Abteilung Informatik
Fakultät IV
Fachhochschule Hannover

Visualisierung von Informationen einer zentralen Netzwerk-Datenbank

Bachelorarbeit im Studiengang Angewandte Informatik

Tobias Ruhe

Juli 2010

Beteiligte Personen

Erstprüfer
Prof. Dr. rer. nat. Josef von Helden
Fachhochschule Hannover
Ricklinger Stadtweg 120
30459 Hannover
E-Mail: josef.vonhelden@fh-hannover.de
Tel.: +49 (0)511-9296-1500 / -1800
Fax.: +49 (0)511-9296-1510

Autor
Tobias Ruhe
Badenstedter Straße 9
30449 Hannover
E-Mail: tobias.ruhe@stud.fh-hannover.de

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich die vorliegende Bachelorarbeit selbstständig, ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.

Hannover, den 2. August 2010

Tobias Ruhe

Inhaltsverzeichnis

Glossar

AR	Access Requestor
ARC	Asynchronous receive channel
FHH	Fachhochschule Hannover
$IFMAP$ ...	Interface Metadata Access Point Protocol
$JAXB$	Java Architecture for XML Binding
$JAXB-RI$	Java Architecture for XML Binding Reference Implementation
JRE	Java Runtime Enviroment
$JSSE$	Java Secure Socket Extension
MAP	Metadata Access Point Server
$MAPC$	Metadata Access Point Client
$MAPS$	Metadata Access Point Server oder MAP-Server
PDP	Policy Decision Point
PEP	Policy Enforcement Point
$SSRC$	Synchronous send-receive channel
TNC	Trusted Network Connect
TNG	Trusted Network Group
$WSDL$	Webservice Description Language

1. Einleitung

1.1. Ziel / Motivation der Arbeit

Unter Visualisierung versteht man allgemein die Veranschaulichung von (abstrakten) Daten. Diese werden in einer geeigneten visuellen Form dargestellt, um somit Auswertungen und Analysen zu ermöglichen. Der Anwender hat ein Hilfsmittel zur Hand, um Zusammenhänge darzustellen, die nicht sofort ersichtlich sind. Besonders bei komplexen Graphenstrukturen ist eine geeignete Visualisierung hilfreich, weil es bei den großen Datenmengen, die miteinander in Verhältnis stehen, schwierig sein kann, den Überblick zu bewahren.
Die Netzwerk-Datenbank, die in dieser Arbeit behandelt wird, hat ihren Platz im *Trusted Computing*, das sich mit dem sicheren Umgang mit Computern und empfindlichen Daten befasst. Diese Datenbank wird in diesem Zusammenhang *Meta Data Access Point* genannt und speichert ihre Daten in Form einer Graphenstruktur.
Das Ziel dieser Arbeit ist es, diese Daten anschaulich zu visualisieren. Dabei wird ein Konzept entwickelt, um die Graphenstruktur aus dieser Datenbank zu lesen und darzustellen. Anschließend wird die Umsetzung dieses Konzeptes vorgenommen.

1.2. Aufbau der Arbeit

Diese Arbeit gliedert sich in folgende Kapitel:

Grundlagen
In Kapitel 2 wird eine Einführung in das von der TNC-Group entworfene Netzwerk-Szenario gegeben. Die Funktionsweise eines MAP-

Servers und seine zugrundeliegenden Techniken werden beschrieben.

Das IF-MAP Protokoll
In Kapitel 3 wird das IF-MAP-Protokoll tiefergehend erklärt.

Analyse des IRON-MAP-Servers
Der IRON-MAP-Server, die Implementierung der Fachhochschule Hannover eines MAP-Servers, wird analysiert und auf Funktionalität und Schnittstellen geprüft (Kapitel 4).

Konzept und Entwurf einer Visualisierung eines MAP-Servers
Nachdem in den vorigen Kapiteln die Grundlagen geschaffen wurden, wird in Kapitel 5 ein Konzept zur Visualisierung der MAP-Daten vorgestellt. Dabei werden zunächst die Anforderungen an den Client und den MAP-Server festgelegt. Darauf aufbauend, werden verschiedene Lösungsansätze diskutiert. Anschließend werden für den besten Lösungsansatz Implementierungsansätze besprochen.

Design und Implementierung der Softwarekomponenten
Basierend auf dem entwickelten Konzept in Kapitel 5, wird hier die Implementierung vorgenommen (Kapitel 6).

Ergebnisse und Bewertung
Die erreichten Resultate werden in Kapitel 7 reflektiert und bewertet.

1.3. Typographische Konventionen

Folgende typographische Konventionen werden in dieser Arbeit eingehalten:

kursiver Text Schlüsselwörter, Technologien, Konfigurationen
`teletype` Klassennamen

2. Grundlagen

Dieses Kapitel bietet zuerst eine kurze Einführung in *Trusted Computing*. Innerhalb des *Trusted Computing* gibt es eine Netzwerkarchitektur namens *Trusted Network Connect* (TNC), die ebenfalls erläutert wird. Der Schwerpunkt liegt hier auf dem MAP-Server, der ein Bestandteil der *TNC*-Architektur ist. Anschließend werden grundlegende Technologien, die für die Arbeitsweise eines MAP-Servers von Bedeutung sind, erklärt.

2.1. Trusted Computing

Die Trusted Computing Group [1] (TCG) ist eine Non-Profit-Organisation, die offene Standards zum Thema Trusted Computing entwickelt und veröffentlicht. Sie verabschiedet Spezifikationen, die von Entwicklern umgesetzt werden können. Die Organisation setzt sich aus mehr als hundert namhaften Mitgliedern der Industrie zusammen (u.a. IBM, Microsoft, Intel).
Die TCG besteht aus mehreren Untergruppen, die sich jeweils mit einem Teilgebiet des *Trusted Computing* beschäftigt. Eine dieser Gruppen ist die *Trusted Network Connect Work Group* (TNC-WG) [2], die sich mit der Integrität von Endpunkten in einem Netzwerk beschäftigt.

[1] http://www.trustedcomputinggroup.org/
[2] http://www.trustedcomputinggroup.org/developers/trusted_network_connect

2.2. Trusted Network Connect

Trusted Network Connect ist eine Architektur zur Netzwerkkontrolle in heterogenen Netzwerken. Es schützt ein Netzwerk vor unautorisierten Zugriffen und prüft die Integrität von Endgeräten und Benutzern durch Richtlinien (policies). Es gibt verschiedene Komponenten in der TNC-Architektur, die je nach Funktionalität unterschiedliche Rollen einnehmen (siehe Abbildung 2.1).

- **AccessRequestor (AR)** Ein Endpunkt, der Zugang zum Netzwerk erfragt (Laptop, mobiles Endgerät)
- **Policy Enforcement Point (PEP)** Entscheidet über den Zugang (RADIUS Server, VPN-Controller)
- **Policy Decision Point (PDP)** Prüft, ob dem Client der Zugang erlaubt wird oder nicht (VPN-Gateways, 802.1X Switches)
- **Metadata Access Point (MAP)** Dient zum Speichern und Verwalten von Meta-Informationen über das Netzwerk (MAP-Server)
- **Metadata Access Point Client (MAPC)** Speichert und sucht Metadaten auf dem MAP-Server (DHCP-Server, Snort)

Alle Netzwerk-Informationen werden zentral im MAP-Server gespeichert und bei Bedarf abgerufen und ausgewertet. Jede Komponente die das IF-MAP-Protokoll implementiert, wird MAP-Client genannt.

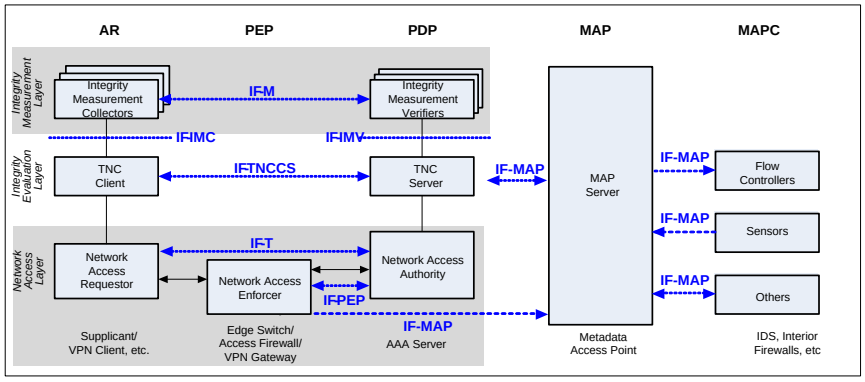


Abbildung 2.1.: Verschiedene Protokolle werden in der TNC-Architektur benutzt [Tru10a, Seite 16 Abbildung 1].

2.3. MAP-Server

Ein Metadata Access Point Server (im folgenden MAP-Server genannt) ist eine Netzwerkdatenbank in der TNC-Architektur, die für die Verwaltung von netzwerkrevelanten Informationen zuständig ist. In einem MAP-Server werden Statusinformationen (Metadaten) über Benutzer im Netzwerk, angeschlossene Geräte und Daten über den Netzwerkverkehr gespeichert. Der MAP-Server stellt über eine Schnittstelle diverse Operationen zum Speichern, Löschen und Suchen der Metadaten zur Verfügung. Das hierzu verwendete Protokoll IF-MAP setzt das Client-Server-Konzept um. Ein Client, der die Dienste eines MAP-Servers in Anspruch nimmt, wird als MAP-Client (MAPC) bezeichnet (Abbildung 2.2). Der MAP-Server arbeitet nach dem publish/subscribe Paradigma: Möchte ein MAP-Client über Statusänderungen von bestimmten Daten informiert werden, so kann er sich für dieses Ereignis beim Server registrieren. Dieser stellt ein Benachrichtigungsmechanismus zur Verfügung und informiert den Client bei Änderungen der Daten. Das IF-MAP-Protokoll und die Funktionalitäten eines MAP-Server werden, wie die TNC-Architektur, von der TCG spezifiziert.

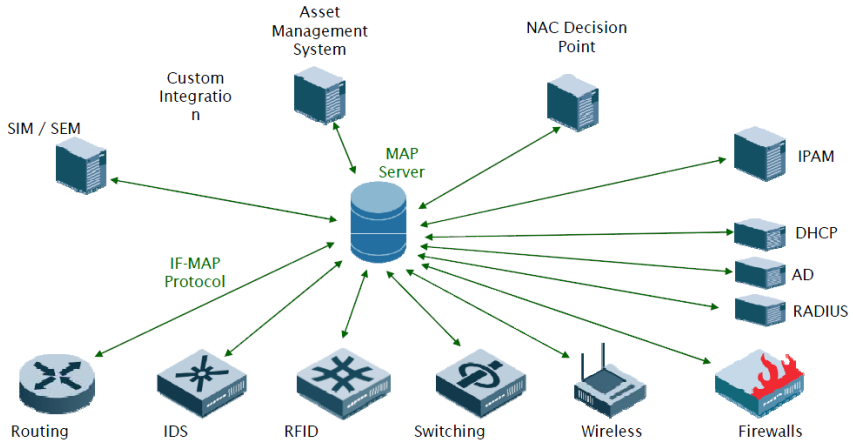


Abbildung 2.2.: Diverse Netzwerkkomponenten kommunizieren mit dem MAP Server über das IF-MAP-Protokoll [Tru09].

2.3.1. Schnittstellen

Die einzige Möglichkeit mit dem MAP-Server zu kommunizieren, ist das IF-MAP-Protokoll. Als Transportprotokoll steht lediglich HTTPS zur Verfügung, welches eine SSL-Verschlüsselung der Daten erfordert. Alle Operationen und Datentypen werden in IF-MAP als XML repräsentiert.

2.4. Verwendete Technologien

In diesem Abschnitt werden Technologien, die für diese Arbeit wichtig sind, erläutert.

2.4.1. XML Schema

Um die Struktur von XML-Dokumenten zu beschreiben, hat die W3C [W3Cb] die Schemasprache XML Schema [W3C04] eingeführt. XML Schema, auch XSD (XML-Schema-Definition) genannt, spezifiziert die Strukturen in XML. Es gibt zwei Datentypen: einfache und komplexe Datentypen. Die einfachen (atomaren) Datentypen werden von XML Schema zur Verfügung gestellt. Sie spiegeln grund-

legende Datentypen wieder, die man in vielen Programmiersprachen verwendet. Beispielsweise steht der Typ `<xsd:string>` für eine Zeichenkette und `<xsd:int>` stellt ein Integer dar. Einfache Datentypen dürfen keine Kind-Elemente enthalten.
Komplexe Datentypen sind aus einfachen Datentypen zusammengesetzt und können selber definiert werden:

```
<xsd:complexType name="Person">
  <xsd:element name="Vorname" type="xsd:string"/>
  <xsd:element name="Nachname" type="xsd:string"/>
  <xsd:element name="Geburtsjahr" type="xsd:integer"/>
</xsd:complexType>
```

Listing 2.1: Komplexer Datentyp in XML Schema

Die komplette Definition kann in der offiziellen Spezifikation[3] des W3C nachgelesen werden.

2.4.2. SOAP

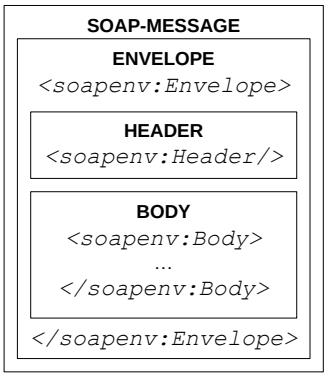


Abbildung 2.3.: Aufbau einer SOAP-Nachricht mit leerem Header

Das Simple Object Access Protokoll (SOAP) [W3C07] dient zum Austausch von XML-Nachrichten über ein Netzwerk. Der Name ist ein wenig irreführend, da keine Objekte ausgetauscht werden, sondern Nachrichten. Die Daten werden als XML repräsentiert und können über diverse Internetprotokolle verschickt werden. Das können beispielsweise FTP, IMAP oder HTTP/HTTPS sein. Letzte-

[3] http://www.w3.org/TR/xmlschema-0/

res Transportprotokoll ist für den MAP-Server vorgeschrieben. Eine SOAP-Nachricht besteht immer aus einem Envelope(Umschlag)-Element, in dem sich ein Body-Element befinden muss (Abbildung 2.3). Optional kann auch ein Header-Element dem Envelope zugefügt werden. Der Rumpf (Body) enthält die eigentlichen Nutzdaten, im Header können sich zusätzliche Meta-Informationen befinden, die aber lediglich den Transport der Daten betreffen.
Das folgende Listing zeigt beispielhaft eine SOAP-Nachricht:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
<soapenv:Header/>
   <soapenv:Body>
      <ns:neueAnfrage attribute1="einszwei" attribute2="dreivier" />
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 2.2: Eine SOAP-Nachricht

neueAnfrage ist die Operation, die aufgerufen wird. *attribute1* und *attribute2* parametrisiert den Aufruf.

2.4.3. WSDL

Die WSDL [W3Ca] (*Webservice Description Language*) ist eine Beschreibungssprache für Webservices. Hier werden Datentypen und Operationen, die ein Service anbieten will, definiert. Eine WSDL kann als Eingabe für *xjc* genommen werden. Sowohl die Datentypen, als auch die Funktionen werden in den erzeugten Klassen erstellt und stehen dem Programmierer zur Verfügung.

2.4.4. Apache Axis2

Axis2 [Apa09] ist ein Java-Framework der Apache Foundation zum Entwickeln von Webanwendungen auf Basis von SOAP. Ein Entwickler von Webdiensten kann seine Funktionen und Datentypen in Form von WSDL-Dateien definieren und daraus Quellcode generieren. Der erzeugte Code enthält Funktionen, die zur Kommunikation zwischen dem Webservice-Anbieter und dem Client zuständig sind.

Client und Server tauschen SOAP-Nachrichten aus. Ein Entwickler arbeitet aber mit Objekten oder anderen Datenstrukturen, deshalb müssen die XML-Nachrichten in JAVA-Objekte transformiert werden und umgekehrt. Die Funktionen und Datentypen aus einer SOAP-Nachricht werden also in JAVA-Objekte abgebildet.
Auch der umgekehrte Weg ist möglich: Aus vorhandenen Quellcode können WSDL-Dateien generiert werden.
Die Art und Weise, wie XML-Nachrichten als JAVA-Objekte abgebildet werden, ist variabel. Hierfür stehen verschiedene Datenbindungsmodelle zur Verfügung, die beim Generieren der Objekte vom Programmierer gewählt werden können. JAXB, das im nächsten Abschnitt erläutert wird, ist ein solches Datenbindungsmodell.
Die Werkzeuge die *Axis2* dafür mitbringt, heißen *WSDL2JAVA* und *JAVA2WSDL*.

2.4.5. JAXB-RI

JAXB-RI [Oraa] ist die Referenzimplementierung der JAXB (*Java Architecture for XML Binding*), die für die Datenbindung zwischen XML-Elementen und Java-Objekten zuständig ist. XML-Daten werden als Java-Objekte abgebildet und umgekehrt. Der Entwickler muss sich also nicht mehr selber die Mühe machen, die Transformationen vorzunehmen.
Um JAXB-RI zu nutzen, definiert man sich zunächst seine Funktionen und Datentypen in Form einer XML-Datei. Daraus generiert man mit dem mitgelieferten Binding-Compiler *xjc* Java-Klassen (oder C++-Code), die das XML-Binding übernehmen.

2.4.6. Java Secure Socket Extension

Die *Java Secure Socket Extension* (JSSE) ist eine seit J2SE 1.4 integrierte API zur Kommunikation über SSL [4]. Sie dient zur Verschlüsselung, Authentifizierung und Sicherung der Datenintegrität. Zum

[4] Secure Socket Layer

Lieferumfang gehört auch das *keytool* [5], mit dem man Zertifikate ausstellen und Schlüssel generieren kann.

