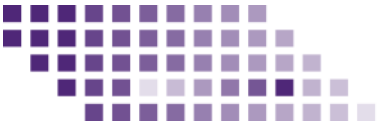
Abteilung Informatik - Fakultät IV - Hochschule Hannover



Trust@FHH

Fachhochschule Hannover
University of Applied Sciences and Arts

Analyse und Implementierung des IF-MAP 2.1 Protokolls

Bachelorarbeit im Studiengang
Angewandte Informatik

Joram Knaack
joram@knaack.name

1. Oktober 2012

Beteiligte Personen

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

Betreuer
Ingo Bente M.Sc.
Hochschule Hannover
Ricklinger Stadtweg 120
D-30459 Hannover
E-Mail: ingo.bente@fh-hannover.de
Tel.: +49 (0)511 9296-1828

Autor
Joram Knaack
Buchenweg 10
D-30916 Isernhagen
E-Mail: joram@knaack.name

Selbstständigkeitserklärung

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

Isernhagen, den 1. Oktober 2012

Joram Knaack

Inhaltsverzeichnis

1 Einleitung

IF-MAP (Interface to Metadata Access Point) ist ein standardisiertes, Client-Server basiertes Protokoll zum Austausch von Metadaten. Es wurde primär dafür entwickelt, interoperabel sicherheitsrelevante Daten zwischen beliebigen Netzwerkkomponenten (wie Paketfilter, Switches, AAA-Systemen) austauschen zu können. Das Kommunikationsprotokoll basiert auf einem inhaltsbasierten Publish-Subscribe Mechanismus: MAP-Clients können über einen MAP-Server neue Metadaten veröffentlichen (publish), nach vorhandenen Metadaten suchen (search) oder sich über Änderungen von Metadaten informieren lassen (subscribe).[2]

1.1 Trusted Computing Group

Die Trusted Computing Group[1] (TCG) hat im Mai 2008 den offenen Standard für Sicherheits- und Netzwerkkoordination IF-MAP veröffentlicht. Dieser steht jedem kostenlos zur Implementation zu Verfügung. [1]

Inzwischen setzen mehrere namhafte Unternehmen das IF-MAP Protokoll ein und haben bereits bewiesen, dass die Kompatibilität zwischen Endgeräten unterschiedlicher Hersteller sichergestellt ist. Halbjährlich treffen sich hierzu Vertreter partizipierender Firmen und Forschungseinrichtungen zum IF-MAP Plugfest, um die Interoperabilität ihrer Implementationen untereinander zu prüfen. Zuletzt trafen diese sich im März 2012 in Darmstadt am Fraunhofer-Institut für Sichere Informationstechnologie. Dabei wurden, mit über 50 Testläufen, die verschiedenen Systeme in ihrer Kommunikation untereinander überprüft und deren Interoperabilität nachgewiesen.[2]

Im Mai 2012 wurde dann die bisher aktuelle Version 2.0 der IF-MAP Spezifikation durch die Trusted Computing Group ratifiziert und als Version 2.1 veröffentlicht. Hierbei wurden einige Punkte verdeutlicht und genauer spezifiziert. Allerdings sind auch zwei grundlegend neue Funktionen aufgenommen worden. Insbesondere sind hier die erweiterten Bezeichner (*engl.* extended identifier) zu nennen, welche es ermöglichen beliebig komplexe Informationen als Referenz für Metadaten auf einem MAP Server zu veröffentlichen.

1.2 trust@FHH

Die Arbeitsgruppe trust@FHH an der Hochschule Hannover beschäftigt sich mit unterschiedlichsten Aspekten der IT Sicherheit. Der Schwerpunkt liegt hierbei auf der Entwicklung von Softwarekomponenten im Zusammenhang mit den von der Trusted Computing Group (TCG) herausgegebenen Spezifikationen zu Trusted

[1] `http://www.trustedcomputinggroup.org/`

[2] `http://www.sit.fraunhofer.de/de/medien-publikationen/pressemitteilungen/2012/03042012-plugfest.html`

Computing[3] und Trusted Network Connect[4].

In Zusammenarbeit mit der TCG partizipiert trust@FHH auch an der Festlegung von zukünftigen Spezifikationen. Außerdem finden Entwicklungen im Open-Source Bereich unter Zusammenarbeit mit anderen Kooperationspartnern der TCG im Rahmen des ESUKOM-Projektes statt. Das ESUKOM Projekt wird vom Bundesministerium für Bildung und Forschung gefördert und hat zum Ziel IF-MAP als Sicherheitslösung zum Schutz für Unternehmensnetzwerke zu etablieren.[5]

1.3 Gegenstand, Anforderungen und Ziele

IF-MAP bietet, durch die Möglichkeit zur Integration verschiedenster Netzwerkkomponenten und Anwendungen, mit einem MAP Server eine zentrale Stelle, an der sicherheitsrelevante Informationen über ein Netzwerk gesammelt werden. Dazu werden die unterschiedlichsten Daten innerhalb des MAP Servers durch einen ungerichteten Graphen strukturiert und miteinander in Beziehung gesetzt. Dabei können die so gesammelten Informationen äußerst umfangreich werden und in praktischen Szenarien sehr komplexe Begebenheiten im Netzwerk darstellen.

Die TCG hat im Mai 2012 die Version 2.1 des IF-MAP Protokolls veröffentlicht. Diese bietet einige Neuerungen im Vergleich zur Version 2.0 – insbesondere die Verwendung der neu geschaffenen erweiterten Bezeichner.

Das Ziel der Arbeit ist es vorhandene Softwarekomponenten, die IF-MAP 2.0 implementieren, auf IF-MAP 2.1 zu migrieren. Dazu zählen:

- **irond** - Die experimentelle Referenzimplementation eines MAP Servers der Hochschule Hannover.
- **ifmapj** - Eine Umsetzung der IF-MAP Spezifikation zur Kommunikation mit einem MAP Server als Bibliothek zur Verwendung in eigenen Implementationen.
- **ifmapj-examples** - Eine exemplarische Umsetzung zur Nutzung der ifmapj-Bibliothek.
- **irongui** - Das IF-MAP Visualisierungswerkzeug zur graphischen Darstellung von Beziehungen zwischen Metadaten und Bezeichnern.

Die technologische Grundlage für diese Arbeit bilden die Entwicklungen der Forschungsgruppe Trust@FHH[6] an der Hochschule Hannover. Erweiterungen müssen dabei rückwärts-kompatibel sein (d.h. jede Komponente muss sowohl IF-MAP 2.0 als auch IF-MAP 2.1 korrekt implementieren). Darüber hinaus sollen die neuen

[3] http://en.wikipedia.org/wiki/Trusted_Computing
[4] http://en.wikipedia.org/wiki/Trusted_Network_Connect
[5] http://www.esukom.de/cms/front_content.php?idcat=4
[6] http://trust.inform.fh-hannover.de/

Features der Version 2.1 prototypisch in den ifmapj-examples demonstriert werden, insbesondere die erweiterten Bezeichner.

