, dass es zwei Möglichkeiten gibt, eine Verbindung zum MAP-Server aufrecht zu erhalten. Zum Einen durch eine offen bleibende TCP Verbindung und zum Anderen durch "renewSession" Operationen. Bei der Implementierung wurde die erste Variante gewählt. Der Client baut den Socket auf und hält ihn, solange eine endSession Operation versendet wird. Wird der Socket z.B. bei Verbindungsproblemen abgebaut, verbindet der Client direkt neu.

8.3 Authentifizierung am Server

Der Server bietet zwei Möglichkeiten, sich zu authentifizieren. Erst nach der Authentifizierung ist der Client autorisiert, Operationen auf dem Server auszuführen. Eine Möglichkeit, sich zu authentifizieren, ist durch eine HTTP Basic Authentication. Der Client sendet hierbei die Authentifizierung mit dem Authorization-Header (User:Password) Base64-codiert an den Server. Schlägt die Authentifizierung fehl, antwortet der Server mit einem HTTP 401 Fehlercode. Die zweite Möglichkeit geht über eine Client-Authentifizierung im SSL Handshake. Hierbei sendet der Client nach dem "Server Hello

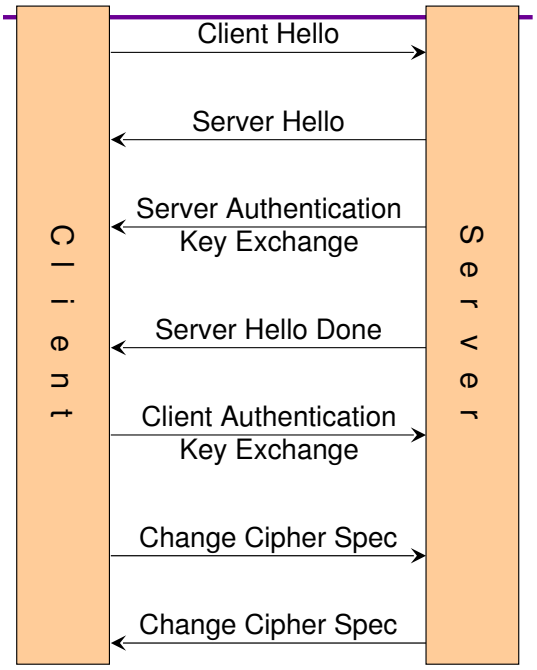

Abbildung 8.1: SSL Handshake: Quelle: [16]

Done" sein Client Zertifikat zum Server.

Bei der zweiten Möglichkeit muss der Truststore im Server zwingend das Client Zertifikat beinhalten. Bei beiden Möglichkeiten muss jedoch der Truststore des Clients das Server Zertifikat beinhalten. Android hat einen globalen Truststore für alle Anwendungen, die auf dem Endgerät installiert sind. Dieser Truststore erlaubt jedoch nur die Hinzufügung von signierten Zertifikaten aus vertrauenswürdigen Zertifizierungsstellen. Selbstsignierte Zertifikate müssen hier aus Sicherheitsgründen außen vor bleiben. Denn wenn eine Anwendung ein selbstsigniertes Zertifikat hinzufügt, vertrauen automatisch alle anderen Android Anwendungen ebenfalls diesem Zertifikat. Einem Angreifer wäre es somit z.B möglich, ein Zertifikat für eine Phishing-Seite bereitzustellen. Dennoch wird bei Android die Möglichkeit gegeben, selbstsignierte Zertifikate verwenden zu können.

8.4 Eigener X509TrustManager

Um die Überprüfung des Server Zertifikates selber zu übernehmen, muss ein eigener X509TrustManager implementiert werden. Ein TrustManager entscheidet, ob ein Zertifikat abgelehnt oder akzeptiert wird. Im Projekt wurde dies folgendermaßen realisiert:

Es wurde eine Klasse mit dem Namen MyX509TrustManager erstellt, die das Interface X509TrustManager implementiert. Die Klasse beinhaltet ein privates Attribut serverTrustManager vom selben Typ wie das implementiere Interface X509TrustManager.

Außerdem bietet die Klasse eine eigene Methode loadTrustStore:

```
public static void loadTrustStore(KeyStore ks, String kpw)
                throws CertificateException {
        keystorePW = kpw;
        try {
                keystore = ks;
                javax.net.ssl.TrustManagerFactory tmf =
                        javax.net.ssl.TrustManagerFactory.getInstance("X509");
                tmf.init(keystore);
                TrustManager[] tms = tmf.getTrustManagers();
                if (tms != null) {
                        for (TrustManager tm : tms) {
                                if (tm instanceof X509TrustManager) {
                                        serverTrustManager = (X509TrustManager)tm;
                                        break;
                                }
                        }
                }
        } catch (Exception e) {
                keystore = null; throw new CertificateException(e);
        }
}
```