[5] http://download.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.html

3. Das IF-MAP Protokoll

Wie in Kapitel 2 erläutert, nutzen MAP-Clients das IF-MAP-Protokoll um mit einem MAP-Server zu kommunizieren. Die Operationen und Datentypen, die das Protokoll beschreibt, werden in diesem Kapitel erklärt.
Die Spezifikation wurde von der TNG in zwei Dokumenten verfasst: *TNC IF-MAP Binding for SOAP* [Tru10a] und *TNC IF-MAP Metadata for Network* [Tru10b]. Diese beiden Dokumente bilden die Grundlage für dieses Kapitel.

3.1. Übersicht

Das IF-MAP-Protokoll stützt sich auf bekannte Internet-Technologien: SOAP 2.4.2 und XML-Schema 2.4.1. Daten und Operationen werden in einem XML-Schema beschrieben und als Nachrichten über das SOAP-Protokoll ausgetauscht.

3.2. Transport der Nachrichten

IF-MAP benutzt SOAP als Transportprotokoll. Alle Nutzdaten werden im Rumpf der SOAP-Nachricht verschickt. IF-MAP packt keine Daten in den Header.

3.3. Sessions

Jeder Nachrichtenaustaustausch wird über Sessions (Sitzungen) abgewickelt. Die MAP-Clients erhalten zum Beginn einer Sitzung (*newSession*) eine eindeutige Session-ID, die bei jeder Anfrage mitge-

schickt werden muss. Eine Session wird nach einer bestimmten Zeit an Inaktivität beendet und die Session-ID wird gelöscht. Weitere Anfragen über diese ID werden dann nicht mehr bearbeitet. Der Client muss eine neue Session-ID anfordern.

3.4. Kommunikationskanäle

Für jede Session darf der Client genau einen synchronen Verbindungskanal (SSRC) zum MAP-Server aufbauen. Synchron bedeutet hier, das der Server sofort nach Bearbeitung der Anfrage eine Antwort schickt. Die meisten Kommandos nutzen diesen Kanal. Optional kann ein Client zusätzlich einen weiteren Kanal öffnen. Dieser ist von asynchroner Natur (ARC), d.h. der MAP-Server muss nicht sofort eine Antwort liefern. Der Client muss solange diese Verbindung offen halten, bis der Server eine Antwort schickt. Das einzige Kommando, das über diesen Kanal abgesetzt werden darf, ist *poll*. Der *Subscribe/Poll*-Mechanismus wird in 3.7.5 erläutert. Für jeden Kanal wird TLS [1] (*Transport Security Layer*), ein Verschlüsselungsprotokoll auf Transportebene, verwendet.

3.5. Authentifizierung

Das Authentifizieren von MAP-Clients erfolgt über das asymetrische Public-Key-Verfahren[Wika]. Dieses Verfahren setzt voraus, dass der Empfänger der Nachricht den öffentlichen Schlüssel des Senders besitzt.
Zusätzlich kann die *Basic Authentication*[Gro] benutzt werden: Hier muss sich der Client beim Server mit Benutzername/Password authentifizieren.

[1] `http://tools.ietf.org/rfcmarkup/5246`

3.6. Datentypen

Zwei Datentypen werden in IF-MAP spezifiziert: Identifier und Metadaten. Eine Beziehung zwischen zwei Identifern wird Link genannt. Alle Datentypen werden in XML-Schema als komplexe Datentypen (*complexType*) definiert, d.h. sie setzen sich aus einfachen Datentypen zusammen (siehe dazu Kapitel 2.4.1).

3.6.1. Identifier

Ein Identifier ist ein eindeutiger Wert aus einem Pool von vielen Werten. Jeder Identifier ist typisiert und kann nur bestimmte Werte annehmen. Beispielsweise präsentiert ein Identifier vom Typ *MAC-Adresstype* eine MAC-Addresse, die nur Werte der Form 6x8 Bit, jeweils getrennt durch Bindestriche, annehmen kann. Derzeit sind fünf verschiedene Typen von Identifiern spezifiziert [Tru10a]:

- **access-request** Repräsentiert einen Endpunkt, der Zugang zum Netzwerk anfordert
- **device** Ein logisches Endgerät im Netzwerk
- **identity** Eine Identität im Netzwerk. Das könnten Benutzer, Endgeräte oder auch Applikationen sein.
- **ip-address** Bezeichnet eine IP-Adresse
- **mac-address** Bezeichnet eine MAC-Adresse

Jeder Identifier wird durch eine XML-Signatur definiert (siehe Anhang A.2). Die Identifier sind in der Spezifikation [Tru10a] näher erläutert.

3.6.2. Links

Ein Link ist eine Beziehung zwischen zwei Identifiern. Er ist dadurch gekennzeichnet, dass er immer Daten(=Metadaten) enthält. Ein Link verbindet nicht beliebige Identifier miteinander, sondern

jeder Typ darf nur zwei, von der TNG empfohlene Identifier-Typen verbinden. Eine Beziehung kann vom einem MAP-Client gesetzt werden. Abbildung 3.1 zeigt Identifier und Beziehungen mit ihren Metadaten, wie sie in einem MAP-Server auftauchen könnten.

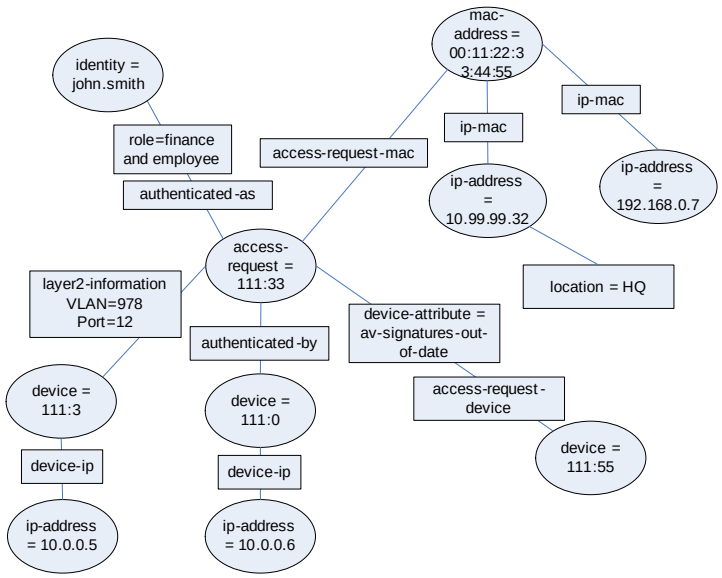


Abbildung 3.1.: Ein MAP-Server verwaltet Metadaten in einer Graphenstruktur [Tru10a, S.17 Abbildung 2]

Die Identifier werden als Ovale dargestellt, ein Link ist eine Linie zwischen zwei Identifiern. Rechtecke repräsentieren Metadaten und sind entweder einem Link oder einem Identifier zugeordnet.

3.6.3. Metadaten

Metadaten können einem Link oder Identifier hinzugefügt werden. Es gibt verschiedene Metadata-Typen die, wie die Identifier, in XML-Schema definiert sind (Anhang A.3. Einige Metadaten werden nur einem Identifier zugeordnet. Folgende Tabelle zeigt die aktuell spezifizierten Metadaten und ihre möglichen Beziehungen zu den Identifiern [Tru10b]:

Metadaten	Identifier 1	Identifier 2
access-request-device	access-request	device
access-request-ip	access-request	ip-address
access-request-mac	access-request	mac-address
authenticated-as	access-request	identity
authenticated-by	access-request	device
capability	access-request	-
device-attribute	access-request	device
device-characteristic	access-request, ip-address, mac-address	device
device-ip	device	ip-address
discovered-by	ip-address, mac-address	device
enforcement-report	ip-address, mac-address	device
event	access-request, identity, ip-address, mac-address	-
ip-mac	ip-address	mac-address
layer2-information	access-request	device
location	identity, ip-address, mac-address	-
request-for-investigation	ip-address, mac-address	device
role	access-request	identity
unexpected-behavior	access-request, identity, ip-address, mac-address	-
wlan-information	access-request	device

Es gibt zwei Arten von Meta-Daten: die oben aufgeführten Standard-Metadaten, die in [Tru10b] beschrieben sind, und selbstdefinierte Meta-Datentypen. Ein MAP-Server muss beide Arten von Metada-

ten unterstützen.

3.6.4. Lebensdauer von Metadaten

Metadaten können unterschiedliche Lebensdauern haben, die beim Veröffentlichen mittels *publishUpdate* festgelegt werden. Dazu dient das Attribut *lifetime*. Mögliche Werte sind *forever* und *session*. *forever* bedeutet, dass die Metadaten für unbegrenzte Zeit gespeichert werden. Metadaten, die mit *lifetime=session* veröffentlicht wurden, werden nach Beendigung einer Sitzung automatisch vom MAP-Server gelöscht. Wird das Attribut nicht angegeben, wird die Lebenszeit standardmäßig auf *session* gesetzt.

3.6.5. Spezielle Attribute von Metadaten

Metadaten werden vom MAP-Server immer mit folgenden zusätzlichen Attributen gespeichert:

ifmap-publisher-id
Die *ifmap-publisher-id* ist mit einem bestimmten MAP-Client assoziiert, der diese Meta-Informationen veröffentlicht hat. Die Zuordnung eines Clients zu einer bestimmten *ifmap-publisher-id* ist implementierungsabhängig.

ifmap-timestamp
Der Zeitpunkt, an dem diese Meta-Informationen veröffentlicht wurden.

ifmap-cardinality
Dieses Attribut muss vom Client zwingend angegeben werden. Mögliche Werte sind *singleValue* oder *multiValue*. Veröffentlicht ein Client Metadaten mit *singleValue*, und es existieren bereits Metadaten des gleichen Typs für denselben Identifier oder Link, so werden alle vorigen Daten gelöscht und durch die neuen Metadaten ersetzt.

Ist der Wert *multiValue*, so werden die alten Metadaten um die Neuen ergänzt.

3.7. Operationen

3.7.1. Überblick

IF-MAP beschreibt verschiedene Funktionen zur Manipulation von Metadaten, die ein MAP-Server implementieren muss. Ein MAP-Client kann Metadaten veröffentlichen, löschen und nach ihnen suchen. Einen Überblick der Operationen, die in IF-MAP spezifiziert wurden, zeigt folgende Tabelle:

Operation	Beschreibung
newSession	Beginnt eine neue Session. Man erhält eine eindeutige *session-id*
publish update	Veröffentlichen von Metadaten auf dem MAPS.
publish notify	Wie *publish update*, allerdings werden die veröffentlichten Metadaten nicht persistent gespeichert, sondern stehen exklusiv den *Subscribern* zur Verfügung. Es kann nicht explizit nach diesen Metadaten gesucht werden, sondern ausschließlich über ein *poll* bezogen werden.
publish delete	Löschen von Metadaten auf dem MAPS.
subscribe update	Ein MAPC kann sich beim MAPS für ein bestimmtes Ereignis registrieren.
subscribe delete	Ein MAPC kann vorige Abbonements auf Statusänderungen löschen
poll	Wenn ein Client ein *subscribe* abgesetzt hat, muss er anschließend eine *poll*-Anfrage stellen, um Änderungen vom MAP-Server abzuholen.
purgePublisher	Löschen von sämtlichen Meta-Informationen eines Publishers.

search	Suchen von Metadaten auf dem MAPS.
endSession	Beendete eine Session
renewSession	Eine Verbindung kann in regelmäßigen Intervallen erneuert werden.

Der nächste Abschnitt beschreibt die oben erläuterten Operationen im Detail und zeigt exemplarisch eine Sitzung zwischen einem MAP-Server und MAP-Client.

3.7.2. Neue Sitzung beginnen

Bevor der Client Anfragen an den MAP-Server stellen kann, muss er sich beim Server authentifizieren (siehe dazu Kapitel 3.5). Ist das erfolgreich, fordert der Client eine neue Sitzung an. Dazu dient die *newSession*-Anfrage, die unbedingt nach der Authentifizierung aufgerufen werden muss. Die Antwort des MAP-Servers enthält eine *ifmap-publisher-id* und *session-id*. Jeder MAP-Client hat eine eindeutige *ifmap-publisher-id*, die bei jeder Sitzung gleich bleibt (siehe hierzu auch 3.6.5). Die *session-id* hingegen verfällt am Ende einer Sitzung und der Client erhält bei jedem *newSession* eine neue ID. Die *session-id* muss bei jeder Operation als Attribut mit angegeben werden.

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:newSession/>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 3.1: Eine *newSession*-Anfrage beginnt eine neue Sitzung

```
<ns3:Envelope xmlns:ns2="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
xmlns:ns3="http://schemas.xmlsoap.org/soap/envelope/">
   <ns3:Header/>
   <ns3:Body>
      <ns2:response>
         <newSessionResult ifmap-publisher-id="5F9831D1B446B7EBDA9E4E45F684DCD8"
session-id="A4058EB410B142F0CCF9A710DCE66BF8"/>
```

```
      </ns2:response>
   </ns3:Body>
</ns3:Envelope>
```

Listing 3.2: Eine erfolgreiche *newSession*-Anfrage liefert als Antwort eine *ifmap-publisher-id* und eine *session-id*

3.7.3. Daten veröffentlichen / löschen

Nach einem erfolgreichen *newSession* kann ein MAP-Client Manipulationen an den Metadaten vornehmen. Dazu dient die *publish*-Anfrage, die in drei Varianten verwendet werden kann: *update*, *notify* und *delete*.

update
Metadaten werden im MAP-Server gespeichert oder ersetzt.

notify
Veröffentlicht ebenfalls Metadaten, mit dem Unterschied, dass diese Informationen nur an die Subscriber weitergereicht werden und nicht in die eigentliche Datenstruktur aufgenommen werden. Die Informationen können nur durch ein *poll* erfasst werden, nicht aber durch eine Such-Operation.

delete
Löscht Datensätze aus dem MAPS.

Jede dieser Operationen kann mehrfach in beliebiger Reihenfolge in einer *publish*-Anfrage untergebracht werden. Die Parameter sind immer Identifier und Metadaten. Die *session-id* muss zwinged angegeben werden.
Im folgenden wird eine IP und eine MAC-Adresse zusammen mit Meta-Informationen veröffentlicht:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
```

```
   <soapenv:Header/>
   <soapenv:Body>
      <ns:publish session-id="77D73D64BC94F763C427C6A156D4D300">
         <update lifetime="forever">
            <ip-address type="IPv4" value="182.178.0.111"/>
            <mac-address value="00:66:16:56:37:98"/>
      <metadata>
        <ip-mac ifmap-cardinality="singleValue"
        ifmap-publisher-id="825DFA25A4D685B104A95C6AEFB1DEDA"
        ifmap-timestamp="2010-04-20T12:00:05Z">
          <start-time value="2010-04-20T12:00:05Z"/>
          <end-time value="2012-06-20T12:00:05Z"/>
          <dhcp-server value="10.0.0.1"/>
        </ip-mac>
      </metadata>
         </update>
      </ns:publish>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 3.3: Ein *ip-mac*-Link und die zugehörigen Metadaten verbinden eine IP-Adresse mit einer MAC-Adresse

```
<ns3:Envelope xmlns:ns2="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
xmlns:ns3="http://schemas.xmlsoap.org/soap/envelope/">
   <ns3:Header/>
   <ns3:Body>
      <ns2:response>
         <publishReceived/>
      </ns2:response>
   </ns3:Body>
</ns3:Envelope>
```

Listing 3.4: Eine erfolgreiche *publish*-Anfrage wird mit einer *publishReceived*-Nachricht quitiert.

3.7.4. Daten suchen

Um nach Daten zu suchen, wird die *search*-Abfrage benutzt. Für die Suche muss man einen beliebigen Identifier, der bereits Metadaten besitzt, als Startpunkt wählen. Es gibt keine Möglichkeit, ohne Angabe eines Identifiers nach Daten zu suchen. Mit den Attributen von *search* kann man das Suchergebnis in der Anzahl einschränken und nach bestimmten Metadaten filtern. Ohne Einschränkungen arbeitet der Algorithmus rekursiv den Graphen ab, bis alle Blätter erreicht sind und liefert alle gefundenen Metadaten als Ergebnis. Die Suche wird bei dem angegebenen Identifier begonnen. Eine beispielhafte Suchanfrage nach Metadaten einer IP-Adresse ist in Abbildung 3.5

und 3.6 zu sehen.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ns="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
   <soapenv:Header/>
   <soapenv:Body>
    <ns:search session-id="4FCB3871FB9EFBB96DA020666FDE5CF2">
    <ip-address type="IPv4" value="192.168.0.23"/>
  </ns:search>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 3.5: Die Suche wird bei der IP von Typ *IPv4* mit Wert *192.168.0.23* begonnen.

```
<ns3:Envelope xmlns:ns2="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
xmlns:ns3="http://schemas.xmlsoap.org/soap/envelope/">
   <ns3:Header/>
   <ns3:Body>
      <ns2:response>
         <searchResult>
            <resultItem>
               <ip-address type="IPv4" value="192.168.0.23"/>
            </resultItem>
            <resultItem>
               <mac-address value="00:66:11:56:37:98"/>
            </resultItem>
            <resultItem>
               <ip-address type="IPv4" value="192.168.0.23"/>
               <mac-address value="00:66:11:56:37:98"/>
               <metadata>
                  <ip-mac ifmap-cardinality="singleValue"
                          ifmap-publisher-id="5F9831D1B446B7EBDA9E4E45F684DCD8"
                        ifmap-timestamp="2010-05-30T19:30:46+0200">
                     <start-time value="2010-04-20T12:00:05Z"/>
                     <end-time value="2012-06-20T12:00:05Z"/>
                     <dhcp-server value="192.168.0.1"/>
                  </ip-mac>
               </metadata>
            </resultItem>
         </searchResult>
      </ns2:response>
   </ns3:Body>
</ns3:Envelope>
```

Listing 3.6: Die Antwort auf die Suche enthält alle erreichten Identifier und ihre Metadaten

In folgender Tabelle sind alle möglichen Attribute der search-Operation erläutert:

Attribut-Name	Beschreibung
match-links	Legt die Kriterien für einen erfolgreichen Treffer im Graphen fest. Irrelevante Identifier und Links können somit übersprungen werden.
max-depth	Definiert die maximale Tiefe der Suche. Die Anzahl bestimmt, wieviel Links der nächste Identifier vom Startpunkt höchstens entfernt sein darf, um in der Suche eingeschlossen zu werden.
max-size	Die maximale Größe des Ergebnises in Bytes
result-filter	Zum Schluss kann das Resultat noch gefiltert werden. Bestimmte Metadaten werden aus der Ergebnismenge entfernt.
terminal-identifier	Legt Identifier fest, bei der die Suche beendet wird. Erreicht der Such-Algorithmus einen Identifier diesen Typs, so wird die Suche abgebrochen und die bisher gefundenen Metadaten zurückgeliefert.