Als zusätzliche Anforderung ist der geänderte Quellcode entsprechend zu dokumentieren, bestehende Codestrukturen beizubehalten und keine weiteren Abhängigkeiten von anderen externen Bibliotheken oder Programmen einzuführen. Änderungen am Quellcode sollen über das von der Hochschule Hannover bereitgestellte Versionskontrollsystem Subversion[7] (als sog. Branch) nachvollziehbar sein. Im Anschluss an diese Arbeit soll der Quellcode in die Hauptlinie (Trunk) der einzelnen Programme eingepflegt werden.

1.4 Aufbau der Arbeit

Die Umsetzung der Spezifikation ist Hauptbestandteil der Arbeit und in die folgenden Abschnitte unterteilt:

- **Grundlagen** - Übersicht über die anzupassenden Softwarekomponenten.
- **Analyse** - Ermitteln notwendiger Anpassungen und Bewertung der Änderungen in der IF-MAP Spezifikation.
- **Konzept** - Planungen zur Umsetzung von Anpassungen und Erweiterungen in den einzelnen Softwarekomponenten.
- **Implementierung** - Dokumentation der in jedem Programm vorgenommenen Implementationen.
- **Tests** - Durchführung des IF-MAP 2.0 Nachweistests mit dem überarbeiteten MAP Server irond sowie Anpassungen an den Komponententests einzelner Funktionen und Klassen.
- **Beispiele** - Beschreibung der in den ifmapj-examples neu hinzugefügten Implementationsbeispiele.
- **Bewertung** - Zusammenfassung der erreichten Ziele dieser Arbeit sowie der erfüllten Anforderungen zur Implementation.
- **Ausblick** - Mögliche zukünftige Änderungen an der Spezifikation und weitere Ansätze zur Implementation.

[7] `http://subversion.apache.org/`

1.5 Typographische Konventionen

Zur Unterscheidung bestimmter Begrifflichkeiten werden diese in einer anderen Schriftart dargestellt:

Schriftart	Beschreibung
Kursivschrift	Schlüsselwörter und Attribute
`Serifenschrift`	Klassen-, Methoden- und Dateinamen

Fachbegriffe sind weitestgehend in ihren deutschen Benennungen verwandt worden. Nur in Einzelfällen war es notwendig diese in englischer Sprache zu gebrauchen. Zur Vermeidung von Verwechslungen ist bei der Einführung neuer Fachbegriffe auch deren englische Übersetzung angegeben.

Die in der Arbeit aufgeführten Listings sind grundsätzlich nur Auszüge aus den einzelnen Programmen und dienen nur zur Veranschaulichung. Sie sind nicht eigenständig lauffähig. Sofern nicht anderweitig erwähnt, sind alle Programmauszüge, zumindest in sinngemäß gleicher Form, auch im Quellcode der behandelten Programme vorhanden.

2 Grundlagen

Zum besseren Verständnis und der Vollständigkeit halber werden in diesem Kapitel die Grundlagen kurz vorgestellt, auf denen diese Arbeit basiert. Hierbei werden anfänglich die einzelnen Technologien und das IF-MAP Protokoll sowie dessen Rolle innerhalb der TNC Architektur näher erläutert. Im Anschluss werden dann die zu untersuchenden Softwarekomponenten vorgestellt.

2.1 XML, XSD, XSLT und SOAP

XML[8] steht für e**X**tensible **M**arkup **L**anguage. Ziel von XML ist es, ein einheitliches Format für unterschiedlichste Dokumente bereitzustellen. Dazu kann in Form eines XML Schemas[9] (XSD) der Aufbau des Dokumentes festgelegt werden. Im Prinzip können XML Schemata als eine Erweiterung von DTDs verstanden werden, da in einem XML Schema neben der Dokumentstruktur auch Datentypen und Gültigkeitsbereiche festgelegt werden. In einer DTD[10] (Document Type Definition) wird die grundlegende Struktur eines XML Dokumentes festgelegt. Zusätzlich zur Prüfung ob ein Dokument alle notwendigen Elemente enthält, kann ein XML Dokument noch mittels eines XML Schemas validiert werden. Dabei werden die in dem Dokument vorgefundenen Elemente und Attribute sowie deren Werte gegen die Definitionen im Schema geprüft. So kann festgestellt werden ob sich unzulässige Elemente oder Attribute in dem Dokument befinden oder die enthaltenen Daten ungültige Werte aufweisen.

Per XSL Transformation[11] (XSLT) lassen sich Dokumente von einer Schema-Definition in eine andere überführen. Dabei werden gleiche Felder beider Dokumente miteinander verknüpft. Zusätzlich lassen sich auch mathematische und logische Operationen definieren, die das Ergebnis aus Werten des Ursprungsdokumentes in ein neues Dokument übernehmen.

XML eignet sich insbesondere zur Übertragung von Daten über ein Netzwerk, so auch dem Internet. Dabei kommt meistens SOAP[12] zum Einsatz, was die Aufgabe hat den Austausch von XML-Nachrichten zwischen einem Client und Server in der Implementation möglichst einfach zu gestalten.

2.2 IF-MAP

Das IF-MAP Protokoll setzt XML als Transportmedium ein, um Informationen zwischen Client und Server über einen sicheren Kanal austauschen zu können. Dazu wurde mit einem XML Schema definiert, wie die Kommunikation zwischen

[8] http://en.wikipedia.org/wiki/XML
[9] http://en.wikipedia.org/wiki/XML_schema
[10] http://en.wikipedia.org/wiki/Document_Type_Definition
[11] http://en.wikipedia.org/wiki/XSLT
[12] http://en.wikipedia.org/wiki/SOAP

Server und Client per SOAP erfolgen soll. Die eigentliche Übertragung erfolgt zum Schutz sensibler Daten ausschließlich authentifiziert und verschlüsselt. Hierzu steht ein einfacher Mechanismus per Benutzername und Passwort sowie eine zertifikatsbasierte Authentifizierung zur Verfügung.

Ein MAP Client kann ein Gerät innerhalb der Netzwerkinfrastruktur oder eine Anwendung innerhalb der Organisationsstruktur sein. Dieser kann verschiedene Operationen auf einem MAP Server durchführen. Hierbei dient der MAP Server als Datenquelle für Informationen, aber auch als Ziel für die Veröffentlichung von eigenen Daten sowie von Daten angeschlossener Peripherie. Das IF-MAP Protokoll spezifiziert die möglichen Befehle zur Datenübertragung sowie die Art und den Umfang der zu übertragenden Daten. Dabei wird zwischen zwei grundlegenden Datentypen unterschieden: Metadaten und Bezeichner. Metadaten dienen dazu verschiedene Bezeichner miteinander in Verbindung zu bringen und somit einen Zusammenhang zu weiteren Bezeichnern und Metadaten herzustellen. Hierzu können Metadaten jeweils mit bis zu zwei Bezeichnern verknüpft werden. Ein Bezeichner wiederum kann beliebig viele Metadaten auf sich vereinen.