Listing 8.1: Methode zum Keystore laden

Der geladene Truststore wird in eine TrustManagerFactory initialisiert. Eine TrustManagerFactory erzeugt mit Hilfe des übergebenen TrustStore einen TrustManager. Es wird ein Array vom Typ TrustManager zurückgegeben, da die Möglichkeit besteht, verschiedene Typen eines TrustManagers zu erzeugen. Da hier jedoch nur ein TrustManager des Typs X509 benötigt wird, wird dieser im Array gesucht und dem privaten Attribut serverTrustManager übergeben.
Bei der Abfrage, ob ein Zertifikat vertrauenswürdig ist, wird immer die Methode checkServerTrusted aufgerufen. Diese muss in der Klasse implementiert bzw. "überschrieben" werden:

```
public void checkServerTrusted(X509Certificate[] chain, String authType)
                throws CertificateException {

serverTrustManager.checkServerTrusted(chain, authType);
chain[0].checkValidity();
}
```

Listing 8.2: Überschreiben der Methode checkServerTrusted

Übergeben wird hierbei ein X509 Zertifikat. Es muss überprüft werden, ob das angefragte Zertifikat im Truststore vorhanden ist. Dies kann an das Attribut serverTrustManager weitergeleitet werden. Bei erfolgreicher Überprüfung wird keine Exception geworfen.

Um den Socket aufzubauen muss daraufhin nur noch bei der Socket Initialisierung der eigene TrustManager und ein KeyManager übergeben werden:

8.5 Nutzung des MAP-Clients

Da der MAP-Client als Local-Service implementiert ist, ist der er über ein Intent ansprechbar.

```
private Mapframework.MapclientBinder mapclientBinder;
private ServiceConnection localServiceConnection =
        new ServiceConnection() {

                @Override
                public void onServiceDisconnected(ComponentName arg0) {

                }

                @Override
                public void onServiceConnected(ComponentName arg0, IBinder arg1) {
                        mapclientBinder = (Mapframework.MapclientBinder) arg1;
                }
        };

private void connectService () {
        Intent intent = new Intent (getApplicationContext(), Mapframework.class);
        bindService(intent, localServiceConnection, Context.BIND_AUTO_CREATE);
}
```

Listing 8.3: Verbindung zum MAP-Client

Das localServiceConnection Objekt vom Typ ServiceConnection repräsentiert die Verbindung. Über die connectService() Methode wird ein Intent zum MAP-Client hergestellt. Mit der bindService() Methode wird die Verbindung von der Activity zum MapClient aufgebaut. Mit dem Objekt mapclientBinder kann daraufhin der ServiceHandler bezogen werden:

```
public ServiceHandler getServiceHandler (String endpoint, int port,
                InputStream isKeyStore, InputStream isTrustStore,
                String keystorePW, String trustStorePW) throws Exception {

        return mapclientBinder.getServiceHandler(endpoint, port, isKeyStore,
                                        isTrustStore, keystorePW, trustStorePW);
}
```

Listing 8.4: Schnittstelle zum MAP-Client

Der getServiceHandler() Methode müssen alle nötigen Parameter zum Verbindungsaufbau übergeben werden. Über das ServiceHandler Objekt können daraufhin alle IF-MAP Operationen aufgerufen werden. Die Methoden im ServiceHandler Objekt sind die einzigen Schnittstellen, die benötigt werden, um den MAP-Client zu bedienen.

Eine NewSession Operation wird z.B. wie folgt aufgerufen:

```
String sessionID;
String publishID;
NewSessionResultType nsrt;
        try {
                nsrt = sh.newSession();
                sessionID = nsrt.getSessionId();
                publishID = nsrt.getIfmapPublisherId();

        } catch (ErrorResultType e) {
                System.out.println(e.getErrorString());
                System.out.println(e.getErrorCode());
        }
```

Listing 8.5: Aufruf einer NewSession Operation

Nach dem Aufruf von newSession() wird ein ein Objekt vom Typ NewSessionResultType zurück gegeben. Dieses Objekt beinhaltet die publisherID und die sessionID. Falls ein Fehler bei der Anfrage aufgetreten ist, antwortet der MAP-Server mit einem ErrorResultType. Die Rückgabe wird hier mit einer Exception realisiert.

8.6 Bewertung der Implementierung

Folgende gestellte Anforderungen aus Kapitel 5.2 wurden durch die Implementierung erfüllt.

Funktionale Anforderungen

- Veröffentlichen, Suchen und Abonnieren von Metadaten: Es wurde der komplette Sprachumfang von IF-MAP der in der IF-MAP Spezifikation [12] beschrieben ist, implementiert. Darin sind die drei genannten Operationen eingeschlossen.

- Benutzung beliebiger Metadaten: Bei der Implementierung wurde die Möglichkeit offen gelassen, beliebige Metadaten zu Versenden und zu Empfangen. Dazu müssen lediglich die nötigen Metadaten als Simple-XML definierte Java-Objekte vorliegen.