3.7.5. Subscribe / Poll

Möchte ein MAP-Client über Statusänderungen von bestimmten Identifiern informiert werden, so kann er den *subscribe/poll*-Mechanismus nutzen. Dabei abonniert der Client zunächst eine oder mehrere Identifier mit einer *subscribe*-Anfrage. Danach öffnet er einen neuen Kanal, führt eine *poll*-Anfrage aus und wartet auf ein Ergebnis. Der MAP-Server verwaltet eine Liste aller Subscriber und benachrichtigt sie bei Änderungen. Möchte der Client nicht mehr über Änderungen von vorher abonierten Identifiern informiert werden, so kann er dazu ebenfalls die *subscribe*-Operation benutzen.
Eine *subscribe*-Anfrage umfasst *update* und *delete*-Elemente in beliebiger Anzahl und Reihenfolge.

Subscribe update

Hiermit registriert sich ein MAP-Client beim Server für Änderungen auf bestimmte Identifier. Die Anfrage ähnelt der einer Suchanfrage, nur das der Identifier in einem *update*-Element eingeschlossen ist und nicht in einem *search*-Element. Die Attribute, die bei einer *search*-Anfrage verwendet werden können, sind auch bei einem *update* verfügbar. Die Ergebnismenge kann also auch hier beeinflusst werden. Zusätzlich muss jedes *update*-Element einen eindeutigen Namen erhalten, der vom Client festgelegt wird (Attribut *name*). Dieser Name wird für spätere Manipulationen an der *Subscription* gebraucht. Eine gültige Anfrage wird mit einem *subscribeReceived* beantwortet. Eine *Subscription* auf eine IP-Adresse und eine MAC-Adresse könnte demnach wie in Listing 3.7 aussehen.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:subscribe session-id="4FCB3871FB9EFBB96DA020666FDE5CF2">
         <update name="1234" max-depth="2">
           <ip-address type="IPv4" value="192.168.0.23"/>
         </update>
         <update name="abcd" max-depth="2">
           <mac-address value="00:66:11:56:37:98"/>
         </update>
      </ns:subscribe>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 3.7: Zwei Subscriptions auf Identifier mit eindeutigen Namen und Limitierung der Suchtiefe auf 2

Existiert schon eine *Subscription* mit dem gleichen Namen, so wird diese durch die Neue ersetzt.

Subscribe delete

Mit einem *delete*-Element wird ein zuvor abonnierter Identifier entfernt. Eine *Subscription* wird durch seinen Namen eindeutig identifiziert (Listing 3.8).

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:ns="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
   <soapenv:Header/>
```

```
   <soapenv:Body>
      <ns:subscribe session-id="4FCB3871FB9EFBB96DA020666FDE5CF2">
         <delete name="1234"/>
         <delete name="abcd"/>
      </ns:subscribe>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 3.8: Ein Abbonement kann über seinen Namen gelöscht werden

Poll

Schickt man nach einem erfolgreichen *subscribe* eine *poll*-Anfrage und es existieren bereits Metadaten für die abonierten Identifier, so bekommt man sofort eine Antwort vom Server (*pollResult*). Ein erneuter Aufruf von *poll* kehrt erst zurück, wenn es Änderungen an den Metadaten gegeben hat.
Das *poll*-Kommando ist das einzige, das über den ARC ausgeführt wird. Das bedeutet, die Anfrage muss nicht sofort beantwortet werden. Im Gegensatz zu den anderen Kommandos, die über SSRC laufen und sofort eine Antwort liefern, kann ein *poll*-Ergebnis sehr viel später eintreffen. Angemerkt sei an dieser Stelle, das dieses Verhalten Konsequenzen für die Implementierung hat. Ein Aufruf von *poll* kann den weiteren Ablauf des Programms blockieren, weil die *poll*-Anfrage nicht sofort beantwortet werden muss. Der MAP-Client hält die Verbindung über den ARC solange offen, bis eine Antwort da ist. Er kann nebenher weitere Kommandos über den SSRC absetzen, der ARC ist allerdings bis zum Rückkehren der *poll*-Anfrage blockiert. Eine gültige *poll*-Anfrage könnte wie in Listing 3.9 aussehen.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns="http://www.trustedcomputinggroup.org/2010/IFMAP/2">
   <soapenv:Header/>
   <soapenv:Body>
      <ns:poll session-id="4FCB3871FB9EFBB96DA020666FDE5CF2"/>
   </soapenv:Body>
</soapenv:Envelope>
```

Listing 3.9: Eine *poll*-Anfrage

Das Ergebnis gleicht dem einer *search*-Anfrage und wird nach unbestimmter Zeit geliefert (Listing 3.10).