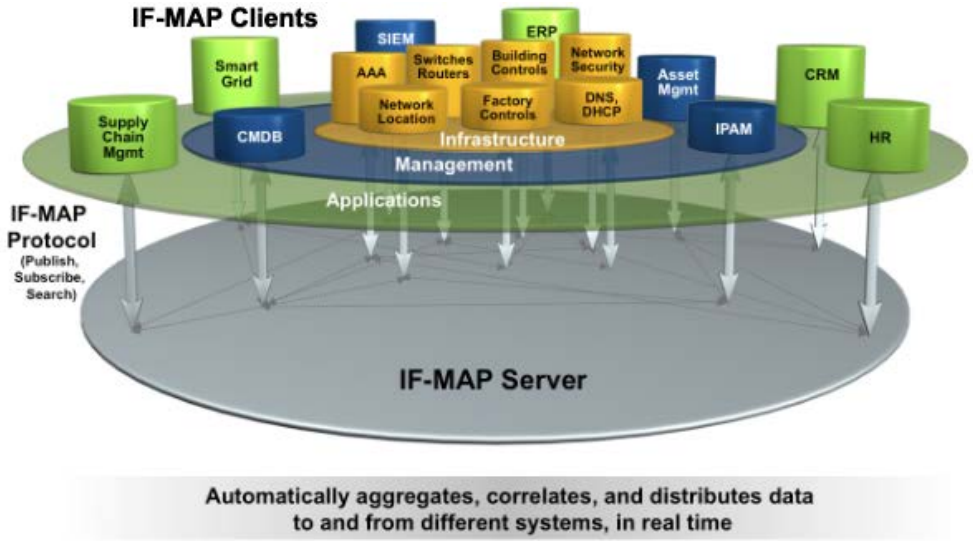


Abbildung 1: Ein MAP Server als zentrale Stelle zur Verwaltung sicherheitsrelevanter Daten im Netzwerk, Quelle: `http://www.if-map.org/what_is_if-map`

Ein MAP Server soll die Möglichkeit zum Abonnement von Daten geben. Dabei kann ein MAP Client eine Anfrage an den Server stellen, welche Daten er abonnieren möchte. Über einen separaten Kommunikationskanal wird der Client bei der Veröffentlichung oder Änderung von entsprechenden Informationen darüber in Kenntnis gesetzt. Außerdem kann ein MAP Client Suchanfragen an den Server zum Abruf veröffentlichter Informationen stellen. Die Angabe von zusätzlichen Filterkriterien kann das Suchergebnis bereits vor der Rückübertragung einschränken und die Netzwerklast minimieren.

2.3 MAP Server irond

Der MAP Server „irond“ (siehe A.3 Verwendeter Quellcode) entstand 2008 innerhalb der trust@FHH Projektgruppe im Rahmen des „iron“ Projektes. Seitdem wird der MAP Server kontinuierlich verbessert und weiter entwickelt.

Der Server bietet eine vollständige Implementation der in den IF-MAP Spezifikationen geforderten Operationen. Dazu besitzt der Server eine Schnittstelle auf Basis von XML-Nachrichten zur Kommunikation mit den verbundenen Clients. Per „$unmarshalling$“ übersetzt der Server die erhaltenen XML-Nachrichten zur internen Verwaltung in Java-Objekte, welche er auch wieder per „$marshalling$“ in XML-Nachrichten zurück transformieren kann. Hierzu wurden drei Abstraktionsschichten erstellt, die jeweils für das Verbindungsmanagement, die XML Transformationen und die Verwaltung von Datenobjekten zuständig sind. Die Transformation in XML Nachrichten (und umgekehrt) erfolgt per JAXB-RI[13] und die Netzwerkübertragung per SOAP.

Die empfangenen Daten der Clients werden nicht persistent gespeichert. Diese werden ausschließlich während der Laufzeit im Arbeitsspeicher vorgehalten. Irond nutzt zum Speichern von Konfigurationsdaten einfache Textdateien. Grundlegend für den Betrieb ist hier die Konfigurationsdatei `ifmap.properties`. Diese beinhaltet alle notwendigen Konfigurationen für den Betrieb des Servers. Im Rahmen dieser Arbeit sind einige Optionen hinzugekommen, welche im Anhang unter A.5.2 zusammenfassend beschrieben sind.

2.4 ifmapj-Bibliothek

Die Java Klassenbibliothek „ifmapj“ (siehe A.3 Verwendeter Quellcode) entstand an der Hochschule Hannover und wird durch die Arbeitsgruppe trust@FHH aktiv weiter entwickelt.

Die Erstellung von Softwarekomponenten mit Anbindung an einen MAP Server wird durch die Verwendung der ifmapj-Bibliothek deutlich vereinfacht. So ist es möglich mit einfachen und logischen Java-Objekten zu arbeiten anstatt selbst XML Nachrichten erstellen zu müssen. Zusätzlich ist die gesamte Kommunikationsschicht inkl. Authentifizierung in der Bibliothek enthalten. Die Nutzung ermöglicht es dem Entwickler sich auf die wesentlichen Aspekte seiner Implementation zu konzentrieren. Außerdem ist die Erweiterung bestehender Anwendungen, die noch keine IF-MAP Schnittstelle besitzen, durch die Verwendung der Bibliothek in sehr kurzer Zeit möglich.

Hauptkriterium ist die Entwicklung von IF-MAP fähigen Anwendungen so einfach wie möglich zu gestalten. Als Einstiegshilfe gehören zu der ifmapj-Bibliothek auch die ifmapj-examples, welche exemplarisch die Verwendung der Bibliothek aufzeigen. Sämtliche Klassen und Funktionen sind vollständig dokumentiert und stellen

[13] `http://jaxb.java.net/`

per Javadoc eine kontextbezogene Hilfe zur intuitiven Nutzung zur Verfügung.

2.5 irongui Visualisierungswerkzeug

Das Visualisierungswerkzeug „irongui“ (siehe A.3 Verwendeter Quellcode) wurde durch Tobias Ruhe im Rahmen seiner Bachelorarbeit „Visualisierung von Informationen einer zentralen Netzwerk-Datenbank“[14] entwickelt und wird derzeit durch die trust@FHH Gruppe weiter gepflegt.