- Schnittstelle für Entwickler: Der IF-MAP Client kann durch einen anderen Service oder Activity angesprochen und voll genutzt werden. Ein Beispiel hierfür wird in Kapitel 8.5 erläutert.

Nichtfunktionale Anforderungen

- Authentifizierung: Die Implementierung der Authentifizierung wurde erfolgreich implementiert. In Kapitel 8.3 wird beschrieben, welche Besonderheit es bei der Implementierung gab.

Technische Anforderungen

- Interoperabilität Der MAP-Client wurde erfolgreich mit dem MAP-Server aus dem IRON Projekt getestet und gewährleistet die Interoperabilität mit jedem MAP-Server.

9 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine vollständige spezifikationskonforme MAP-Client Library für die Android-Plattform entwickelt. Es wurde bei der Anforderungsanalyse erkannt, dass ein MAP-Client auf der Android-Plattform sehr viele Einsatzmöglichkeiten haben kann. Aufgrund dessen wurde festgelegt, dass ein MAP-Client entwickelt wird, der möglichst für jedes Szenario ohne eine Anpassung genutzt werden kann. Dies erforderte, dass der MAP-Client alle in der IF-MAP Spezifikationen beschriebenen Operationen beherrschen muss und sowohl als "subscriber" als auch als "publsiher" fungieren kann.
Daraufhin wurde im Beispielszenario gezeigt, wie eine solche MAP-Client Library genutzt werden könnte. Die aus dem Konzept und Implementierung resultierende MAP-Client Library ist wie im Implementierungsbeispiel mit jeglichen Android-Service oder Activity benutzbar. Weiterhin wurde eine Beispiel-Activity entwickelt, die alle IF-MAP Operationen ausführt und die Ergebnisse dieser Operationen grafisch demonstriert.

Der ServiceHandler arbeitet derzeit noch mit den generierten Klassen von SimpleXML. Das beeinträchtigt nicht die Funktionsweise, aber widerspricht dem Klassendiagramm. Es wurden alle Anforderungen und Ziele der Aufgabenstellung erfüllt. Die implementierte MAP-Client Library soll für Entwickler eine leichte Möglichkeit bereitstellen, Operationen auf einer MAP auszuführen. Die Library kann für eine Anwendung als Kommunikationsschicht benutzt werden. Da das IF-MAP Protokoll noch jung ist, wurde in der Vergangenheit die IF-MAP Spezifikation aktualisiert und wird sich höchstwahrscheinlich auch in Zukunft verändern. Aufgrund dessen muss die MAP-Client API bei Aktualisierungen dahingehen angepasst werden, um weiterhin eine Interoperabilität zum MAP-Server zu gewährleisten.

10 Literaturverzeichnis

[1] Android (betriebssystem). Verfügbar online auf http://de.wikipedia.org/wiki/Android_(Betriebssystem).

[2] Javadoc - keystore. Verfügbar online auf http://download.oracle.com/docs/cd/E17476_01/javase/1.5.0/docs/api/java/security/KeyStore.html.

[3] Rfc 5246 the transport layer security (tls) protocol version 1.2. Verfügbar online auf http://tools.ietf.org/html/rfc5246.

[4] Schnittstellenbeschreibungssprache. Verfügbar online auf http://de.wikipedia.org/wiki/Schnittstellenbeschreibungssprache.

[5] What is android? Verfügbar online auf http://developer.android.com/guide/basics/what-is-android.html.

[6] A. Becker and M. Pant. *Android - Grundlagen der Programmierung*. dpunkt.verlag, 2009.

[7] I. Bente. Iron: Project structure and current status. Verfügbar online auf http://trust.inform.fh-hannover.de/joomla/index.php/projects/iron/92-iron-overview.

[8] P. D. Bruns. *Software Engineering 2 - Vorlesungsscript*. WS 09/10.

[9] P. D. Dunkel. *Software Engineering 1 - Vorlesungsscript*. SS09.

[10] C. U. Nicola. Imvs fokus report, 2009.

[11] TCG. *TNC Architecture for Interoperability*, May 2009. Specification Version 1.4 Revision 4.

[12] TCG. *TNC IF-MAP Binding for SOAP*, March 2010. Specification Version 2.0 Revision 34 (unpublished).

[13] TCG. *TNC IF MAP Metadata for Network Security*, March 2010. Specification Version 1.0 Revision 20.

[14] S. Webster. 160,000 android phones sold daily, market nears 70,000 applications. Verfügbar online auf http://www.androidguys.com/2010/06/23/160000-android-phones-sold-daily-market-nears-70000-applications/.

[15] J. Westhuis. Integritätsmessung und -prüfung für android-basierte endgeräte.

[16] P. D. Wohlfeil. *Betriebssysteme und Netze 2 - Vorlesungsscript*. WS 09/10.

[17] S. Y.Hashimi and S. Komatineni. *Pro Android*. apress, 2009.