```
<ns3:Envelope xmlns:ns2="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
   xmlns:ns3="http://schemas.xmlsoap.org/soap/envelope/">
   <ns3:Header/>
   <ns3:Body>
      <ns2:response>
         <pollResult>
            <searchResult name="1234">
               <resultItem>
                  <ip-address type="IPv4" value="192.168.0.23"/>
               </resultItem>
               <resultItem>
                  <mac-address value="00:66:11:56:37:98"/>
               </resultItem>
               <resultItem>
                  <ip-address type="IPv4" value="192.168.0.23"/>
                  <mac-address value="00:66:11:56:37:98"/>
                  <metadata>
                     <ip-mac ifmap-cardinality="multiValue"
                     ifmap-publisher-id="5F9831D1B446B7EBDA9E4E45F684DCD8"
                     ifmap-timestamp="2010-06-30T14:50:40+0200">
                        <start-time value="2020-04-20T12:00:05Z"/>
                        <end-time value="2022-06-20T12:00:05Z"/>
                        <dhcp-server value="192.168.0.1"/>
                     </ip-mac>
                  </metadata>
               </resultItem>
            </searchResult>
         </pollResult>
      </ns2:response>
   </ns3:Body>
</ns3:Envelope>
```

Listing 3.10: Eine *poll*-Antwort

Abbildung 3.2 zeigt den *subscribe/poll*-Mechanismus zu verschiedenen Zeitpunkten. Bei *t0* setzt der Client ein *subscribe* mit den zu beobachtenden Identifiern ab. Er bekommt sofort eine Antwort. Der erstmalige *poll*-Aufruf bei *t1* liefert ebenfalls sofort eine Antwort mit den bisherigen Metadaten der Identifier. Ein erneutes *poll* kehrt nicht augenblicklich zurück. Erst zum Zeitpunkt *t2* wird ein *pollResult* zurückgeliefert (es hat Statusänderungen gegeben). Bei *t3* ändern wir unsere Subscriptions und die vorangegangene *poll*-Anfrage liefert uns danach sofort die Ergebnisse.

3.7.6. Löschen von allen Metadaten eines Publishers

Sendet ein Client ein *purgePublish* so werden alle Metadaten, die vorher von diesem Publisher veröffentlicht wurden, gelöscht.

3.7.7. Sitzung aufrecht erhalten

Ein MAP-Server schließt eine Sitzung, wenn länger als drei Minuten keine Aktion von einem Client durchgeführt wird. Um eine Sitzung zu verlängern, muss der Client entweder die TCP-Verbindung auf dem SSRC oder ARC aufrecht erhalten oder in regelmäßigen Abständen ein *renewSession* senden.

3.7.8. Sitzung beenden

Möchte ein MAP-Client eine Sitzung beenden, so kann er ein *endSession* schicken. Die Verbindung wird darauf abgebaut und der MAP-Server löscht alle Metadaten mit *lifetime*-Attribut *session*, die von diesem Publisher veröffentlicht wurden.

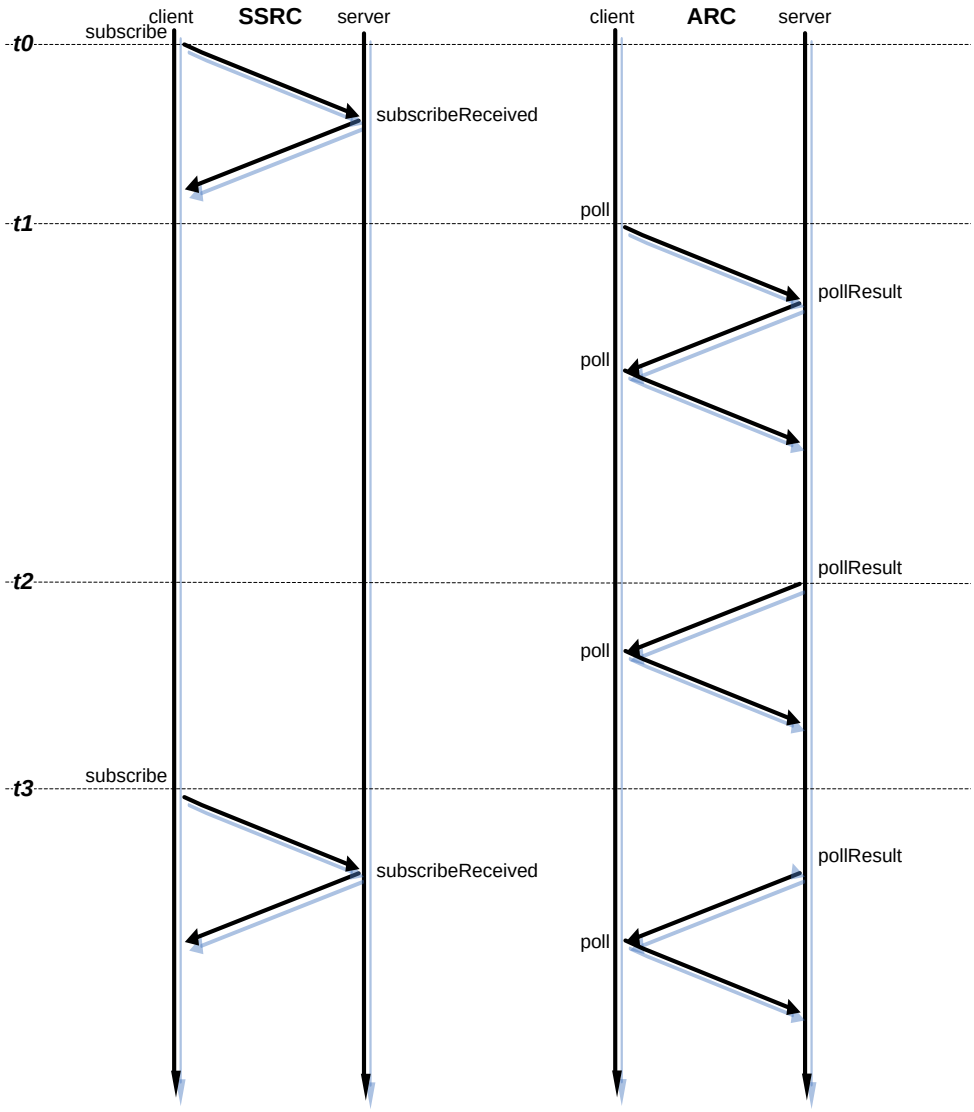


Abbildung 3.2.: Subscribe/Poll-Mechanismus [Tru10a, S.41 Abbildung 4]

4. Analyse des IRON-MAP-Servers

Im Rahmen des IRON[1]-Projektes der FHH, wurde eine MAP-Server Implementierung entwickelt und verschiedene Netzwerkkomponenten um IF-MAP-Funktionalität erweitert. Dieses Kapitel analysiert den IRON-MAP-Server.

4.1. Implementierungs-Stand

Der IRON-MAPS beherrscht alle wichtigen Operationen, die die IF-MAP-Spezifikation fordert: *newSession*, *purgePublisher*, *renewSession*, *endSession*, *publishUpdate*, *publishDelete*, *publishNotify*, *subscribe*, *poll* und *search*.

4.1.1. Technologien

Der MAP-Server wurde in Java umgesetzt. Für das Umwandeln der XML-Nachrichten in Java-Objekte und zurück wird JAXB-RI verwendet.

4.1.2. Architektur

Der IRON-MAPS setzt sich aus mehreren Schichten zusammen (Abbildung 4.1). Das `Connection Management` verwaltet die Verbindungen, die zum MAP-Server aufgebaut werden und kümmert sich um den Nachrichtenaustausch. Empfangene (XML-)Nachrichten werden in Java-Objekte transformiert. Dieser Prozess wird auch als *unmarshalling* bezeichnet. Der Transformation-Layer wandelt die Java-Objekte in interne Datenmodell-Objekte um. Danach werden

[1] Intelligent Reaction on Network Events

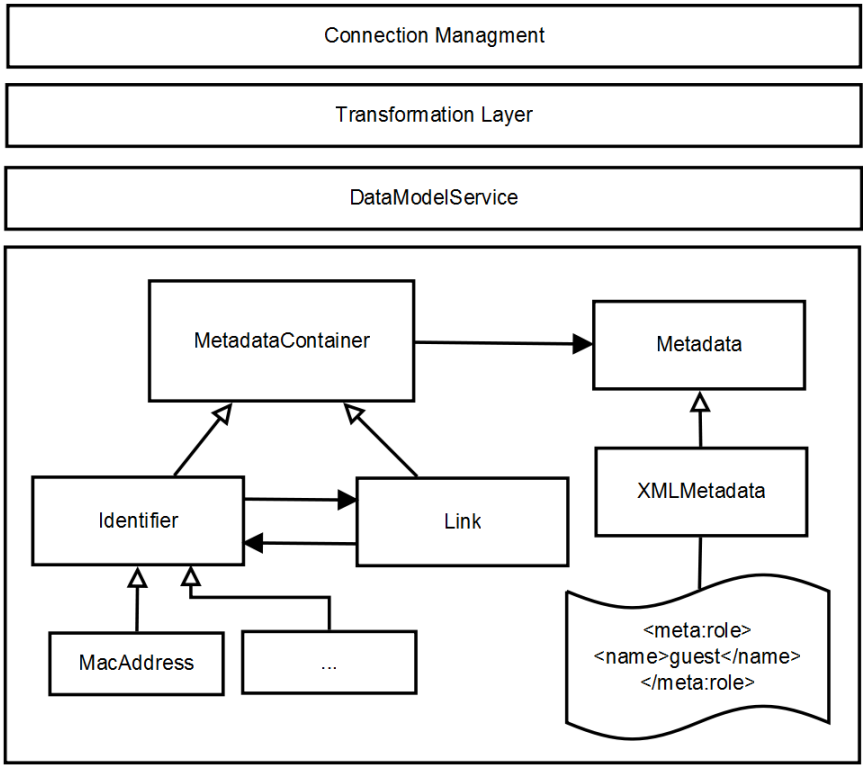


Abbildung 4.1.: Verschiedene Schichten des IRON-MAPS[Val09]

die Informationen einer weiteren Schicht übergeben, dem `Data-model Service`. Jenachdem welche Operation auf die Daten angewandt wird, werden hier die Objekte in der Datenstruktur gespeichert, gelöscht oder gesucht.
Die Daten werden in Containern verwaltet, es existiert ein Container für jeden `Identifier` und ein Container für `Links`. Beide Klassen erben von der abstrakten Superklasse `MetaDataContainer`. `Identifier` ist ebenfalls eine abstrakte Klasse, die konkreten Implementierungen sind die fünf Unterklassen `MacAddress`, `IpAddress`, `AccessRequest`, `Identity` und `Device`. Alle Container können Metadaten vom Typ `XMLMetadata` aufnehmen, welche eine Implementierung der abstrakten Superklasse `Metadata` ist. Zwischen den `Identifiern` und `Links` besteht eine bi-direktionale Beziehung, d.h. ein Identifier kennt seine Links und jeder Link kennt

die zwei beteiligten Identifier.
Alle Identifier und Links werden in Objekten der Klassen `IdentifierRep` bzw. `LinkRep` abgelegt, welche die Daten in einer Liste verwaltet. Diese Klassen werden vom DataModelService instanziert und benutzt.
Der umgekehrte Weg, also der Versand von Nachrichten, ist der gleiche nur in anderer Richtung: Der `Datamodel Service` holt sich die gewünschten Daten aus den Repositories und reicht sie dem `Transformation Layer` nach oben. Dieser wandelt die Daten in JAXB-RI konforme Java-Objekte und übergibt sie dem `Connection Management`. Hier werden die Objekte in IF-MAP konformes XML umgewandelt, auch *marshalling* genannt. Danach wird die Antwort zum Client geschickt.

4.1.3. Authentifizierung

Um die Integrität der Benutzer zu gewährleisten, wird das asymetrische Public-Key-Verfahren verwendet (vgl. Kapitel 3.5). Beim IRON-MAP-Server werden die Schlüssel und Zertifikate in drei Dateien verwaltet: *keystore*, *truststore* und *user.properties*. In dem *keystore* sind die privaten Schlüssel des MAP-Server, seine Zertifikate und die zugehörigen öffentlichen Schlüssel untergebracht. Der *truststore* enthält Zertifikate, die der MAP-Server für vertrauenswürdig hält (also die der MAP-Clients). Die Informationen aus dem *keystore* und dem *truststore* können auch in einer Datei untergebracht werden.
Die *user.properties* enthält Benutzername/Passwort-Kombinationen in Klartext, die für die Basic-Authentifizierung verwendet wird. Der IRON-Map-Server hat bisher kein fein granuliertes Rechtesystem. Jeder authorisierte Client darf alle Operationen uneingeschränkt nutzen.
Wie in Kapitel 3.6.5 erwähnt, ist die Verknüpfung eines Clients mit einer bestimmten *ifmap-publisher-id* implementierungsabhängig. Der IRON-MAP-Server vergibt die IDs nach den Zertifikaten im *truststore*. Unterschiedliche Clients, die das selbe Zertifikat be-

nutzen, bekommen die gleiche *ifmap-publisher-id*.

4.1.4. Persistenz

Der IRON-MAP-Server hält alle Informationen im Arbeitsspeicher, eine weitere Persistenzschicht gibt es nicht. Das bedeutet, dass bei einem Neustart oder Absturz des Servers, alle bisher veröffentlichten Daten verloren gehen.

4.1.5. Datenstruktur

Wie in Abbildung 4.1 zu sehen ist, sind die Klassen `Identifier` und `Link` eine Spezialisierung von `MetadataContainer`. Beide nutzen die `XMLMetadata`-Klasse, um die Metadaten zu halten. Diese Klasse ist von `Metadata` abgeleitet. Die `Identifier`-Klasse ist außerdem eine Generalisierung der verschiedenen Identifier-Typen (MacAdress, IPAdress, etc.). `Identifier` und `Link` haben eine bi-direktionale Beziehung. Ein Identifier kennt seine Links und jeder Link kennt die beteiligten Identifier.

4.1.6. Installation

Der MAP-Server ist als ausführbares JAR-Archiv verfügbar, eine Installation ist deshalb nicht notwendig.

4.1.7. Konfiguration

Eine Konfigurationsdatei mit Namen `ifmap.properties` liegt dem MAP-Server bei, die bei Bedarf angepasst werden muss. Die wichtigsten Konfigurations-Parameter sind in folgender Tabelle aufgeführt:

Parameter	**Bedeutung**
CredentialAuthPort	TCP-Port für die Verbindung
keystorePath	Pfad zum Keystore
keystorePW	Passwort für den Keystore

truststorePath	Pfad zum Truststore
truststorePW	Passwort für den Truststore

Die weiteren Konfigurationsdetails können der *ifmap.properties* entnommen werden.

5. Konzept zur Visualisierung der Metadaten eines MAP-Servers

5.1. Übersicht

Nachdem in den vorigen Kapiteln die grundlegene Funktionsweise eines MAP-Servers und das IF-MAP-Protokoll erklärt wurden, folgt nun ein Konzept zur Visualisierung der MAP-Server-Daten. Dabei werden zuerst die Anforderungen an den Client und an den MAP-Server festgelegt. Bezogen auf diese Anforderungen werden verschiedene Lösungsansätze diskutiert und dessen Implementierungs-Möglichkeiten vorgestellt.

5.2. Anforderungen an den Client

Folgende Anforderungen werden an den Client gestellt:

Auswahl der Daten
Es soll die Möglichkeit bestehen, alle Daten eines MAP-Servers vollständig darzustellen.

Datengewinnung
Der Client muss die Daten aus dem MAP-Server auslesen. Das sollte auch über Netzwerkgrenzen hinweg funktionieren und dabei möglichst effizient sein. Sicherheitsaspekte müssen berücksichtigt werden.

Darstellung der Daten
Die Metadaten sollen anschaulich dargestellt werden. Dabei soll die

Graphenstruktur, die Identifier und Links bilden, abgebildet werden.

Erweiterbarkeit
Der Client soll leicht um neue Visualisierungskomponenten erweiterbar sein.

Plattformunabhängigkeit
Der Client soll nicht auf ein Betriebssystem festgelegt werden, sondern plattformunabhängig verfügbar sein.

5.3. Anforderungen an den MAP-Server

Auslesen der Metadaten
Der MAP-Server muss die Möglichkeit bieten, auf sämtliche Daten zuzugreifen. Auch ohne Kenntnis eines bisher verlinkten Identifier muss es möglich sein, an die gewünschten Meta-Informationen zu gelangen. Muss der Server angepasst werden, sollte der Aufwand dafür gering sein.

Sicherheit
Die Möglichkeit an alle Daten des MAP-Servers zu gelangen, birgt ein Sicherheitsrisiko in sich. Verschlüsselung und Authentifizierung müssen daher geboten sein.

5.4. Lösungsansätze

5.4.1. Datengewinnung

Folgende fünf Ansätze wurden zur Gewinnung der Daten untersucht:

- Einführung einer neuen Schnittstelle
- Auswertung der Log-Datei
- Nutzung der vorhandenen Schnittstelle (*search*)

- Nutzung der vorhandenen Schnittstelle (*subscribe/poll*)
- Erweiterung des IF-MAP Protokolls

Einführung einer neuen Schnittstelle

Das IF-MAP-Protokoll könnte durch die Einführung einer neuen Schnittstelle (beispielsweise über eine TCP/IP-Verbindung) komplett umgangen werden. Auch könnten vorhanden Technologien wie RMI [Orab] oder MOM [SUN] genutzt werden.

Vorteile

- Freiheit in der Wahl der neuen Schnittstelle
- Die Schnittstelle kann auf Performanz ausgelegt werden

Nachteile

- Erheblicher Aufwand für Entwurf und Implementierung
- Sicherheitskonzepte müssen selbst entwickelt werden (Verschlüsselung der Daten / Benutzer-Authorisierung)
- Vorhandenen Technologien sind nicht immer für alle Plattformen verfügbar

Auswertung der Log-Datei

Dieser Ansatz bezieht sich speziell auf die IRON-Implementierung des MAP-Servers. Dieser schreibt detailierte Informationen über seine Datenstrukur und Benutzer in eine Log-Datei, die man auslesen könnte.

Vorteile

- Der MAP-Server muss nicht angepasst werden

Nachteile

- Die Debug-Ausgaben dienen nur zur Entwicklung des IRON-MAP-Servers und können sich jederzeit ändern (Andere Implementierungen könnten auf Log-Ausgaben sogar gänzlich verzichten).
- Der Client, oder zumindest ein Teil von ihm, muss Zugriff auf die Datei bekommen. Entweder der Client läuft auf der gleichen Maschine wie der MAP-Server oder zusätzliche Maßnahmen zur Bereitstellung der Datei im Netzwerk müssen ergriffen werden.

Nutzung der vorhandenen Schnittstelle (*search*)

Die *search*-Operation bietet eine Möglichkeit, um an die gewünschten Metadaten zu kommen. Über die Suche kommt man auch an bisher unbekannte Identifier, nämlich solche, die mit dem angegebenen Identifier über Links verbunden sind. Der Such-Algorithmus arbeitet rekursiv jeden Identifier ab, bis der komplette Graph durchlaufen ist und liefert das Ergebnis zurück.
Mit einer Suchanfrage kann man also einen kompletten Graphen in der Datenstruktur durchlaufen. Es kommt aber vor, dass mehrere, untereinander nicht verbundene Graphen existieren. Dann reicht eine Anfrage nicht mehr aus. Man muss mindestens einen Identifier aus den anderen Graphen kennen. Die Anzahl der Suchanfragen, um an alle Graphen zu gelangen, ist äquivalent zur Anzahl der Graphen: N Graphen = N Suchanfragen. Voraussetzung ist, dass die Suche ohne Einschränkung gemacht wird, damit wirklich alle Knoten besucht wurden. Die Suche muss zudem in regelmäßigen Intervallen geschehen, damit Änderungen an der Graphenstruktur zeitnah bemerkt werden.

Vorteile

- Der MAP-Server muss nicht angepasst werden und funktioniert mit allen MAPS-Implementierungen, die das IF-MAP-Protokoll korrekt umgesetzt haben.

- Die Infrastruktur des MAP-Server kann genutzt werden (Authentifizierung/Verschlüsselung)

Nachteile

- Man muss mindestens einen Identifier aus jedem Graphen kennen, damit man die Suche starten kann. Kennt man keinen Identifier aus einem Graphen, so ist es nicht möglich an die Metainformationen zu gelangen.
- Die *search*-Anfrage durchläuft den ganzen Graphen, also auch Identifier und Links, die sich nicht geändert haben. Das kostet unötig Zeit und Resourcen. Auch eine Suchanfrage mit einer Limitierung der Suchtiefe (*max-depth*) hilft hier nicht weiter, da für jeden nicht erreichten Identifier trotzdem eine weitere Anfrage gestellt werden muss. Somit wird der Baum ebenfalls komplett durchlaufen, nur eben mit mehreren Suchanfragen.

Nutzung der vorhandenen Schnittstelle (*subscribe/poll*)

Eine weitere Möglichkeit, mit Bordmitteln des MAP-Servers Metadaten zu gewinnen, bietet der *Subscribe/Poll*-Mechanismus. Dabei wird auf jeden Identifier eine *Subscribtion* gemacht und anschließend das Ergebnis gepollt.

Vorteile

- Der Client wird bei Statusänderungen sofort benachrichtigt. Auch werden nicht alle verfügbaren Metadaten und Identifier übertragen, sondern nur die Änderungen.
- Eine regelmäßige Suche nach neuen Änderungen der abonierten Identifier ist nicht erforderlich. Die Benachrichtigung erfolgt durch einen (asynchronen) *poll*-Aufruf.

Nachteile

- Wie in 5.4.1, kann man ohne Kenntnis eines Identifiers keine Metadaten abfragen.

Erweiterung des IF-MAP Protokolls
Die Problematik aus den letzten beiden Ansätzen (das Wissen über Identifier), kann durch eine Erweiterung der IF-MAP-Schnittstelle behoben werden. Ohne Kenntnis der vorhandenen Datenstruktur des MAP-Servers sollte es mit der Erweiterung möglich sein, eine Liste mit den verlinkten Identifiern abzurufen (ohne ihre Metadaten). Dies kann erreicht werden, indem man eine neue Operation einführt oder eine bestehende erweitert. Der Client kann nun entscheiden, welche Metadaten er abfragen möchte. Diese Identifier werden dann mit regulären IF-MAP-Operationen verwendet, wie in 5.4.1 beschrieben. Damit nicht jedesmal eine komplette Liste der Identifier abgerufen werden muss, sollte die neue/erweiterte Operation die Möglichkeit bieten, sich zu informieren, ob es überhaupt Statusänderungen gegeben hat. Ist dies der Fall, kann die Liste mit Identifiern bezogen werden.

Vorteile

- Man nutzt die Infrastruktur des MAP-Server: Authentifizierung und Session-Management ist wie bei jedem anderen MAP-Client. Aus Sicht des Servers verhält sich der Client wie ein regulärer MAP-Client

- Die Suche und der Benachrichtigungs-Mechanismus sind im MAP-Server schon implementiert, deshalb muss der Server nur minimal erweitert werden.

Nachteile

- Eine bestehende Operation zu erweitern, kann zu Verwirrung führen: Die Funktionalität ist nicht mehr wie in IF-MAP spezifiziert, sondern verhält sich anders.

- Eine Operation, die es ermöglicht, sämtliche Metadaten abzufragen, könnte ein Sicherheitsrisiko darstellen.

5.4.2. Darstellung der Daten

Es müssen alle verlinkten Identifier und ihre zugehörigen Metadaten aus dem MAP-Server gelesen werden und anschließend visuell aufbereitet werden. Dabei soll eine saubere Trennung von Darstellung und Datengewinnung der Daten erzielt werden. Vorhandene Visualisierungs-Bibliotheken werden eingesetzt, um die Daten attraktiv darzustellen.

5.4.3. Erweiterbarkeit des Clients

Die Visualisierung sollte nicht auf einen bestimmten Typ limitiert sein, denn Daten können auf verschiedenste Weise dargestellt werden. Deshalb sollte der Client so konzipiert werden, dass es leicht möglich ist, die implementierten Visualisierungskomponenten zu erweitern. Neue Komponenten sollten sich ebenfalls leicht integrieren lassen.

5.4.4. Plattformunabhängigkeit

Damit der Client nicht auf eine Plattform beschränkt ist, sollten die verwendete Programmiersprache und Technologien plattformunabhägig sein.

5.4.5. Bewertung

Für die vorgestellten Lösungsansätze zur Datengewinnung wird nun eine Entscheidung getroffen, welche Möglichkeit für eine Implementierung in Frage kommt.
Der Ansatz, die IF-MAP Schnittstelle ohne Anpassung zu nutzen, kann so nicht realisiert werden. Da ein Client vorher nicht weiß, welche Identifier mit Metadaten versehen sind, müsste er alle Identifier,

die existieren könnten, suchen oder abonieren. Da die Menge der möglichen Identifier viel zu groß ist, scheidet dieser Ansatz aus.
Die Einführung einer neuen Schnittstelle bietet die größt mögliche Freiheit und Flexibilität. Allerdings ist der Aufwand, das zu realisieren, sehr groß: Datentypen und Operationen müssen entworfen, Sicherheitskonzepte entwickelt und implementiert werden. Die Verwendung von vorhandenen Technologien hat den Nachteil, dass man in der Regel an eine Plattform gebunden ist oder sogar eine Middleware benötigt.
Die Auswertung der Log-Datei kommt nicht in Frage, da die genannten Nachteile schwerwiegend sind. Es ist nicht tragbar, dass eine Visualisierung auf dem selben Rechner wie der MAP-Server laufen muss. Möchte man über das Internet die Datei auslesen, müsste man sogar einen Webserver oder anderen Dienst dafür bereitstellen. Bei manchen Implementierungen wird das Vorgehen gar nicht möglich sein, weil sie ihre Informationen nicht in eine Log-Datei schreiben.
Die Erweiterung des IF-MAP-Protokolls scheint die sinnvollste Lösung zu sein. Sieht man von der Änderung einer bereits existierenden Operation ab, so ist die Einführung eines neuen Kommandos die sauberste Lösung. Die Funktionalität der vorhandenen Operationen bleibt gewährleistet und wird nicht verändert. Stattdessen zeigt ein neues Kommando, dass der MAP-Server um eine weitere Funktionalität ergänzt wurde. Mit dieser Operation in Kombination mit den vorhandenen Funktionen, ist es kein Problem mehr, an alle Metadaten zu gelangen. Außerdem ist die Nutzung des Authentifizierungs-Mechanismus und Session-Management des MAP-Servers von Vorteil. Denkbar wäre eine fein-granulare Benutzerverwaltung, die diese neue Funktionalität nur für bestimmte Clients zulässt.

5.5. Implementierungsansätze

Dieses Kapitel beschreibt die Implementierungsansätze der Konzepte aus Kapitel 5.4.1, 5.4.2 und 5.4.3.

5.5.1. Erweiterung des IRON-MAP-Servers

Diese Sektion beschäftigt sich mit der oben erläuterten Möglichkeit, die IF-MAP-Schnittstelle gemäß dem gewählten Lösungsansatz zu erweitern.

Neue Operation

Wie in Kapitel 4.1.1 beschrieben, sind einige Klassen, darunter die IF-MAP-Operationen und Datentypen in XML-Schema beschrieben und mit JAXB-RI generiert. Deshalb müssen die zugrundeliegenden XSD-Dateien angepasst werden.
Dem MAP-Server wird ein neuer Anfragetyp hinzugefügt, damit eine Liste mit allen bisher verlinkten Identifieren abgefragt werden kann (Abbildung 5.1). Die neue Operation wird durch die folgenden XSD-Signaturen definiert:

```
<xsd:element name="dump" type="DumpRequestType"/>
```

Listing 5.1: Neues Element zufügen

Definiert die neue Operation mit Namen *dump*. Der Typ *DumpRequestType* wird wie folgt spezifiziert:

```
<xsd:complexType name="DumpRequestType">
    <xsd:attributeGroup ref="sessionAttributes"/>
  <xsd:attribute name="identifier" type="xsd:string"/>
</xsd:complexType>
```

Listing 5.2: Neuen Typ zufügen

Die Attribut-Gruppe *sessionAttributes* beinhaltet lediglich das *session-id*-Attribut:

```
<xsd:attributeGroup name="sessionAttributes">
    <xsd:attribute name="session-id" type="xsd:string"
     use="required"/>
</xsd:attributeGroup>
```

Listing 5.3: IF-MAP: sessionAttributes

Die Anfrage fordert zwingend das *session-id*-Attribut, das bei jeder Anfrage mitgeschickt werden muss.
Das Attribut *identifier* schränkt die Liste von Identifiern ein und filtert unerwünschte Identifier-Typen aus der Ergebnismenge. Mögliche Werte sind:

- ***** - liefert eine komplette Liste aller verlinkten Identifier

- **IDENTIFIER 1[,IDENTIFIER N]** - liefert nur Identifier, die von einem bestimmten Typ sind. Mehrere Identifier-Typen können durch ein Komma getrennt angegeben werden. Typen, die nicht im MAP-Server existieren, werden ignoriert.

- **-** - liefert ein leeres Resultat. Ein leeres Resultat wird ebenfalls zurückgeliefert, falls keine Metadaten auf dem MAP-Server existieren.

Wird das Attribut *identifier* weggelassen, so ist das äquivalent zu *identifier='*'*.
Die Antwort des MAP-Servers muss ebenfalls in der XSD-Datei festgelegt werden:

```
<xsd:complexType name="DumpResponseType">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
  <xsd:element name="access-request" type="AccessRequestType"/>
  <xsd:element name="identity" type="IdentityType"/>
  <xsd:element name="ip-address" type="IPAddressType"/>
  <xsd:element name="mac-address" type="MACAddressType"/>
  <xsd:element name="device" type="DeviceType"/>
 </xsd:choice>
 <xsd:attribute name="last-update" type="xsd:long"/>
</xsd:complexType>
```

Listing 5.4: Neuer Rückgabetyp

Das Resultat ist eine Liste mit Identifiern und das *last-update*-Attribut enthält einen Zeitstempel mit der letzten Änderung der Datenstruktur des MAPS. Gibt es keine Metadaten auf dem MAPS oder wurde die Anfrage mit *identifier='-'* gestellt, so ist das Ergebnis leer und *last-update* ist 0.

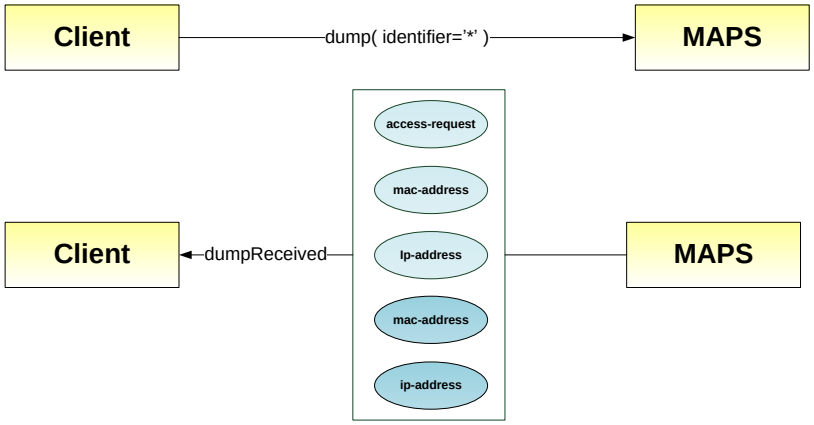


Abbildung 5.1.: *dump* liefert eine Liste mit verlinkten Identifiern

Statusänderungen: Das *last-update* Attribut

Eine *dump*-Antwort enthält immer das Attribut *last-update* mit dem letzten Aktualisierungsdatum als Zeitstempel. Werden auf dem MAP-Server Daten gelöscht, geändert oder sind neu dazu gekommen, so wird der Zeitstempel aktualisiert.
Ein MAP-Client kann regelmäßig eine *dump*-Anfrage mit Attribut *identifier="-"* stellen, um sich über Statusänderungen im MAPS zu informieren. Dazu wird *last-update* ausgewertet. Ist die Zeit zwischen zwei *dump*-Anfragen unterschiedlich, weiß der Client, dass es Modifikationen an der Datenstruktur des MAPS gegeben hat und kann ggf. Suchanfrage(n) starten oder sich eine aktuelle Liste mit Identifiern holen. Somit spart man einige Anfragen, da bei zwei identischen Zeitstempeln keine Suche angestoßen werden muss.

Statusänderungen einzelner Identifier

Statt jedesmal die komplette Graphenstruktur auszulesen, kann auch der *subscribe/poll*-Mechanismus genutzt werden, um über Statusänderungen von einzelnen Identifiern informiert zu werden. Dazu muss auf jeden Identifier, der in einer *dump*-Anfrage enthalten ist, eine *Subscription* gemacht und danach gepollt werden. Damit spart man die Suche durch den kompletten Baum, da man die Metadaten nicht über eine *search*-Anfrage bekommt, sondern durch ein *poll*. Außer-

dem müssen nicht mehr alle Metadaten erneut verarbeitet werden, sondern nur solche, die sich geändert haben.

5.5.2. Implementierungsansätze für den Client

Nachdem in Abschnitt 5.2 die Anforderungen an einen Client festgelegt und entsprechende Lösungsansätze vorgestellt wurden (Kapitel 5.4.2 und 5.4.3), zeigt der folgende Abschnitt Implementierungsansätze für die Entwicklung eines Clients auf. Folgende Design-Entscheidungen werden getroffen:

- **Plattform** Für den Client und die Bibliothek wird Java als Sprache gewählt, da diese Sprache auf den gängigsten Betriebssystemen verfügbar ist. Auf dem Client muss mindestes JRE [1] 1.5 oder höher installiert sein.

- **Modularität** Die Software besteht aus zwei Teilen: Eine Bibliothek und ein Client auf Swing-Basis. Die Bibliothek ist für die benötigten IF-MAP-Operationen und zum Verwalten der Datenstruktur zuständig. Außerdem implementiert sie verschiedene Export-Formate. Die API bietet lesenden Zugriff auf die Daten, schreiben ist nicht möglich. Die Client-Komponente beinhaltet Fenster und Menus zu Darstellung der Daten. Sie benutzt die Bibliothek, um an die Daten zu kommen.

- **Export-Formate** Die Bibliothek soll einfach um weitere Export-Formate erweitert werden können. Ein implementierter Datentyp, der exportiert werden kann, ist ein Graph auf *Prefuse*-Basis [Ber].

- **Benachrichtigungs-Mechanismus** Die Bibliothek sollte die Möglichkeit bieten, seine Clients über Änderungen an der Datenstruktur des MAP-Servers zu informieren. Lösen könnte man dies mit einem Callback-Mechanismus.

[1] Java Runtime Enviroment

Komponenten der Bibliothek

Die Bibliothek setzt sich aus mehreren Schichten zusammen, die jeweils verschiedene Aufgabenbereiche haben:

- Senden und Empfangen der Nachrichten
- Verarbeiten und Speichern der Daten
- Export der Graphenstruktur
- Registrieren für Ereignisse

Konfigurationsdatei

Die Bibliothek erfordert eine Konfigurationsdatei mit folgenden Konfigurationen:

Parameter	Bedeutung
mapserver.url	URL des MAP-Servers
mapserver.keystore.path	Pfad zum Keystore
mapserver.keystore.password	Passwort für den Keystore
mapserver.truststore.path	Pfad zum Truststore
mapserver.truststore.password	Passwort für den Truststore
mapserver.basicauth.enabled	Basic Authentification nutzen
mapserver.basicauth.user	User für Basic Authentification
mapserver.basicauth.password	Passwort für Basic Authentification

Senden und Empfangen der Nachrichten

Dieser Teil kümmert sich um die Kommunikation mit dem MAPS über IFMAP. Schnittstellen für die MAP-Operationen werden zur Verfügung gestellt. Für diesen Zweck wird, wie beim MAP-Server, das Webservice-Framework *AXIS2* verwendet, um die benötigten Stub-Klassen zu generieren. Als Datenbindungsmodell wird ebenfalls *JAXB-RI* gewählt. Gefundene Identifier und Metadaten werden an die Verarbeitungsschicht weitergegeben.

Verarbeiten und Speichern der Daten

Das Abbilden von XML-Nachrichten in Java-Objekte findet hier statt. Nach einem *dump* werden die enthaltenen Metadaten in einer Datenstruktur abgelegt. Dabei können neue Identifier/Metadaten zugefügt und gelöscht werden. Gibt es schon Metadaten für einen Identifier, so wird dieser um die neuen Daten ergänzt. Die Datenstruktur wird der Export-Schicht lesend zu Verfügung gestellt. Das heißt, die Daten können von außen nicht manipuliert werden.

Export der Graphenstruktur

Die Datenstruktur kann auf vielfältige Weise verabeitet werden: Eine Abbildung könnte sowohl als Graph als auch als Tabelle realisiert werden. Diverse Export-Formate können implementiert werden, die auf die gleiche Datenbasis zugreifen (nämlich die von der Verarbeitungsschicht gefüllten Datenstruktur).
Eine Implementierungsmöglichkeit ist die Verwendung von *Prefuse*, einer freien Bibliothek zur Visualisierung von Graphenstrukturen. Sie stellt Funktionen zum Anlegen, Löschen und Manipulieren von Knoten und Kanten bereit und lässt sich einfach in Swing integrieren. Das Zeichnen des Baumes und die korrekte Anordnung der Knoten und Kanten übernimmt dabei die *Prefuse*-Bibliothek. Diverse Layout-Algorithmen sind verfügbar, man kann aber auch seine eigenen implementieren und verwenden. Das Aussehen kann ebenfalls flexibel angepasst werden. Die Bibliothek wurde unter der BSD-Lizenz [Ope] veröffentlicht.

Registrieren für Ereignisse

Damit der Client nicht immer die komplette Graphenstruktur verarbeiten muss, kann er sich bei der Bibliothek für Statusänderungen an der Metadaten-Struktur registrieren. Folgende Ereignisse können eintreffen:

- Neue Metadaten werden zugefügt
- Alte Metadaten werden gelöscht

- Vorhandene Metadaten werden ergänzt oder ersetzt

Die Bibliothek bietet einen Callback-Mechanismus, bei dem der Client sich für die oben aufgeführten Ereignisse registrieren kann.

5.5.3. Client und GUI

Swing, Bestanteil jeder JVM, ist ein Framework zum Zeichnen von GUI-Elementen und Fenstern. Zwei Schichten sind für den Client vorgesehen: Eine Schicht kümmert sich um das Verwalten und füllen der Swing-Komponenten mit Daten. Dabei wird auf die Methoden der Bibliothek zurückgegriffen, um an die Metadaten zu gelangen. Die andere Schicht hat die Aufgabe die GUI-Elemente zu zeichnen und Benutzer-Interaktionen zu behandeln.

5.5.4. Bibliothek

Überblick

Wie in 5.5.2 erläutert, setzt sich die Bibliothek aus mehreren Komponenten zusammen. Abbildung 5.2 zeigt die verschiedenen Aufgabenbereiche der einzelnen Komponenten.

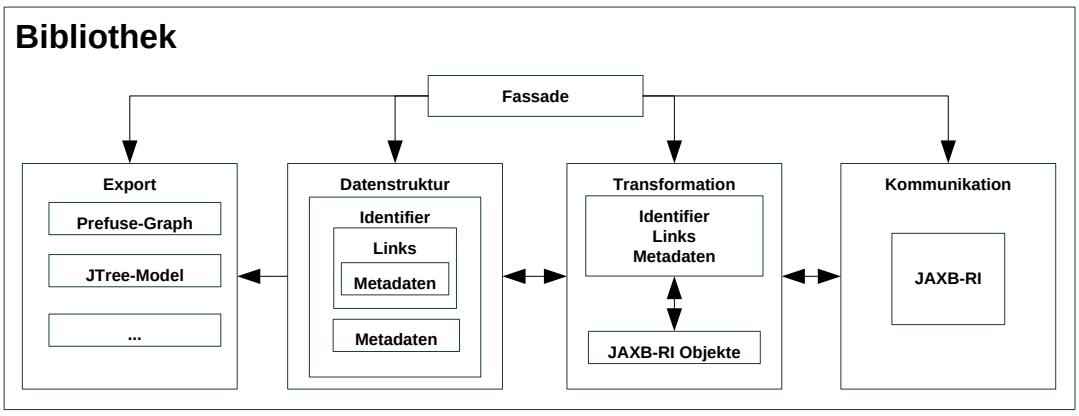


Abbildung 5.2.: Architektur der Bibliothek

Kommunikation

Diese Schicht ist für die Kommunikation mit dem MAP-Server verantwortlich. JAXB-RI Objekte werden an die Transformationsschicht weitergeleitet oder von dieser bezogen.

Transformation

Hier werden die JAXB-RI Elemente in die interne Datenstruktur umgewandelt und zur Datenstrukturschicht weitergegeben. Die Transformation kann auch in die andere Richtung geschehen: Die Identifier werden dann in JAXB-RI Objekte umgewandelt und der Kommunikationsschicht zur Verfügung gestellt.

Datenstruktur

Hier werden die Identifier gespeichert und der Export-Schicht durch geeignete Methoden angeboten. Die Datenstruktur setzt sich wie folgt zusammen:

- **Identifier** beschreibt einen Identifier. Dieser kann mehrere Links und Metadaten enthalten.
- **Link** kennzeichnet eine Beziehung zu einem anderem Identifier. Ein Link enthält Metadaten und kennt seine beiden Identifier.
- **Metadaten** sind Informationen, die eine Beziehung zwischen zwei Identifiern beschreibt. Jeder Link besitzt Metadaten. Ein Identifier kann ebenfalls Metadaten besitzen, auch wenn keine Beziehung zu einem anderen Identifier besteht.

Fassade

Die Fassade kapselt die einzelnen Komponenten von dem Client ab und deligiert die Anfragen an die einzelnen Komponenten. Der Client kann nur die Methoden der Fassade nutzen. Die anderen Komponenten sind für ihn verborgen.

6. Design und Implementierung der Softwarekomponenten

6.1. IRON-MAPS-Erweiterung

Die Kommunikationsschicht des IRON-MAPS verwendet Klassen, die mit JAXB-RI 2.4.5 aus einer WSDL 2.4.3 generiert wurden. Die WSDL und zwei XSD-Dateien wurden dem IRON-Projekt von der TNG zur Verfügung gestellt. Während die WSDL den eigentlichen Service definiert, beschreibt eine XSD die Operationen und Identifier von IF-MAP. Die andere enthält Definitionen für die Metadaten. Das Konzept dieser Arbeit sieht einen zusätzlichen Datentyp vor, die WSDL muss also angepasst werden und anschließend müssen daraus neue Klassen generiert werden. Da keine neuen Metadaten zugefügt werden, muss nur die erste XSD-Datei geändert werden.
Als erstes führen wir eine neue Anfrage-Operation ein (Listing 6.1).

```
<xsd:element name="dump" type="DumpRequestType"/>
```

Listing 6.1: Eine neue Anfrage-Operation wird erstellt (dump)

Der Typ ist *DumpRequestType* und wird ebenfalls definiert (Listing 6.2).

```
<xsd:complexType name="DumpRequestType">
   <xsd:attributeGroup ref="sessionAttributes"/>
 </xsd:complexType>
```

Listing 6.2: Ein neuer Anfragetyp wird erstellt

Es werden Attribute der Gruppe *sessionAttributes* akzeptiert (die *session-id* gehört dazu). Die Antwort einer Anfrage darf nur Identifier enthalten. Die Anzahl der Identifier, die das Resultat enthält, ist von null bis unbegrenzt (Listing 6.3).

```
<xsd:complexType name="DumpResponseType">
 <xsd:choice minOccurs="0" maxOccurs="unbounded">
       <xsd:element name="access-request" type="AccessRequestType"/>
       <xsd:element name="identity" type="IdentityType"/>
       <xsd:element name="ip-address" type="IPAddressType"/>
       <xsd:element name="mac-address" type="MACAddressType"/>
       <xsd:element name="device" type="DeviceType"/>
 </xsd:choice>
</xsd:complexType>
```

Listing 6.3: Die Antwort einer *dump*-Anfrage kann mehrere Identifier enthalten

Der *DumpResponseType* muss noch dem Typ *ResponseType* zugefügt werden (Listing 6.4).

```
<xsd:complexType name="ResponseType">
    <xsd:choice>
      ...
    <xsd:element name="dumpResult" type="DumpResponseType"/>
    ...
    </xsd:choice>
    <xsd:attributeGroup ref="validationAttributes"/>
  </xsd:complexType>
```

Listing 6.4: *DumpResponseType* wird dem IRON-MAP-Server bekannt gemacht

In der WSDL werden die neuen Datentypen und Operationen ergänzt (Listing 6.5).

```
<wsdl:message name="DumpRequest">
      <wsdl:part name="request" element="ifmap:dump"></wsdl:part>
</wsdl:message>

<wsdl:portType name="IfmapPortType">
       ...
 <wsdl:operation name="dump">
      <wsdl:input message="ifmapwsdl:DumpRequest"></wsdl:input>
        <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
 </wsdl:operation>
     ...
</wsdl:portType>

<wsdl:binding name="Service" type="ifmapwsdl:IfmapPortType">
        ...
        <wsdl:operation name="dump">
           <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        ...
</wsdl:binding>
```

Listing 6.5: Die WSDL muss um die neuen Operationen und Datentypen erweitert werden

Die Klassen können nun mit dem *xjc*[1] erstellt werden:

```
xjc.exe -wsdl .\ifmap-2.0.wsdl -d .\src\output\
```

Es gibt noch eine weitere WSDL-Datei, aus der Klassen generiert werden müssen. Die dort enthaltenen Datentypen und Funktionen sind nicht der Spezifikation der TNG entsprungen, sondern wurde speziell für den IRON-MAP von den Entwicklern der FHH erstellt (siehe Anhang A.5).

6.1.1. Neu generierte Klassen

Einige neue Klassen, die vorher nicht vorhanden waren, müssen dem MAP-Server zugefügt werden. Die meissten Klassen bleiben unberührt, ein paar Klassen werden durch neue ausgetauscht:

- DumpRequestType.java
- DumpResponseType.java
- ObjectFactory.java
- ResponseType.java
- Body.java

6.1.2. Alte Klassen anpassen

Damit der MAP-Server die neue Anfrage mit einer Liste von Identifiern beantworten kann, müssen Anpassungen an allen Schichten bis zum Datenmodell vorgenommen werden.

de.fhhannover.inform.iron.mapserver.communication
`Client.java`

[1] vergleiche Kapitel 2.4.5

```
private Envelope processRequestToResponse(Envelope env) {
     ...
     else if (bodyReq.getDump() != null) {
      resp = ServiceHandler.dump(bodyReq.getDump());
     }
     ...
```

de.fhhannover.inform.iron.mapserver.api.services

ServiceHandler.java

```
public static ResponseType dump(SessionRequestType session) {
    try {
      return DumpSessionReceive.dump(session);
    } catch (NoSuchAlgorithmException e) {
      return getSystemErrorResponse(e.getMessage());
    } catch (SessionNotFoundException e) {
      return getInvalidSessionIDResponse();
    } catch (ParameterException e) {
      e.printStackTrace();
      return getFailureResponse(e.getMessage());
    } catch (RunningSessionException e) {
      e.printStackTrace();
      return getFailureResponse(e.getMessage());
    } catch (PublisherConstructionException e) {
      e.printStackTrace();
      return getFailureResponse(e.getMessage());
}
```

de.fhhannover.inform.iron.mapserver.api.services.ssrc

DumpSessionReceive.java (neu)