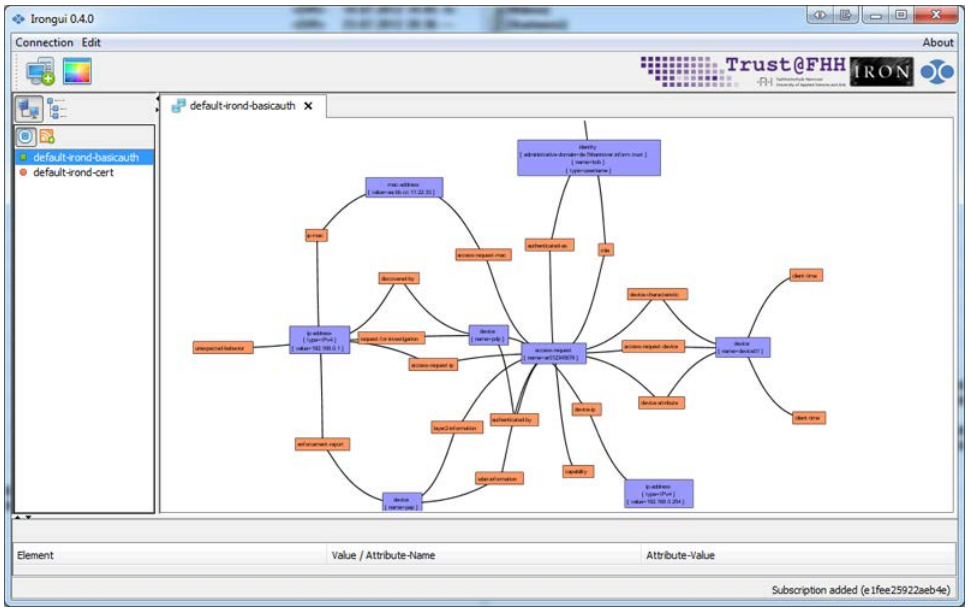


Abbildung 2: Visualisierung eines IF-MAP Metagraphen durch „irongui“

Die Verwendung von irongui ermöglicht dem Nutzer eine einfache und übersichtliche graphische Darstellung des Graphen eines MAP Servers. Über die bestehende XML-Schnittstelle des Servers kann irongui die vorgehaltenen Metadaten und Bezeichner abrufen und ihre Zusammenhänge entsprechend optisch darstellen.

Hierzu wurde bisher immer ein bereits bekannter Bezeichner als Einstiegspunkt benötigt, denn ein vollständiger Abruf aller im Server vorgehaltenen Daten war nicht möglich. Auch sah die Spezifikation hierzu keine entsprechende Funktion vor. Aus diesem Grund wurde von Herrn Ruhe der MAP Server irond um eine sog. Dump-Funktion erweitert, die es ermöglicht sämtliche Daten des Servers abzurufen, ohne einen bekannten Bezeichner als Einstiegspunkt zu haben. Zur Serverkommunikation wird die ifmapj-Bibliothek verwendet.

Irongui dient dem Forschungsprojekt VisITMeta[15] als Grundlage zur Entwicklung eines umfangreichen Werkzeugs zur Darstellung sehr komplexer und um-

[14] http://www.esukom.de/cms/upload/pdfs/ba-tobias-ruhe.pdf
[15] http://trust.inform.fh-hannover.de/joomla/index.php/projects/visitmeta

fangreicher Metadaten innerhalb eines MAP Graphen. Dabei sollen zusätzliche Funktionen zur Suche und Darstellung sowie eine Historie vergangener Daten implementiert werden.

3 Analyse der IF-MAP Spezifikation

In diesem Kapitel werden die Unterschiede zwischen den Versionen 2.0 und 2.1 der IF-MAP Spezifikation analysiert. Darüber hinaus wird auch die bereits vorhandene Implementation auf Vollständigkeit und Korrektheit untersucht. Zusätzlich werden die Änderungen in ihrem Zusammenhang bewertet.

3.1 Bezeichner

Ein Bezeichner (*engl.* identifier) stellt ein Objekt im Netzwerk dar. Dabei kann es sich um ganz unterschiedliche Objekte handeln. Einige standardmäßig genutzte Typen sind von der TCG bereits festgelegt worden. So wäre beispielsweise der Gerätebezeichner (*engl.* device identifier), MAC-Adressenbezeichner (*engl.* mac address identifier) und IP-Adressenbezeichner (*engl.* ip address identifier) zu nennen. Ein Schwerpunkt der IF-MAP 2.1 Spezifikation ist die Schaffung der erweiterten Bezeichner (*engl.* extended identifier), die beliebig komplexe Objektinformationen beinhalten können.

Die Möglichkeiten zur Erstellung und die Gleichheitsbedingungen von Bezeichnern wurden erweitert und näher spezifiziert. In diesem Abschnitt werden die Änderungen, die sich aus der überarbeiteten Spezifikation ergeben, genauer analysiert.

3.1.1 Bedingungen zur Nutzung einer administrativen Domäne

Eine administrative Domäne (*engl.* administrative domain) ist ein zusätzliches Kriterium zur Identifikation von Netzwerkkomponenten, die ggf. mehrere Instanzen des gleichen Bezeichners erstellen. Hierbei werden unter Umständen sogar die gleichen Informationen veröffentlicht, so dass es kein anderes Unterscheidungskriterium außer der administrativen Domäne gibt. Durch dessen Nutzung kann sichergestellt werden, dass ein Bezeichner eindeutig zugeordnet wird und Verwechslungen durch eine konsequente Verwendung ausgeschlossen werden können.

Die IF-MAP Spezifikation sieht ausdrücklich vor, dass bei der Angabe einer adminstrativen Domäne auf Groß- und Kleinschreibung geachtet werden muss. Sollte jedoch eine externe Quelle dies nicht tun, so ist von dem Client die administrative Domäne in Kleinbuchstaben zu normalisieren.[3, S. 15] In Frage kommen hier beispielsweise Windows-Umgebungen, bei denen der FQDN[16] bei einer Migration zu Active-Directory aus dem NetBIOS[17]-Namen des Computers ermittelt wurde.

Es werden außerdem Empfehlungen ausgesprochen, wie die Angabe einer administrativen Domäne jeweils erfolgen soll. So darf bei einem „$access - request$“ Bezeichner diese nicht mehr angegeben werden, da dessen Angabe als veraltet gekennzeichnet wurde. Außerdem kann sie grundsätzlich nicht für Gerätebezeich-

[16] http://en.wikipedia.org/wiki/Fully_qualified_domain_name
[17] http://en.wikipedia.org/wiki/Netbios

ner genutzt werden. Daher wird auch die allgemeine Empfehlung ausgesprochen, sofern eine MAP Client hierzu eine Standardkonfiguration besitzt, in dieser die administrative Domäne nicht mehr zu setzen.[3, S. 15]

Bei einem „$ip - address$“ Bezeichner soll die administrative Domäne der sog. Routing Domäne entsprechen. Diese gibt an, welche Systeme zur gleichen Netzwerkverwaltung gehören. Wobei hingegen beim „$mac - address$“ Bezeichner die administrative Domäne das lokale Netzwerksegment darstellen soll. Bei Identitätsbezeichnern (*engl.* identity identifiers) soll sie als zugehörige authentifizierende Stelle angeben werden. Lediglich für erweiterte Bezeichner ist die Angabe einer administrativen Domäne nicht gestattet.[3, S. 15f] Details hierzu werden im Kapitel 3.1.5 behandelt.

In Abschnitt 3.2.2.5 der IF-MAP Spezifikation wird explizit auf den Zusammenhang zwischen MAC-Adresse und der optionalen administrativen Domäne bei einem „$mac - address$“ Bezeichner eingegangen. Hierbei wird klargestellt, dass ein Bezeichner nur einem anderen Bezeichner gleichzusetzen ist, wenn sowohl die MAC-Adresse als auch die angegebene administrative Domäne übereinstimmt. Dies soll das Risiko manipulierter MAC-Adressen verringern und das sog. „mac address spoofing“ erschweren, da zusätzlich zur MAC-Adresse dem Angreifer auch die zugehörige administrative Domäne bekannt sein muss.[3, S. 19f]

Die Angabe einer administrativen Domäne soll in erster Linie eine organisatorische Struktur schaffen und im Falle des „$mac - address$“ Bezeichners auch für zusätzliche Sicherheit sorgen. Dessen Verwendung ist allerdings nicht zwingend notwendig, aber die genannten Gründe sprechen jedoch eindeutig für eine Nutzung. Es empfiehlt sich diese konsequent einzusetzen, insbesondere um die Organisationsstruktur in komplexen standort-übergreifenden Netzwerken übersichtlich zu gestalten.

3.1.2 Änderungen am Gerätebezeichner

Bisher war es erlaubt einem Gerätebezeichner entweder den Namen eines AIK (*engl.* **A**ttestation **I**dentity **K**ey) mitzugeben oder einen eindeutigen Gerätenamen.

Bei einem AIK handelt es sich um ein RSA Schlüsselpaar zur Beglaubigung einer Plattform durch einen unabhängigen Dritten. Dieses wird mittels eines TPM[18] generiert und kann beispielsweise zur Nutzung kostenpflichtiger Dienste von Anbietern genutzt werden. Zur Beglaubigung gegenüber einem Anbieter wird die Systemkonfiguration einer Plattform mit dem AIK signiert und zusammen mit dem öffentlichen Schlüssel an diesen übermittelt. Der Anbieter wiederum kann über Dritte die Validität des Schlüssels und somit auch die der verwendeten Systemkonfiguration überprüfen.

Der Name dieses Schlüssels konnte bisher über das Attribut „$aik - name$“ für

[18] `http://en.wikipedia.org/wiki/Trusted_Platform_Module`

einen Gerätebezeichner gesetzt werden. Es ist jedoch nicht gewährleistet, dass der verwendete Name immer einzigartig ist. Zur Vermeidung von Verwechslungen darf das Attribut „$aik - name$“ bei der Erstellung eines Gerätebezeichners vom Client nicht mehr genutzt werden. Ein MAP Client soll jetzt ausschließlich das Attribut „$name$“ verwenden.[3, S. 16]

Zur Erstellung eines eindeutigen Namens werden genaue Vorgaben gemacht, wie dieser zu generieren ist. Er besteht entweder aus einer kryptographisch starken Zufallszahl, aus einer UUID[19] oder einem Ordinalwert (einfacher Zahlenwert), dem die vom Server zugeordnete Publisher ID vorangestellt wird. Zur Wahrung der Abwärtskompatibilität muss ein MAP Server hingegen beide Attribute weiterhin verarbeiten können und dabei auch konsequent zwischen den Attributtypen unterscheiden.

Das Generieren eines eindeutigen Gerätenamens wird durch diese Vorgaben erschwert, daher ist eine Bereitstellung von entsprechenden Funktionalitäten in der ifmapj-Bibliothek sinnvoll.

3.1.3 Regeln zum Vergleich von Identitätsbezeichnern

Ein Identitätsbezeichner kann einen Endbenutzer, ein Gerät, eine Anwendung oder eine beliebige logische oder physische Einheit repräsentieren. Dabei kann eine Organisation in mehrere administrative Domänen unterteilt sein. Damit ein Gerätebezeichner eindeutig ist, kann zusätzlich seine jeweilige administrative Domäne angegeben werden.[3, S. 17]

Eine Ausnahme hiervon sind die erweiterten Bezeichner, die im Abschnitt 3.1.5 näher behandelt werden.

Um verschiedene Identitäten, deren Namen unter Umständen gleich sind, unterscheiden zu können, ist die Angabe eines Identitätstypen notwendig. Der Typ muss bei einem Vergleich von Identitäten ebenfalls mit berücksichtigt werden. Um einen einfachen Binärvergleich zu ermöglichen, müssen die Daten bei einigen Identitätstypen zusätzlich normalisiert werden.

Dabei muss der „$distinguished - name$“ Typ speziell behandelt werden. Details hierzu sind im nächsten Abschnitt zu finden. Die Normalisierung erfolgt ansonsten indem alle Daten, bei denen keine Unterscheidung in der Groß- und Kleinschreibung vorgenommen wird, in Kleinbuchstaben umgewandelt werden.[3, S. 17]

Die Analyse ergibt, dass die folgenden Identitätstypen hiervon betroffen sind:

[19] http://en.wikipedia.org/wiki/Universally_Unique_Identifier
[6] http://www.ietf.org/rfc/rfc1034
[7] http://www.ietf.org/rfc/rfc2822
[8] http://www.ietf.org/rfc/rfc3261

Identitätstyp	betroffene Daten
dns-name	FQDN oder Domainname (entsprechend RFC 1034[20])
email-address	Domain der E-Mailadresse (entsprechend RFC 2822[21])
sip-uri	vollständige SIP URI (entsprechend RFC 3261[22])
hip-hit	hexadezimale Darstellung (siehe [3, S. 18])
distinguished-name	*spezielle Behandlung nach XACML 2.0* (siehe 3.1.4)

3.1.4 Gleichheitsbedingungen von „distinguished-name“ Identitäten

Bei einem eindeutigen X.500[23] Namen (*engl.* X.500 distinguished name) handelt es sich um eine Zeichenkette, die aus ein oder mehreren Attributpaaren besteht. Dabei stellt die Zeichenkette einen global gültigen Namen innerhalb eines Verzeichnisses (z.B. LDAP[24]) dar, welcher ein Objekt eindeutig identifiziert. Dies soll eine weltweit eindeutige Unterscheidung von Personen und Systemen ermöglichen. Ein MAP Client soll einen eindeutigen X.500 Namen nach den Vorgaben der XACML 2.0[25] Spezifikation normalisieren und vergleichen können.[3, S. 17]

Zusammengefasst sind dies die XACML 2.0 Bedingungen, die zum Vergleich herangezogen werden:

- Normalisieren der Zeichenkette nach RFC 2253[26] durch entfernen unnötiger Leerzeichen.
- Sofern mehrere Attributpaare (RDN) vorhanden sind, müssen diese aufsteigend alphabetisch sortiert werden.
- Abschließend die einzelnen Attributpaare nach den Reglen in RFC 3280[27] vergleichen.