```
package de.fhhannover.inform.iron.mapserver.communication.services.ssrc;

import java.security.NoSuchAlgorithmException;
import java.util.List;
import org.apache.log4j.Logger;
import org.trustedcomputinggroup._2010.ifmap._2.DumpResponseType;
import org.trustedcomputinggroup._2010.ifmap._2.ResponseType;
import org.trustedcomputinggroup._2010.ifmap._2.SessionRequestType;

import de.fhhannover.inform.iron.mapserver.communication.
DeprecatedDataModelServiceAccessor;
import
   de.fhhannover.inform.iron.mapserver.communication.connections.ConnectionContainer;
import de.fhhannover.inform.iron.mapserver.communication.services.ssrc.exceptions.
SessionNotFoundException;
import de.fhhannover.inform.iron.mapserver.datamodel.exceptions.ParameterException;
import de.fhhannover.inform.iron.mapserver.datamodel.exceptions.
PublisherConstructionException;
import
   de.fhhannover.inform.iron.mapserver.datamodel.exceptions.RunningSessionException;
import de.fhhannover.inform.iron.mapserver.datamodel.graph.AccessRequest;
import de.fhhannover.inform.iron.mapserver.datamodel.graph.Device;
import de.fhhannover.inform.iron.mapserver.datamodel.graph.Identifier;
import de.fhhannover.inform.iron.mapserver.datamodel.graph.Identity;
```

```
import de.fhhannover.inform.iron.mapserver.datamodel.graph.IpAddress;
import de.fhhannover.inform.iron.mapserver.datamodel.graph.MacAddress;
import de.fhhannover.inform.iron.mapserver.logging.Logging;
import de.fhhannover.inform.iron.mapserver.utils.IdentifierTransformer;

public class DumpSessionReceive {
  private static Logger logger;
  private static IdentifierTransformer iTrans = IdentifierTransformer.getInstance();
  static {
    logger = Logging.getTheLogger();
  }

  /**
   * Transform a given identifier into an object of a autogenerated class
   * and add it into the given ResultItemType object.
   *
   * @param rit
   * @param identifier
   */
  private static void add(DumpResponseType rit, Identifier identifier) {
    if (identifier != null && rit != null) {
      List<Object> retlist = rit.getAccessRequestOrIdentityOrIpAddress();
      if (identifier instanceof AccessRequest) {
        retlist.add(iTrans.transformAR((AccessRequest)identifier));
      } else if (identifier instanceof Device) {
        retlist.add(iTrans.transformDevice((Device)identifier));
      } else if (identifier instanceof Identity) {
        retlist.add(iTrans.transformIdentity((Identity)identifier));
      } else if (identifier instanceof IpAddress) {
        retlist.add(iTrans.transformIP((IpAddress)identifier));
      } else if (identifier instanceof MacAddress) {
        retlist.add(iTrans.transformMacAddress((MacAddress)identifier));
      }
    }
  }


  public static ResponseType dump(SessionRequestType session)
  throws NoSuchAlgorithmException, SessionNotFoundException, ParameterException,
     RunningSessionException, PublisherConstructionException{
    ConnectionContainer.getInstance().renewSession(session.getSessionId());
    ResponseType resp = new ResponseType();
      DumpResponseType dt =  new DumpResponseType();
      Identifier[] idents =
         DeprecatedDataModelServiceAccessor.getDataModelServiceInstance().dump();
      for(Identifier id : idents){
        DumpSessionReceive.add(dt, id);
      }
      resp.setDumpResult(dt);
    return resp;
  }
}
```

de.fhhannover.inform.iron.mapserver.datamodel

DataModelService.java

```
synchronized public Identifier[] dump() throws
  ParameterException, RunningSessionException, PublisherConstructionException {
    return this.identifierRep.getAllIdentifiers();
```

```
}
```

IdentifierRep.java

```
Identifier[] getAllIdentifiers(){
    Identifier[] idents = new Identifier[identifiers.size()];
    Collection<Identifier> list = identifiers.values();
    int i=0;
    for (Identifier id : list) {
      idents[i++] = id;
    }
    return idents;
  }
```

de.fhhannover.inform.iron.mapserver.util

LinkedMap.java

```
public int size(){
    return elements.size();
}
```

Mit dieser Erweiterung ist es dem MAP-Server möglich, eine *dump*-Anfrage zu bearbeiten und zu beantworten, wie in Kapitel 5.5.1 gefordert.

6.2. Implementierung der Client-Komponenten

Auf der Basis der in Kapitel 5.5.4 erörterten Implementierungsansätze folgt nun die konkrete Implementierung des Clients und der Bibliothek. Die wichtigsten Methoden aus den einzelnen Paketen werden erörtert und die Schnittstellen erklärt. Das Klassen diagramm ist aus Platzgründen im Anhang zu finden (Anhang **??**).

6.2.1. Bibliothek

Wie im Entwurf festgelegt, wird die Bibliothek in mehrere Komponenten aufgeteilt, die jeweils unterschiedliche Aufgabenbereiche haben (vgl. Kapitel 5.5.4).

Datenstruktur

de.fhhannover.inform.ifmap.datastructure

Die Datenstruktur setzt sich aus vier Klassen zusammen. `IdentifierContainer` enthält eine Menge von Objekten der Klasse `Identifier`, die wiederum über mehrere `LinkContainer` verfügen kann. Ein `LinkContainer` enthält mindestens ein `Link`-Objekt (wenn die Metadaten mit *ifmap-cardinality=singleValue* veröffentlicht wurden sind) oder mehrere `Link`-Objekte bei *ifmap-cardinality=multiValue*.

Events

de.fhhannover.inform.ifmap.event

In diesem Paket finden sich Interfaces, die ein Client implementieren kann, wenn er über Statusänderungen informiert werden will. Dazu muss er sich bei der Bibliothek registrieren. Die Bibliothek ruft bei Änderungen an der Datenstruktur die Methoden auf.

`IdentifierChangedReceiver.java`

```
public void processNew(IdentifierContainer conti);
public void processUpdate(IdentifierContainer conti);
public void processDelete(IdentifierContainer conti);
```

processNew wird bei neuen Metadaten aufgerufen. *processUpdate* ergänzt vorhandene Metadaten und *processDelete* gibt Metadaten zum Löschen frei.

`StatusChangedReceiver.java`

```
public void processStatusEvent(String event);
```

Gibt den Client die Möglichkeit, die Aktionen die die Bibliothek gerade bearbeitet, nachzuvollziehen.

`NodeSelectedReceiver.java`

```
public void nodeSelected(VisualItem item);
```

nodeSelected wird aufgerufen, wenn ein Knoten im *Prefuse*-Graph selektiert wird.

Fassade

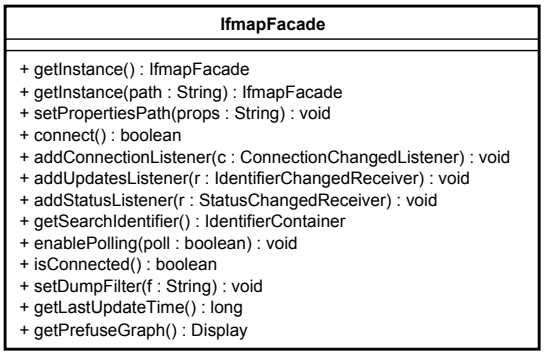


Abbildung 6.1.: IfmapFacade

Die Klasse `IfmapFacade` ist der Einstiegspunkt und kapselt den Zugriff auf die darunterliegenden Schichten. Alle Funktionalitäten der Bibliothek lassen sich ausschliesslich über Instanzen dieser Klasse nutzen.

de.fhhannover.inform.ifmap.control

`IfmapFacade.java`

Objekte der Klasse `IfmapFacade` werden nicht mit dem *new* Operator erstellt, sondern über die *getInstance*-Methoden. Der Pfad zur Konfigurationsdatei kann gleich mit übergeben werden.
setPropertiesPath setzt den Pfad zur Konfigurationsdatei.
Mit *getSearchIdentifier* wird eine Liste mit den aktuell verlinkten Identifiern abgerufen. Wird die Konfigurationsdatei nicht gefunden oder wurde der Pfad vor dem Aufruf dieser Methode nicht gesetzt, so wird eine `PropertiesNotFoundException` ausgelöst. Konnte keine Verbindung zum MAP-Server aufgebaut werden, so muss der

Client die `RemoteException` behandeln.
Mit *connect* und *disconnect* wird die Verbindung zum MAP-Server auf- bzw. abgebaut. *isConnected* prüft den aktuellen Verbindungsstatus. *subscribe* registriert Identifier für Statusänderungen. *enablePolling* startet und stoppt das Polling auf die abonierten Identifier. *getLastUpdateTime* liefert einen Zeitstempel der letzten Änderung der Datenstruktur des MAP-Servers.
addStatusListener und *addUpdatesListener* registriert einen Client bei der Bibliothek für Updates.
Ein Graph auf *Prefuse*-Basis wird durch die *getPrefuseGraph*-Methode erzeugt.
Ist eine Verbindung zu einem MAP-Server aufgebaut, so kann mit *getPublisherID* die aktuelle Publisher-ID und mit *getSessionID* die Session-ID abgerufen werden.

Kommunikationsschicht

Einige Klassen werden mit Hilfe von *WSDL2JAVA*, basierend auf der aktuellen Spezifikation von IF-MAP, generiert. Diese Klassen kümmern sich um die Kommunikation mit dem MAP-Server und nehmen das *marshalling* und *unmarshalling* der XML-Nachrichten und JAVA-Objekte vor. Sie sind in den Paketen *org.trustedcomputinggroup.** untergebracht.

org.trustedcomputinggroup._2010.ifmap._2
Enthält die Typ-Deklarationen der IF-MAP-Operationen.

org.trustedcomputinggroup._2010.ifmap_metadata._2
Sämtliche von der TNG spezifizierte Metadata-Typen finden sich hier.

org.trustedcomputinggroup._2010.ifmap_metadata._2
Enthält die Stub-Klasse `IfmapServiceStub`, über die sämtliche Kommunikation abgewickelt wird.

de.fhhannover.inform.ifmap.communication
`IFMAPConnection.java`
Diese Klasse ist von `IfmapServiceStub` abgeleitet. Sie kümmert sich um die Initialisierung der Verbindung zum MAP-Server und das Einlesen der Konfigurationen. Sie greift auf die Methoden der `PropertiesReader`-Klasse zu.
Ein `Connection`-Objekt beinhaltet eine Instanz von `IFMAPConnection.java`.
Die `SOAPConnector`-Klasse benutzt ein `Connection`-Objekt, um die IF-MAP-Operationen abzusetzen.

Transformationsschicht

de.fhhannover.inform.ruhe.ifmap.transform
`MessageTransformer.java`

Die Klasse `MessageTransformer` kümmert sich um die Transformation der JAXB-RI- Objekte in Objekte der internen Datenstrukur. Sie benutzt eine private Thread-Klasse `Poller`, um asynchron Statusänderungen vom MAP-Server zu beziehen.

Exportschicht

de.fhhannover.inform.ruhe.ifmap.output
In diesem Paket finden sich die Visualisierungs-Implementierungen.
`JPrefuse.java`
Diese Implementierung ist ein Graph basierend auf der *Prefuse*-Bibliothek. Sie hat das Interface `IdentifierChangedReceiver` implementiert und wird somit automatisch mit Updates versorgt.

7. Bewertung

7.1. Erreichte Ziele

In dieser Bachelorarbeit wurden folgende Zielsetzungen erreicht:

- Analyse des IF-MAP-Protokolls und der aktuellen Implementierung des IRON-MAP-Servers
- Entwicklung eines Konzepts, um sämtliche IF-MAP-Metadaten aus einem MAP-Server zu gewinnen und darzustellen.
- Erweiterung des IRON-MAP-Servers um eine neue Funktionalität.
- Entwicklung und Implementierung einer Bibliothek zum Auslesen und zur Darstellung der Metadaten.
- Referenzimplementierung eines Clients, der diese Bibliothek verwendet.

7.2. Ausblick

Während diese Arbeit geschrieben wurde, ist die zugrundeliegende IF-MAP-Spezifikation noch nicht in der Version, wie sie dem Autor vorliegt, veröffentlicht. Im Rahmen des IRON-Projektes wurde von der TNG eine Vorab-Version zur Verfügung gestellt, die beinahe final ist und mit Abschluss dieser Arbeit kurz vor der Veröffentlichung steht. Für diese Version stellt das hier vorgestellte Konzept eine nützliche Erweiterung für die Visualisierung der Metadaten eines MAP-Servers dar. Zukünftige Änderungen an der Spezifikation

könnten das Konzept obsolet machen, weil dann möglicherweise auch ohne Erweiterung des MAP-Servers alle gewünschten Informationen abrufbar sind.
Die verwendete *Prefuse*-Bibliothek ist sehr komplex und die Möglichkeiten, die diese Bibliothek bietet, sind nicht ausgeschöpft. Eine tiefergehende Beschäftigung mit der Bibliothek und speziell mit den Layout-Algorithmen würde die Darstellung für den Anwender noch anschaulicher machen.

Literaturverzeichnis

[Apa09] Apache Software Foundation (Hrsg.): *Apache Axis2/Java - Next Generation Web Services*. Apache Software Foundation, 2004-2009. – http://ws.apache.org/axis2/

[Ber] Berkeley Institute of Design (Hrsg.): *Prefuse - Information visualization toolkit*. Berkeley Institute of Design. – http://www.opensource.org/licenses/bsd-license.php

[Gro] Group, Network W.: *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication*. Website, . – http://www.ietf.org/rfc/rfc2617.txt

[Ope] Open Source Initiative (Hrsg.): *Open Source Initiative OSI - The BSD License*. Open Source Initiative. – http://www.opensource.org/licenses/bsd-license.php

[Oraa] Oracle (Hrsg.): *JAXB Reference Implementation Project*. Oracle. – https://jaxb.dev.java.net/

[Orab] Oracle (Hrsg.): *Remote Method Invocation Home*. Oracle. – http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[SUN] SUN (Hrsg.): *Message-Oriented Middleware (MOM)*. SUN. – http://docs.sun.com/app/docs/doc/819-7759/aeraq

[Tru09] Trusted Network Group (Hrsg.): *Overview of Trusted Network Connect (TNC) IF-MAP*. Trusted Network Group, April 2009

[Tru10a] Trusted Network Group: *TNC IF-MAP Binding for SOAP*, March 2010. – Specification Version 2.0 Revision 33 (unpublished)

[Tru10b] Trusted Network Group: *TNC IF-MAP Metadata for Network*, March 2010. – Specification Version 1.0 Revision 20 (unpublished)

[Val09] Valerij Procenko, Arne Welzel: *MAP-Server Datenmodell*, October 2009

[W3Ca] W3C (Hrsg.): *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language.* W3C. – http://www.w3.org/TR/wsdl20/

[W3Cb] W3C (Hrsg.): *World Wide Web Consortium.* W3C. – http://www.w3.org/

[W3C04] W3C (Hrsg.): *XML Schema Part 0: Primer Second Edition.* W3C, October 2004. – http://www.w3.org/TR/xmlschema-0/

[W3C07] W3C (Hrsg.): *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).* W3C, April 2007

[Wika] Wikipedia (Hrsg.): *Public-key cryptography.* Wikipedia. – http://en.wikipedia.org/wiki/Public-key_cryptography

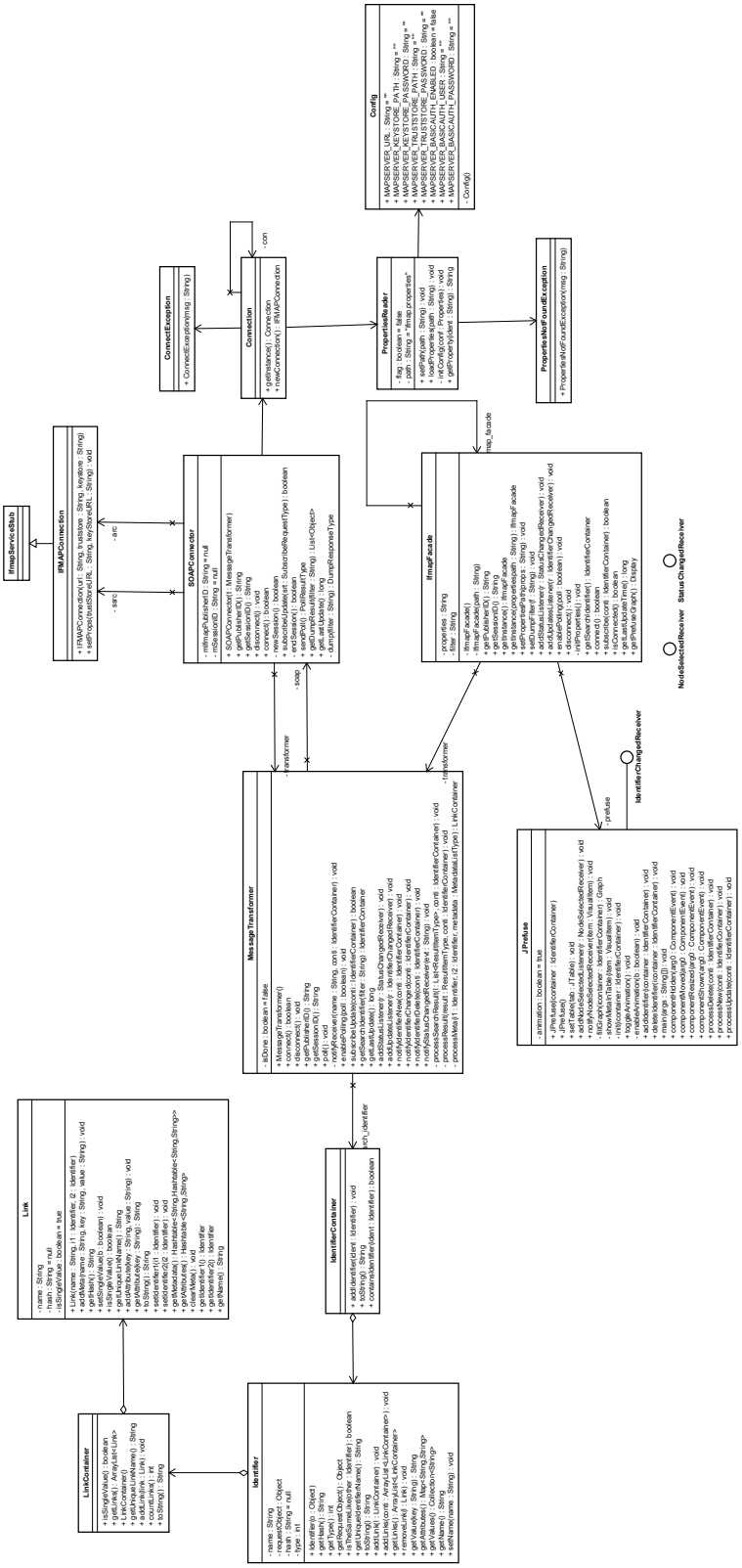
[Wikb] Wikipedia (Hrsg.): *Schemasprache (XML).* Wikipedia. – http://de.wikipedia.org/wiki/Schemasprache_(XML)

Abbildungsverzeichnis

Listings

A. Anhang

A.1. Klassendiagramm der entwickelten Bibliothek



A.2. ifmap-base-2.0v17.xsd