Der Vergleich von eindeutigen Namen nach XACML 2.0 beruht darauf, dass jeder Teil (RDN) in beiden Zeichenketten mit identischen Attributwerten vorhanden ist und dabei die Attributschlüssel eine unterschiedliche Reihenfolge sowie Groß- und Kleinschreibung aufweisen dürfen. Dies schließt einen einfachen Binärvergleich der Zeichenkette zur Identifizierung gleicher eindeutiger Namen aus.

Um die Interoperabilität zwischen unterschiedlichen Plattformen zu gewährleisten, soll vor der Übertragung eines eindeutigen Namens zwischen MAP Server und Client, die Zeichenkette UTF-8[28] encodiert werden.

[23] http://en.wikipedia.org/wiki/X.500
[24] http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol
[25] http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
[26] http://www.ietf.org/rfc/rfc2253
[27] http://www.ietf.org/rfc/rfc3280
[28] http://en.wikipedia.org/wiki/UTF-8

Die korrekte Behandlung eindeutiger X.500 Namen ist in der derzeitigen Version der IF-MAP Spezifikation noch optional und daher nicht zwingend vorgeschrieben. Eine Umsetzung wird jedoch empfohlen, da voraussichtlich in zukünftigen Versionen der Spezifikation das Verfahren notwendig wird.

3.1.5 Erweiterte Bezeichner

Grundlegend für die Entwicklung des IF-MAP Protokolls waren Anwendungsszenarien innerhalb der TNC Architektur. Dafür wurde mit IF-MAP eine universelle Möglichkeit zur Speicherung von Metadaten geschaffen. Dies weckte das Interesse anderer Anwendungsentwickler einen MAP Server auch für ihre Zwecke nutzen zu wollen.

Die strikte Vorgabe an verschiedenen Typen für Bezeichner schränkt die Nutzbarkeit in anderen Szenarien erheblich ein. Daher soll ein neuer Bezeichnertyp geschaffen werden, der es ermöglicht beliebig komplexe Daten miteinander zu verknüpfen. Allerdings ist es nicht möglich einen neuen Bezeichnertypen zu bestimmen, ohne dabei bestehende Implementationen der IF-MAP Spezifikation anpassen zu müssen. In dem zugrunde liegenden XML Basisschema `ifmap-base-2.0v17.xsd` sind die verwendbaren Typen bereits eindeutig spezifiziert und bei einer Typenerweiterung wäre das Schema anzupassen. Da bestehende Implementationen auf dem bereits bekannten Schema basieren, könnten sie den für sie unbekannten Bezeichner nicht verarbeiten. Bei einer korrekten Umsetzung würde der Bezeichner nur abgelehnt oder ignoriert werden. Im ungünstigsten Fall wäre auch ein Programmabbruch durch eine unbehandelte Ausnahme (*engl.* Exception) möglich.

Zur Wahrung der Abwärtskompatibilität wurde daher der bereits bestehende Identitätstyp „$other$“ genutzt. Dieser ermöglicht die Angabe zusätzlicher Typinformationen in Form des Attributs „$other - type - definition$“. Hierbei wurde bisher schon berücksichtigt, dass das Attribut „$other - type - definition$“ in dem zugrunde liegenden XML Basisschema beliebige Informationen enthalten kann. Jedoch sind die eigentlichen Nutzdaten nach wie vor auf das XML Attribut „$name$“ beschränkt.

Um diese Restriktion zu umgehen und gleichzeitig die Abwärtskompatibilität zu bestehenden Implementationen aufrecht zu erhalten, wurde festgelegt, dass der Bezeichnertyp „$other$“ in der neuen, von der TCG spezifizierten, Definition „$extended$“ nun auch vollständige XML Dokumente in gekapselter Form beinhalten kann.[3, S. 20]

Das verwendete XML Dokument kann beliebig komplex sein, sollte jedoch inkl. dem XML Gerüst zur Übertragung eines Bezeichners vom Typ „$other$“ nicht 1000 Bytes überschreiten, da dies die maximal zu unterstützende Größe eines Bezeichners ist. Es wird allerdings empfohlen, auch deutlich größere Bezeichner zu akzeptieren. Dies obliegt jedoch der jeweiligen Serverimplementation und kann nicht als Voraussetzung angenommen werden.[3, S. 15]

Die Angabe der administrativen Domäne erfolgt bei erweiterten Bezeichnern nicht wie gewohnt als Attribut des Bezeichners, sondern wird innerhalb des Namensattributs angegeben. Daher muss dem XML Dokument eines erweiterten Bezeichners ein Schema (XSD) zugrunde liegen, welches zumindest die administrative Domäne beinhaltet:

Listing 1: Das XML Schema als Basis für erweiterte Bezeichner

```
1 <xsd:complexType name="IdentifierType">
2    <xsd:attribute name="administrative-domain" type="xsd:string" use="
       required"/>
3 </xsd:complexType>
```

Zur Übertragung als erweiterter Bezeichner muss das XML Dokument speziell behandelt werden. Dazu sind die folgenden Schritte notwendig:

- Erstellen des XML Dokumentes basierend auf dem vorgegebenen Schema für erweiterte Bezeichner, die in einem Identitätsbezeichner gekapselt sind.
- Zur optimalen Nutzung der geringen Übertragungsmenge soll ein evtl. vorhandener Namensraumpräfix entfernt und der Namensraum als Standardnamensraum definiert werden.
- Normalisieren des XML Dokumentes nach den Kriterien der W3C. Dies bedeutet insbesondere leere Elemente und unnötige Leerzeichen zu entfernen sowie die Attribute alphabetisch aufsteigend zu sortieren.
- Die speziellen Steuerzeichen in einem XML Dokument müssen zur Übertragung innerhalb eines XML Dokumentes maskiert werden. Hierzu werden die relevanten Zeichen durch ihre XML Entitäten ersetzt.

Der beschriebene Mechanismus ist derzeit nur eine Übergangslösung, um die Abwärtskompatibilität zur IF-MAP 2.0 Spezifikation beizubehalten. Offensichtlich ist man aber gewillt in der Zukunft eine universellere, jedoch inkompatible Lösung zu etablieren.[3, S. 22] Dies zeigt auf, wie groß das Interesse an der Nutzung erweiterter Bezeichner ist.

3.2 Metadaten

Metadaten werden in XML Dokumenten übertragen. Dabei liegt ihnen ein XML Schema als Basis zugrunde, welches „$operational - metadata$“ in Form zwei spezieller Attribute verbindlich vorschreibt. Hierzu gehört der vom MAP Server zu setzende Zeitstempel „$ifmap - timestamp$“ sowie eine, ebenfalls durch den MAP Server bestimmte, eindeutige „$ifmap - publisher - id$“ für den jeweils verbundenen Client. Dazu können von einer Anwendung beliebig viele weitere

Attribute hinzugefügt werden. Das Präfix „$ifmap$“ darf jedoch bei weiteren Attributen, welche Nutzdaten enthalten, nicht verwendet werden. Die Verwendung ist für Definitionen, die sich aus der aktuellen wie auch zukünftigen Spezifikation ergeben, reserviert.