```
<?xml version="1.0"?>
<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">

  <!-- 18.02.10 aw - Some modifications to get the wsdl2java
       tool generating better code. Additionally removed
       minOccurs and maxOccurs in the UpdateType -->

  <!-- top-level elements represent all the possible
       requests and responses -->
  <xsd:element name="publish" type="PublishRequestType"/>
  <xsd:element name="search" type="SearchRequestType"/>
  <xsd:element name="subscribe" type="SubscribeRequestType"/>
  <xsd:element name="poll" type="PollRequestType"/>
  <xsd:element name="purgePublisher" type="PurgePublisherRequestType"/>
  <xsd:element name="newSession" type="NewSessionRequestType"/>
  <xsd:element name="attachSession" type="SessionRequestType"/>
  <xsd:element name="renewSession" type="SessionRequestType"/>
  <xsd:element name="endSession" type="SessionRequestType"/>
  <xsd:element name="response" type="ResponseType"/>
  <xsd:element name="dump" type="DumpRequestType"/>

  <!-- AccessRequestType Identifier represents an endpoint
       which is attempting to gain entry to the network-->
  <xsd:complexType name="AccessRequestType">
    <xsd:attribute name="administrative-domain" type="xsd:string"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
  </xsd:complexType>

  <!-- DeviceType Identifier represents a physical asset
       which is attempting to gain entry to the network -->
  <xsd:complexType name="DeviceType">
    <xsd:choice>
      <xsd:element name="aik-name" type="xsd:string"/>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:choice>
  </xsd:complexType>

  <!-- IdentityType Identifier represents an end-user -->
  <xsd:complexType name="IdentityType">
    <xsd:attribute name="administrative-domain" type="xsd:string"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="type" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="aik-name"/>
          <xsd:enumeration value="distinguished-name"/>
          <xsd:enumeration value="dns-name"/>
          <xsd:enumeration value="email-address"/>
          <xsd:enumeration value="hip-hit"/>
          <xsd:enumeration value="kerberos-principal"/>
          <xsd:enumeration value="trusted-platform-module"/>
          <xsd:enumeration value="username"/>
          <xsd:enumeration value="sip-uri"/>
          <xsd:enumeration value="tel-uri"/>
          <xsd:enumeration value="other"/>
        </xsd:restriction>
      </xsd:simpleType>
```

```
    </xsd:attribute>
    <xsd:attribute name="other-type-definition" type="xsd:string"/>
  </xsd:complexType>

  <!-- IPAddressType Identifier represents a single IP address -->
  <xsd:complexType name="IPAddressType">
    <xsd:attribute name="administrative-domain" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:string" use="required"/>
    <xsd:attribute name="type">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="IPv4"/>
          <xsd:enumeration value="IPv6"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>

  <!-- MACAddressType Identifier represents an Ethernet MAC address -->
  <xsd:complexType name="MACAddressType">
    <xsd:attribute name="administrative-domain" type="xsd:string"/>
    <xsd:attribute name="value" type="xsd:string" use="required"/>
  </xsd:complexType>

  <!-- MetadataListType is a container for metadata within
       other elements -->
  <xsd:complexType name="MetadataListType">
    <xsd:sequence>
      <xsd:any minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- FilterType is a subset of XPath -->
  <xsd:simpleType name="FilterType">
    <xsd:restriction base="xsd:string"/>
  </xsd:simpleType>

  <xsd:complexType name="NewSessionRequestType">
    <xsd:attribute name="max-poll-result-size" type="xsd:integer" use="optional"/>
  </xsd:complexType>

  <xsd:attributeGroup name="validationAttributes">
    <xsd:attribute name="validation" use="optional">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="None"/>
          <xsd:enumeration value="BaseOnly"/>
          <xsd:enumeration value="MetadataOnly"/>
          <xsd:enumeration value="All"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:attributeGroup>

  <xsd:attributeGroup name="sessionAttributes">
    <xsd:attribute name="session-id" type="xsd:string" use="required"/>
  </xsd:attributeGroup>

  <!-- SessionRequestType is stateful session handling -->
  <xsd:complexType name="SessionRequestType">
    <xsd:attributeGroup ref="sessionAttributes"/>
  </xsd:complexType>
```

```
<xsd:complexType name="DumpRequestType">
  <xsd:attributeGroup ref="sessionAttributes"/>
<xsd:attribute name="identifier" type="xsd:string"/>
</xsd:complexType>

<!-- UpdateType is the type for requests that update metadata -->
<xsd:complexType name="UpdateType">
  <xsd:sequence>
    <xsd:choice minOccurs="1" maxOccurs="2">
      <xsd:element name="access-request" type="AccessRequestType"/>
      <xsd:element name="identity" type="IdentityType"/>
      <xsd:element name="ip-address" type="IPAddressType"/>
      <xsd:element name="mac-address" type="MACAddressType"/>
      <xsd:element name="device" type="DeviceType"/>
    </xsd:choice>
    <xsd:element name="metadata" type="MetadataListType" minOccurs="1"
       maxOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="lifetime" default="session">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="session"/>
        <xsd:enumeration value="forever"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>


<!-- Just a copy of UpdateType to produce better code when using
     JAXBRI code generation. getUpdate or getDelete was created,
     but no getNotify
-->
<xsd:complexType name="NotifyType">
  <xsd:sequence>
    <xsd:choice minOccurs="1" maxOccurs="2">
      <xsd:element name="access-request" type="AccessRequestType"/>
      <xsd:element name="identity" type="IdentityType"/>
      <xsd:element name="ip-address" type="IPAddressType"/>
      <xsd:element name="mac-address" type="MACAddressType"/>
      <xsd:element name="device" type="DeviceType"/>
    </xsd:choice>
    <xsd:element name="metadata" type="MetadataListType"/>
  </xsd:sequence>
  <xsd:attribute name="lifetime" default="session">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="session"/>
        <xsd:enumeration value="forever"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:complexType>

<!-- DeleteType is the type for the delete element of
     a publish request, and specifies which metadata
     to delete. -->
<xsd:complexType name="DeleteType">
  <xsd:sequence>
    <xsd:choice minOccurs="1" maxOccurs="2">
      <xsd:element name="access-request" type="AccessRequestType"/>
```

```
      <xsd:element name="identity" type="IdentityType"/>
      <xsd:element name="ip-address" type="IPAddressType"/>
      <xsd:element name="mac-address" type="MACAddressType"/>
      <xsd:element name="device" type="DeviceType"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="filter" type="FilterType" use="optional"/>
</xsd:complexType>

<!-- PublishRequestType updates or deletes metadata -->
<xsd:complexType name="PublishRequestType">
  <xsd:sequence>
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
      <xsd:element name="update" type="UpdateType"/>
      <xsd:element name="notify" type="NotifyType"/>
      <xsd:element name="delete" type="DeleteType"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attributeGroup ref="sessionAttributes"/>
  <xsd:attributeGroup ref="validationAttributes"/>
</xsd:complexType>

<!-- SearchType specifies the parameters for a search, and is used
     for the search element as well as the update sub-element of a
     subscribe element -->
<xsd:complexType name="SearchType">
  <xsd:sequence>
    <xsd:choice minOccurs="1" maxOccurs="1">
      <xsd:element name="access-request" type="AccessRequestType"/>
      <xsd:element name="identity" type="IdentityType"/>
      <xsd:element name="ip-address" type="IPAddressType"/>
      <xsd:element name="mac-address" type="MACAddressType"/>
      <xsd:element name="device" type="DeviceType"/>
    </xsd:choice>
  </xsd:sequence>
  <xsd:attribute name="match-links" type="FilterType"/>
  <xsd:attribute name="max-depth" type="xsd:unsignedInt"/>
  <xsd:attribute name="terminal-identifier-type" type="xsd:string"/>
  <xsd:attribute name="max-size" type="xsd:unsignedInt"/>
  <xsd:attribute name="result-filter" type="FilterType"/>
</xsd:complexType>

<!-- SearchRequestType queries the server for matching
     metadata -->
<xsd:complexType name="SearchRequestType">
  <xsd:complexContent>
    <xsd:extension base="SearchType">
      <xsd:attributeGroup ref="sessionAttributes"/>
      <xsd:attributeGroup ref="validationAttributes"/>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<!-- DeleteSearchRequestType is for removing subscriptions -->
<xsd:complexType name="DeleteSearchRequestType">
  <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>

<!-- SubscribeRequestType is for managing subscriptions -->
<xsd:complexType name="SubscribeRequestType">
  <xsd:sequence>
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
```

```
        <xsd:element name="update">
          <xsd:complexType>
            <xsd:complexContent>
              <xsd:extension base="SearchType">
                <xsd:attribute name="name" type="xsd:string" use="required"/>
              </xsd:extension>
            </xsd:complexContent>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="delete" type="DeleteSearchRequestType"/>
      </xsd:choice>
    </xsd:sequence>
    <xsd:attributeGroup ref="sessionAttributes"/>
    <xsd:attributeGroup ref="validationAttributes"/>
  </xsd:complexType>

  <!-- PollRequestType is for polling for notification of
       metadata changes that match subscriptions -->
  <xsd:complexType name="PollRequestType">
    <xsd:attributeGroup ref="validationAttributes"/>
    <xsd:attributeGroup ref="sessionAttributes"/>
  </xsd:complexType>

  <!-- PurgePublisherRequestType is for removing all metadata
       published by a particular publisher -->
  <xsd:complexType name="PurgePublisherRequestType">
    <xsd:attribute name="ifmap-publisher-id" type="xsd:string"/>
    <xsd:attributeGroup ref="sessionAttributes"/>
  </xsd:complexType>

  <!-- DumpType for visualisation -->
<xsd:complexType name="DumpResponseType">
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="access-request" type="AccessRequestType"/>
        <xsd:element name="identity" type="IdentityType"/>
        <xsd:element name="ip-address" type="IPAddressType"/>
        <xsd:element name="mac-address" type="MACAddressType"/>
        <xsd:element name="device" type="DeviceType"/>
  </xsd:choice>
  <xsd:attribute name="last-update" type="xsd:string"/>
 </xsd:complexType>

  <!-- ResultItemType is for search or poll results showing
       metadata attached to identifiers and links-->
  <xsd:complexType name="ResultItemType">
    <xsd:sequence>
      <xsd:choice minOccurs="1" maxOccurs="2">
        <xsd:element name="access-request" type="AccessRequestType"/>
        <xsd:element name="identity" type="IdentityType"/>
        <xsd:element name="ip-address" type="IPAddressType"/>
        <xsd:element name="mac-address" type="MACAddressType"/>
        <xsd:element name="device" type="DeviceType"/>
      </xsd:choice>
      <xsd:element name="metadata" type="MetadataListType" minOccurs="0"
         maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

  <!-- SearchResultType contains the identifiers and links
       along with associated metadata -->
  <xsd:complexType name="SearchResultType">
    <xsd:sequence>
```

```
      <xsd:element name="resultItem" type="ResultItemType" minOccurs="0"
         maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string"/>
  </xsd:complexType>

  <!-- PollResultType contains a searchResult for each
       subscription that had changes since the last poll -->
  <xsd:complexType name="PollResultType">
    <xsd:sequence>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="searchResult" type="SearchResultType"/>
        <xsd:element name="updateResult" type="SearchResultType"/>
        <xsd:element name="deleteResult" type="SearchResultType"/>
        <xsd:element name="notifyResult" type="SearchResultType"/>
        <xsd:element name="errorResult" type="ErrorResultType"/>
      </xsd:choice>
    </xsd:sequence>
  </xsd:complexType>

  <!-- ErrorResultType indicates the cause of an error -->
  <xsd:complexType name="ErrorResultType">
    <xsd:sequence>
      <xsd:element name="errorString" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute name="errorCode" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="AccessDenied"/>
          <xsd:enumeration value="Failure"/>
          <xsd:enumeration value="InvalidIdentifier"/>
          <xsd:enumeration value="InvalidIdentifierType"/>
          <xsd:enumeration value="IdentifierTooLong"/>
          <xsd:enumeration value="InvalidMetadata"/>
          <xsd:enumeration value="InvalidSchemaVersion"/>
          <xsd:enumeration value="InvalidSessionID"/>
          <xsd:enumeration value="MetadataTooLong"/>
          <xsd:enumeration value="SearchResultsTooBig"/>
          <xsd:enumeration value="PollResultsTooBig"/>
          <xsd:enumeration value="SystemError"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="name" type="xsd:string"/>
  </xsd:complexType>

  <!-- SessionResultType is for stateful session handling -->
  <xsd:complexType name="SessionResultType">
    <xsd:attribute name="session-id" type="xsd:string" use="required"/>
    <xsd:attribute name="ifmap-publisher-id" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="NewSessionResultType">
    <xsd:attribute name="session-id" type="xsd:string" use="required"/>
    <xsd:attribute name="ifmap-publisher-id" type="xsd:string" use="required"/>
    <xsd:attribute name="max-poll-result-type" type="xsd:integer" use="optional"/>
  </xsd:complexType>

  <!-- ResponseType encapsulates results from all the different
       requests -->
<xsd:complexType name="ResponseType">
    <xsd:choice>
```

```
    <xsd:element name="errorResult" type="ErrorResultType"/>
    <xsd:element name="pollResult" type="PollResultType"/>
    <xsd:element name="searchResult" type="SearchResultType"/>
    <xsd:element name="subscribeReceived">
      <xsd:complexType/>
    </xsd:element>
    <xsd:element name="publishReceived">
      <xsd:complexType/>
    </xsd:element>
    <xsd:element name="purgePublisherReceived">
      <xsd:complexType/>
    </xsd:element>
    <xsd:element name="newSessionResult" type="NewSessionResultType"/>
    <xsd:element name="attachSessionResult" type="SessionResultType"/>
    <xsd:element name="renewSessionResult">
      <xsd:complexType/>
    </xsd:element>
    <xsd:element name="endSessionResult">
      <xsd:complexType/>
    </xsd:element>
  <xsd:element name="dumpResult" type="DumpResponseType"/>
  </xsd:choice>
  <xsd:attributeGroup ref="validationAttributes"/>
</xsd:complexType>

<!-- metadataAttributes specifies attributes on metadata which are
     used by MAP servers.

     ifmap-publisher-id and ifmap-timestamp are added to all metadata by the
     server before storage in the database. MAP clients MUST NOT
     include ifmap-publisher-id or ifmap-timestamp in published metadata.

     cardinality is used by the MAP client to indicate to the server
     whether the metadata can have multiple values.

     anyAttribute enables metadata elements which include the
     ifmap:metadataAttributes attributeGroup to add new attributes
     for use with future versions of IF-MAP. -->
<xsd:attributeGroup name="metadataAttributes">
  <xsd:attribute name="ifmap-publisher-id" type="xsd:string"/>
  <xsd:attribute name="ifmap-timestamp" type="xsd:dateTime"/>
  <xsd:anyAttribute/>
</xsd:attributeGroup>

<xsd:attributeGroup name="singleValueMetadataAttributes">
  <xsd:attributeGroup ref="metadataAttributes"/>
  <xsd:attribute name="ifmap-cardinality" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="singleValue"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:attributeGroup>

<xsd:attributeGroup name="multiValueMetadataAttributes">
  <xsd:attributeGroup ref="metadataAttributes"/>
  <xsd:attribute name="ifmap-cardinality" use="required">
    <xsd:simpleType>
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value="multiValue"/>
      </xsd:restriction>
```

```
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:attributeGroup>

</xsd:schema>
```

A.3. ifmap-metadata-2.0v8.xsd

```
<?xml version="1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
  xmlns="http://www.trustedcomputinggroup.org/2010/IFMAP-METADATA/2"
  targetNamespace="http://www.trustedcomputinggroup.org/2010/IFMAP-METADATA/2">

  <!-- 18.02.10 (aw) The import element was added later as
       wsdl2java complained about it being missing -->
  <xsd:import namespace="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
     schemaLocation="ifmap-base-2.0v17.xsd"></xsd:import>

  <!-- Schema for IF-MAP Metadata for Network Security -->

  <!-- access-request-device is link metadata that
       associates an access-request identifier with
       a device identifier -->
  <xsd:element name="access-request-device">
    <xsd:complexType>
      <xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