Listing 2: Das XML Schema als Basis für Metadaten

```
1 <xsd:attributeGroup name="metadataAttributes">
2    <xsd:attribute name="ifmap-publisher-id"/>
3    <xsd:attribute name="ifmap-timestamp" type="xsd:dateTime"/>
4    <xsd:anyAttribute/>
5 </xsd:attributeGroup>
```

In diesem Kapitel wird untersucht, welche notwendigen Anpassungen und Erweiterungen sich durch die IF-MAP 2.1 Spezifikation bei der Nutzung von Metadaten ergeben.

3.2.1 Normalisieren von Metadaten

Ein Client hat alle Metadaten von Quellen, die keine Unterscheidung in der Groß- und Kleinschreibung vornehmen, auf Kleinschreibung zu normalisieren.[3, S. 24] Als Quelle für Metadaten kommen prinzipiell beliebige Endgeräte in Frage. Ein Client kann ggf. auch Daten für mehrere Geräte veröffentlichen. Aus diesem Grund ist es nur möglich die Normalisierung vorzunehmen, sofern bekannt ist wie die jeweilige Quelle die Daten selbst behandelt.

3.2.2 Beziehung eines MAP Client zu einer Publisher ID

Die vom MAP Server vergebene Publisher ID soll für jeden Client eindeutig sein und den Client über mehrere Sitzungen hinweg identifizieren. Hierbei dürfen zur Erstellung der „$ifmap - publisher - id$“, je nach Authentifizierungstyp, nur bestimmte Daten des Clients Berücksichtigung finden. Die folgende Tabelle enthält eine Übersicht der empfohlenen Parameter zur Berechnung einer eindeutigen ID für einen Publisher bzw. MAP Client.[3, S. 25]

Basic Authentifizierung	Zertifikatsauthentifizierung
Benutzername	„Subject“-Feld des Clientzertifikats „Issuer“-Feld des ersten Zertifikats in der Kette

Neben den genannten Parametern soll auch die Authentifizierungsmethode und ggf. noch die IP Adresse Berücksichtigung finden, um eine theoretisch mögliche Verwechslung von Clients grundsätzlich auszuschließen. Außerdem können vom Server weitere beliebige Parameter herangezogen werden. Dabei ist jedoch zu berücksichtigen, dass keine geheimen Daten in die Berechnung einfließen. Es ist daher nicht erlaubt das Passwort oder den privaten Schlüssel einer Verbindung

zu verwenden. Dies soll sicherstellen, dass eine Änderung des Passwortes oder des Clientzertifikates keine Auswirkungen auf die Identifikation eines MAP Client haben.

Die Zuordnung eines Client zu seiner ID soll persistent gespeichert werden und bei Clients, die mehrere Verbindungen gleichzeitig über unterschiedliche Netze bzw. IPs öffnen, zusätzlich anhand der IP-Adresse unterschieden werden. Hierzu kann entweder eine Persistenzschicht im MAP Server genutzt oder aber die Berechnung einer ID nur anhand der genannten notwendigen Kriterien durchgeführt werden.

Bisher wurde beim MAP Server irond die Zuordnung anhand des Benutzernamens bzw. „Common Name“ aus dem „Subject DN“ des Zertifikates vorgenommen. Diese Zuordnung wird in einer lesbaren textbasierten Konfigurationsdatei persistent gespeichert. Dabei wurde bei Erstellung einer neuen ID diese noch mit einem Mikrosekundenzeitstempel gesalzen, was einen Rückschluss auf den zugrunde liegenden Benutzernamen oder „Common Name“ verhindern soll.

Allerdings ist es einfach den „Common Name“ eines selbst-signierten Zertifikates auf einen beliebigen Hostnamen auszustellen und so ein gefälschtes Zertifikat zu erzeugen. Aus diesem Grund soll nun zusätzlich bei der Zertifikatsauthentifizierung der Ausstellername (*engl.* issuer name) des ersten Zertifikates (dem sog. Root-CA) in der Kette zur Berechnung herangezogen werden.[3, S. 25]

Die Spezifikation schreibt inzwischen auch zwingend vor, dass die Berechnung der ID nicht zeitabhängig erfolgen darf. Die Grundlage zur Berechnung war bisher jedoch ein Zeitstempel. Daher sind Anpassungen notwendig, die das geforderte Verhalten vollständig umsetzen. Vgl. [4, S. 19]

3.2.3 Kardinalität von Metadaten

Wie bereits erwähnt, basieren alle Metadaten auf einem Basisschema, welches zwei Attribute bereits zwingend vorschreibt. Zusätzlich werden Metadaten noch in Gruppen eingeteilt. Jede Gruppe von Metadaten kann einen von zwei Kardinalitätswerten annehmen. Der Wert „$singleValueMetadataAttributes$“ bedeutet, dass diese Gruppe von Metadaten nur ein einziges Mal im Graphen veröffentlicht werden kann. Wobei hingegen der Wert „$multiValueMetadataAttributes$“ bedeutet, dass eine Gruppe von Metadaten mehrfach veröffentlicht werden darf. Ein MAP Server hat diese Daten dem Graphen hinzuzufügen, selbst wenn diese mit bereits bestehenden Metadaten identisch sind.

Dabei ist entscheidend, welcher Kardinalitätstyp bei der Aktualisierung von Metadaten des zugehörigen Bezeichners oder Link mit veröffentlicht wird. Die Kardinalität von bereits veröffentlichten Metadaten anderer Bezeichner soll fortan nicht mehr herangezogen werden. Bisher war es der Implementation in ihrem Verhalten diesbezüglich freigestellt.[3, S. 26]

3.2.4 Größe von Metadaten

Es müssen mindestens 100.000 Bytes (~98 KiBi) an Metadaten in einer Anfrage sowohl vom Client an den Server als auch umgekehrt übertragen werden können.[3, S. 27]

Es sind weder in der ifmapj-Bibliothek noch im irond MAP Server künstliche Beschränkungen zur Größe von Metadaten eingebaut. Somit ist die Größe lediglich vom verfügbaren Speicherplatz des MAP Servers bzw. Clients abhängig.

Da bei irond der gesamte Graph im Arbeitsspeicher gehalten wird, ist dieser die einzige Beschränkung. Der verfügbare Arbeitsspeicher bei heutigen Geräten, der selbst im Embedded-Bereich bereits bei mehreren Megabyte oder sogar Gigabyte liegt, ist somit kein nennenswertes Problem in diesem Zusammenhang.