  <!-- access-request-ip is link metadata that
       associates an access-request identifier with
       an ip-address identifier -->
  <xsd:element name="access-request-ip">
    <xsd:complexType>
      <xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

  <!-- access-request-mac is link metadata that
       associates an access-request identifier with
       a mac-address identifier -->
  <xsd:element name="access-request-mac">
    <xsd:complexType>
      <xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

  <!-- authenticated-as is link metadata that
       associates an access-request identifier with
       an identity identifier -->
  <xsd:element name="authenticated-as">
    <xsd:complexType>
      <xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

  <!-- authenticated-by is link metadata that
```

```
      associates an access-request identifier with
      the device identifier of the PDP that
      authenticated the access-request -->
  <xsd:element name="authenticated-by">
    <xsd:complexType>
      <xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

  <!-- capability is access-request metadata that names
       a collection of privileges assigned to an endpoint -->
  <xsd:element name="capability">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
        <xsd:element name="administrative-domain" type="xsd:string" minOccurs="0"
            maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

  <!-- device-attribute is link metadata that associates
       an access-request identifier with a device identifier
       and which includes information about the device such
       as its health -->
  <xsd:element name="device-attribute">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

  <!-- device-characteristic is link metadata that associates
       an access-request, ip-address, or mac-address identifier
       of an endpoint with the device identifier of the MAP
       Client publishing the metadata, which includes information
       about what kind of device is represented by the
       mac-address, ip-address, or access-request -->
  <xsd:element name="device-characteristic">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="manufacturer" type="xsd:string" minOccurs="0"
            maxOccurs="1"/>
        <xsd:element name="model" type="xsd:string" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="os" type="xsd:string" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="os-version" type="xsd:string" minOccurs="0"
            maxOccurs="1"/>
        <xsd:element name="device-type" type="xsd:string" minOccurs="0"/>
        <xsd:element name="discovered-time" type="xsd:dateTime" minOccurs="1"
            maxOccurs="1"/>
        <xsd:element name="discoverer-id" type="xsd:string" minOccurs="1"
            maxOccurs="1"/>
        <xsd:element name="discovery-method" type="xsd:string" minOccurs="1"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

<!-- device-ip is link metadata that associates a device
```

```
     identifier of a PDP with an IP address which it has
     authenticated -->
  <xsd:element name="device-ip">
    <xsd:complexType>
      <xsd:attributeGroup
        ref="ifmap:singleValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

  <!-- discovered-by is link metadata that associates
       an ip-address or mac-address identifier of an endpoint
       with the device identifier of a MAP Client that has
       noticed the endpoint on the network -->
  <xsd:element name="discovered-by">
    <xsd:complexType>
      <xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

<!-- enforcement-report is link metadata between a device
     identifier of a PEP or flow controller and an ip-address
     or mac-address identifier of an endpoint, indicating an
     enforcement action in progress -->
  <xsd:element name="enforcement-report">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="enforcement-action" minOccurs="1" maxOccurs="1">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="block"/>
              <xsd:enumeration value="quarantine"/>
              <xsd:enumeration value="other"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="other-type-definition" type="xsd:string" minOccurs="0"
           maxOccurs="1"/>
        <xsd:element name="enforcement-reason" type="xsd:string" minOccurs="0"
           maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attributeGroup
        ref="ifmap:multiValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

  <!-- event is access-request, identity, ip-address, or
       mac-address metadata that describes activity of
       interest detected on the network -->
  <xsd:element name="event">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="discovered-time" type="xsd:dateTime" minOccurs="1"
         maxOccurs="1"/>
      <xsd:element name="discoverer-id" type="xsd:string" minOccurs="1"
         maxOccurs="1"/>
        <xsd:element name="magnitude" minOccurs="1" maxOccurs="1">
          <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
              <xsd:minInclusive value="0"/>
              <xsd:maxInclusive value="100"/>
            </xsd:restriction>
```

```
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="confidence" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="0"/>
            <xsd:maxInclusive value="100"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="significance" minOccurs="1" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="critical"/>
            <xsd:enumeration value="important"/>
            <xsd:enumeration value="informational"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="type" minOccurs="0" maxOccurs="1">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="p2p"/>
            <xsd:enumeration value="cve"/>
            <xsd:enumeration value="botnet infection"/>
            <xsd:enumeration value="worm infection"/>
            <xsd:enumeration value="excessive flows"/>
            <xsd:enumeration value="behavioral change"/>
            <xsd:enumeration value="policy violation"/>
            <xsd:enumeration value="other"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element name="other-type-definition" type="xsd:string" minOccurs="0"
          maxOccurs="1"/>
      <xsd:element name="information" type="xsd:string" minOccurs="0"
          maxOccurs="1"/>
      <xsd:element name="vulnerability-uri" type="xsd:anyURI" minOccurs="0"
          maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
  </xsd:complexType>
</xsd:element>

  <!-- ip-mac is link metadata that associates an
     ip-address identifier with a mac-address identifier
     and which includes optional DHCP lease information -->
<xsd:element name="ip-mac">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="start-time" type="xsd:dateTime" minOccurs="0"
          maxOccurs="1"/>
      <xsd:element name="end-time" type="xsd:dateTime" minOccurs="0"
          maxOccurs="1"/>
      <xsd:element name="dhcp-server" type="xsd:string" minOccurs="0"
          maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
  </xsd:complexType>
</xsd:element>

<!-- layer2-information is link metadata that
```

```
       associates an access-request identifier with
       the device identifier of the PEP through
       which the endpoint is accessing the network -->
  <xsd:element name="layer2-information">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="vlan" type="xsd:integer" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="vlan-name" type="xsd:integer" minOccurs="0"
           maxOccurs="1"/>
        <xsd:element name="port" type="xsd:integer" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="administrative-domain" type="xsd:string" minOccurs="0"
           maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

<!-- location indicates information about the location of an
    identity, ip-address, or mac-address -->
  <xsd:element name="location">
    <xsd:complexType>
      <xsd:sequence>
      <xsd:element name="location-information" minOccurs="1" maxOccurs="unbounded">
            <xsd:complexType>
              <xsd:attribute name="type" type="xsd:string"/>
              <xsd:attribute name="value" type="xsd:string"/>
            </xsd:complexType>
          </xsd:element>
        <xsd:element name="discovered-time" type="xsd:dateTime" minOccurs="1"
           maxOccurs="1"/>
        <xsd:element name="discoverer-id" type="xsd:string" minOccurs="1"
           maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

<!-- request-for-investigation is link metadata that associates
     an ip-address or mac-address identifier with a device
     identifier representing a PDP that would like a Sensor to
     investigate the ip-address or mac-address -->
  <xsd:element name="request-for-investigation">
    <xsd:complexType>
      <xsd:attribute name="qualifier" type="xsd:string" use="optional"/>
      <xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

  <!-- role is link metadata that associates an
       access-request identifier with an identity
       identifier and which names collections of
       privileges associated with the end-user -->
  <xsd:element name="role">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="administrative-domain" type="xsd:string" minOccurs="0"
           maxOccurs="1"/>
        <xsd:element name="name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>
```

```
  <xsd:simpleType name="wlan-security-enum">
      <xsd:restriction base="xsd:string">
          <xsd:enumeration value="open"/>
          <xsd:enumeration value="wep"/>
          <xsd:enumeration value="tkip"/>
          <xsd:enumeration value="ccmp"/>
          <xsd:enumeration value="bip"/>
          <xsd:enumeration value="other"/>
      </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="wlan-security-type">
      <xsd:simpleContent>
          <xsd:extension base="wlan-security-enum">
              <xsd:attribute name="other-type-definition" type="xsd:string"
                 use="optional"/>
          </xsd:extension>
      </xsd:simpleContent>
  </xsd:complexType>
  <!-- wlan-information is link metadata between an access-
     request and device, and indicates properties of the
     wireless LAN connection of the access-request -->
  <xsd:element name="wlan-information">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ssid" type="xsd:string" minOccurs="0" maxOccurs="1"/>
        <xsd:element name="ssid-unicast-security" minOccurs="1"
           maxOccurs="unbounded" type="wlan-security-type">
        </xsd:element>
        <xsd:element name="ssid-group-security" minOccurs="1" maxOccurs="1"
           type="wlan-security-type"/>
        <xsd:element name="ssid-management-security" minOccurs="1"
           maxOccurs="unbounded" type="wlan-security-type"/>
      </xsd:sequence>
      <xsd:attributeGroup
      ref="ifmap:singleValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>

<!-- unexpected-behavior is access-request, identity, ip-
     address, or mac-address metadata that describes activity
of interest detected on the network -->
  <xsd:element name="unexpected-behavior">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="discovered-time" type="xsd:dateTime" minOccurs="1"
           maxOccurs="1"/>
        <xsd:element name="discoverer-id" type="xsd:string" minOccurs="1"
           maxOccurs="1"/>
        <xsd:element name="information" type="xsd:string" minOccurs="0"
           maxOccurs="1"/>
        <xsd:element name="magnitude" minOccurs="1" maxOccurs="1">
          <xsd:simpleType>
            <xsd:restriction base="xsd:integer">
              <xsd:minInclusive value="0"/>
              <xsd:maxInclusive value="100"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="confidence" minOccurs="0" maxOccurs="1">
          <xsd:simpleType>
```

```
            <xsd:restriction base="xsd:integer">
              <xsd:minInclusive value="0"/>
              <xsd:maxInclusive value="100"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="significance" minOccurs="1" maxOccurs="1">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="critical"/>
              <xsd:enumeration value="important"/>
              <xsd:enumeration value="informational"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="type" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="ifmap:multiValueMetadataAttributes"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

A.4. ifmap-2.0.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
   targetNamespace="http://www.trustedcomputinggroup.org/2010/IFMAP/2/ifmap.wsdl"
 xmlns:ifmapwsdl="http://www.trustedcomputinggroup.org/2010/IFMAP/2/ifmap.wsdl"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <wsdl:types>
    <xsd:schema>
      <xsd:import namespace="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
          schemaLocation="ifmap-base-2.0v17.xsd"></xsd:import>
      <xsd:import
          namespace="http://www.trustedcomputinggroup.org/2010/IFMAP-METADATA/2"
          schemaLocation="ifmap-metadata-2.0v8.xsd"></xsd:import>
    </xsd:schema>
  </wsdl:types>

    <wsdl:message name="Response">
        <wsdl:part name="response" element="ifmap:response"></wsdl:part>
    </wsdl:message>

    <wsdl:message name="PublishRequest">
        <wsdl:part name="request" element="ifmap:publish"></wsdl:part>
    </wsdl:message>

    <wsdl:message name="SearchRequest">
        <wsdl:part name="request" element="ifmap:search"></wsdl:part>
    </wsdl:message>
```

```
  <wsdl:message name="SubscribeRequest">
      <wsdl:part name="request" element="ifmap:subscribe"></wsdl:part>
  </wsdl:message>

  <wsdl:message name="PollRequest">
      <wsdl:part name="request" element="ifmap:poll"></wsdl:part>
  </wsdl:message>

  <wsdl:message name="PurgePublisherRequest">
      <wsdl:part name="request" element="ifmap:purgePublisher"></wsdl:part>
  </wsdl:message>

  <wsdl:message name="NewSessionRequest">
      <wsdl:part name="request" element="ifmap:newSession"></wsdl:part>
  </wsdl:message>

  <wsdl:message name="RenewSessionRequest">
      <wsdl:part name="request" element="ifmap:renewSession"></wsdl:part>
  </wsdl:message>

  <wsdl:message name="AttachSessionRequest">
      <wsdl:part name="request" element="ifmap:attachSession"></wsdl:part>
  </wsdl:message>

  <wsdl:message name="EndSessionRequest">
      <wsdl:part name="request" element="ifmap:endSession"></wsdl:part>
  </wsdl:message>

<wsdl:message name="DumpRequest">
    <wsdl:part name="request" element="ifmap:dump"></wsdl:part>
  </wsdl:message>

  <wsdl:portType name="IfmapPortType">
      <wsdl:operation name="publish">
          <wsdl:input message="ifmapwsdl:PublishRequest"></wsdl:input>
          <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="search">
          <wsdl:input message="ifmapwsdl:SearchRequest"></wsdl:input>
          <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="subscribe">
          <wsdl:input message="ifmapwsdl:SubscribeRequest"></wsdl:input>
          <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="poll">
          <wsdl:input message="ifmapwsdl:PollRequest"></wsdl:input>
          <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="purgePublisher">
          <wsdl:input message="ifmapwsdl:PurgePublisherRequest"></wsdl:input>
          <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="newSession">
          <wsdl:input message="ifmapwsdl:NewSessionRequest"></wsdl:input>
          <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="attachSession">
          <wsdl:input message="ifmapwsdl:AttachSessionRequest"></wsdl:input>
          <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
      </wsdl:operation>
```

```
        <wsdl:operation name="renewSession">
            <wsdl:input message="ifmapwsdl:RenewSessionRequest"></wsdl:input>
            <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="endSession">
            <wsdl:input message="ifmapwsdl:EndSessionRequest"></wsdl:input>
            <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
        </wsdl:operation>
    <wsdl:operation name="dump">
          <wsdl:input message="ifmapwsdl:DumpRequest"></wsdl:input>
          <wsdl:output message="ifmapwsdl:Response"></wsdl:output>
        </wsdl:operation>
    </wsdl:portType>

    <wsdl:binding name="Service" type="ifmapwsdl:IfmapPortType">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
           style="document"/>
        <wsdl:operation name="publish">
            <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="search">
            <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="subscribe">
            <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="poll">
            <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="purgePublisher">
            <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
        </wsdl:operation>
```

```
        <wsdl:operation name="newSession">
           <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="attachSession">
            <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
    </wsdl:operation>
        <wsdl:operation name="renewSession">
           <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
        </wsdl:operation>
        <wsdl:operation name="endSession">
           <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    <wsdl:operation name="dump">
           <soap:operation soapAction=""/>
            <wsdl:input>
                <soap:body parts="request" use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body parts="response" use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>

    <wsdl:service name="IfmapService">
        <wsdl:documentation>IF-MAP Service</wsdl:documentation>
        <wsdl:port name="IfmapPort" binding="ifmapwsdl:Service">
    <soap:address location="https://localhost:8444/axis2/services/IfmapService"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

A.5. soap11.xsd

```
<?xml version='1.0' encoding='UTF-8' ?>
```

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:tns="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
           targetNamespace="http://schemas.xmlsoap.org/soap/envelope/" >

     <xs:import namespace="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
        schemaLocation="ifmap-base-2.0v17.xsd"/>

  <!-- Envelope, header and body -->
  <xs:element name="Envelope" type="tns:Envelope" />
  <xs:complexType name="Envelope" >
    <xs:sequence>
      <xs:element ref="tns:Header" minOccurs="0" />
      <xs:element ref="tns:Body" minOccurs="1" />
      <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
         processContents="lax" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>

  <xs:element name="Header" type="tns:Header" />
  <xs:complexType name="Header" >
    <xs:sequence>
      <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
         processContents="lax" />
    </xs:sequence>
    <xs:anyAttribute namespace="##other" processContents="lax" />
  </xs:complexType>

  <xs:element name="Body" type="tns:Body" />

  <xs:complexType name="Body" >
  <xs:choice>
    <xs:element ref="ifmap:newSession"/>
    <xs:element ref="ifmap:publish"/>
    <xs:element ref="ifmap:search"/>
    <xs:element ref="ifmap:subscribe"/>
    <xs:element ref="ifmap:poll"/>
    <xs:element ref="ifmap:dump"/>
    <xs:element ref="ifmap:purgePublisher"/>
    <xs:element ref="ifmap:attachSession"/>
    <xs:element ref="ifmap:renewSession"/>
    <xs:element ref="ifmap:endSession"/>
    <xs:element ref="ifmap:response"/>
  </xs:choice>
    <xs:anyAttribute namespace="##any" processContents="lax" >
    <xs:annotation>
      <xs:documentation>
      Prose in the spec does not specify that attributes are allowed on the Body
         element
    </xs:documentation>
    </xs:annotation>
  </xs:anyAttribute>
  </xs:complexType>


  <!-- Global Attributes.  The following attributes are intended to be usable via
     qualified attribute names on any complex type referencing them.  -->
  <xs:attribute name="mustUnderstand" >
     <xs:simpleType>
     <xs:restriction base='xs:boolean'>
```

```
      <xs:pattern value='0|1' />
    </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="actor" type="xs:anyURI" />

  <xs:simpleType name="encodingStyle" >
    <xs:annotation>
    <xs:documentation>
      'encodingStyle' indicates any canonicalization conventions followed in the
         contents of the containing element.  For example, the value
         'http://schemas.xmlsoap.org/soap/encoding/' indicates the pattern
         described in SOAP specification
    </xs:documentation>
  </xs:annotation>
    <xs:list itemType="xs:anyURI" />
  </xs:simpleType>

  <xs:attribute name="encodingStyle" type="tns:encodingStyle" />
  <xs:attributeGroup name="encodingStyle" >
    <xs:attribute ref="tns:encodingStyle" />
  </xs:attributeGroup>

  <xs:element name="Fault" type="tns:Fault" />
  <xs:complexType name="Fault" final="extension" >
    <xs:annotation>
    <xs:documentation>
      Fault reporting structure
    </xs:documentation>
  </xs:annotation>
    <xs:sequence>
      <xs:element name="faultcode" type="xs:QName" />
      <xs:element name="faultstring" type="xs:string" />
      <xs:element name="faultactor" type="xs:anyURI" minOccurs="0" />
      <xs:element name="detail" type="tns:detail" minOccurs="0" />
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="detail">
    <xs:sequence>
      <xs:any namespace="##any" minOccurs="0" maxOccurs="unbounded"
         processContents="lax" />
    </xs:sequence>
    <xs:anyAttribute namespace="##any" processContents="lax" />
  </xs:complexType>

</xs:schema>
```