3.2.5 Neue operative Metadaten

Zur Realisierung der Zeitsynchronisation (siehe 3.8 Zeitsynchronisation) wurde es notwendig einen neuen Typ von Metadaten einzuführen. Die operativen Metadaten (*engl.* operational metadata) haben ihren Namen aufgrund ihrer Verwendung erhalten, da sie eine grundlegende Funktion bei der Kommunikation mit einem MAP Servers einnehmen.[3, S. 27]

Hierfür wurde durch die TCG ein separates XML Schema mit einem eigenen Namensraum erstellt, welches derzeit nur ein einziges Attribut, die „$current-timestamp$“ beinhaltet.[3, S. 28]

Leider wurde in der Schema-Definition das Attribut fälschlich mit dem Namen „$current-time$“ beschrieben. Alle auf dem Schema basierenden Beispiele und die zugehörige Erläuterung bezeichnen das Attribut jedoch als „$current-timestamp$“. Aufgrund dessen wird davon ausgegangen, dass die durchgängig genutzte Attributsbezeichnung „$current-timestamp$“ korrekt ist.[3, S. 28/80]

Listing 3: Gekürzte Version des XML Schemas für operative Metadaten: "http://www.trustedcomputinggroup.org/2012/IFMAP-OPERATIONAL-METADATA/1"

```
1 <xsd:element name="client-time">
2    <xsd:complexType>
3       <xsd:attribute name="current-timestamp" type="xsd:dateTime" use=
            "required"/>
4       <xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
5    </xsd:complexType>
6 </xsd:element>
```

Die angepasste Version des Schemas mit dem Attribut „$current-timestamp$“ ist im Anhang unter A.6.4 zu finden.

3.2.6 Herstellerspezifische Metadaten

Ein Client, welcher herstellerspezifische Metadaten nicht auswerten kann, soll diese ignorieren, damit der normale Betrieb des Clients und die Kommunikation zum Server nicht beeinträchtigt wird.[3, S.14]

Die Umsetzung dieses Verhaltens obliegt rein der jeweiligen Clientimplementation und nicht der ifmapj-Bibliothek. Die Bibliothek wertet die Metadaten nicht aus und ist somit bereits in der Lage diese mit beliebigem Inhalt zu transportieren. Daher ist eine konkrete Umsetzung des geforderten Verhaltens hier nicht vorgesehen.

Bisher war es auch nicht vorgesehen herstellerspezifische Metadaten mit der ifmapj-Bibliothek zu erzeugen. Zur Erstellung herstellerspezifischer Metadaten bedarf es eines eigenen XML Namensraums, welcher für die in den Metadaten enthaltenen Elemente zuständig ist. Dazu muss die Möglichkeit geschaffen werden neben den eigentlichen Nutzdaten auch den Namensraum mit entsprechendem URI angeben zu können.

3.3 XML Schema-Validierung

Optional kann ein MAP Server eine XML Schema-Validierung der an ihn übertragenen XML Dokumente vornehmen. Dies beinhaltet die gesamte Kommunikation über XML Nachrichten zwischen MAP Client und MAP Server inklusive der variablen Nutzdaten der Übertragung: Metadaten und erweiterte Bezeichner. Sofern der Server bei der Validierung einen Fehler findet, soll er die übertragenen Daten nicht speichern sowie die Anfrage des Clients ablehnen und mit einer entsprechenden Fehlermeldung reagieren.[3, S. 29]

Grundsätzlich ist ein MAP Server angewiesen, alle Metadaten zu akzeptieren, die wohlgeformtem (gültigem Syntax) XML entsprechen. Als logische Konsequenz muss der Server auch herstellerspezifische Metadaten akzeptieren, für die er kein Schema vorliegen hat. Optional kann der Server jedoch in einem abgeschotteten Modus betrieben werden, in dem er alle Metadaten abweist für die keine entsprechenden Schemata vorliegen.[3, S. 30]

Zur Übermittlung ob und in welchem Umfang das übertragene XML validiert wurde, kann vom MAP Server ein vordefinierter Wert aus der Attributgruppe „$validationAttributes$“ in seinen Antworten an MAP Clients gesetzt werden:

Attribut	Bedeutung
$None$	Die übertragenen XML Daten wurden nicht validiert.
$BaseOnly$	Die XML Kommunikation ohne die ggf. enthaltenen Metadaten wurde erfolgreich validiert.
$MetadataOnly$	Nur die enthaltenen Metadaten wurden validiert.
All	Alle übertragenen Daten wurden erfolgreich validiert.

Auch ein MAP Client darf eine Schema-Validierung durchführen und die gleiche Attributgruppe setzen, um dieses dem Server mitzuteilen. Trotzdem darf sich der Server nicht auf den vom Client übertragenen Validierungsstatus verlassen und ist angehalten die Validierung nötigenfalls selber durchzuführen.

3.4 Fehlermeldungen

Die von der TCG definierten Fehlermeldungen, die ein MAP Server ggf. auf Anfragen eines Client zurückgibt, bestehen unverändert. Lediglich die Definition der `AccessDenied` Meldung ist näher spezifiziert worden.[3, S. 30ff]

Bisher wurde in der Spezifikation auf die Behandlung von Suchanfragen im Zusammenhang mit Berechtigungen nicht eingegangen. Von jetzt an ist klar festgelegt, dass ein Server Antworten auf Suchanfragen unter Umständen zu zensieren hat.[3, S. 31]

Dabei sollen nur die Informationen an den Client übermittelt werden, für die der Client die erforderlichen Berechtigungen besitzt. Alle übrigen Suchergebnisse sollen stillschweigend unterdrückt werden. Weitere Details zu den Änderungen am Suchalgorithmus sind unter Abschnitt 3.6 zu finden.

Mögliche Anpassungen betreffen hier nur den MAP Server irond. Dabei sind umfangreiche Anpassungen vorzunehmen, die ein Berechtigungssystem grundsätzlich erst einmal etablieren würden. Hierbei sind verschiedene Ansätze denkbar. Am sinnvollsten erscheint die Nutzung eines auf Rollen (oder Gruppen) basierenden Systems, um für gleichartige Clients die Vergabe von Berechtigungen einfacher gestalten zu können.

Die Behandlung der Implementation eines solchen Systems ist sehr umfangreich und im Rahmen dieser Arbeit nicht realisierbar bzw. ist als gesondertes Thema zu betrachten. Ein Prototyp zur Abbildung der notwendigen Funktionalitäten wird derzeit bereits durch die Trust@FHH Forschungsgruppe entwickelt.

3.5 Nutzung von „match-links“

Zur Filterung der Ergebnisse, die ein MAP Server auf eine Suchanfrage hin zurückgibt, kann ein Client das optionale Attribut „$match-links$“ verwenden und darin angeben, welche Daten der Server ausgeben soll. Dann wird vom MAP Server erwartet, dass er alle Suchergebnisse ungefiltert ausgibt. Zusätzlich hierzu soll es jetzt noch möglich sein eine leere Zeichenkette als Attributwert zu übergeben (vgl. [4, S. 29]). Dabei wird vom Server erwartet ein leeres Suchergebnis auszuliefern. [3, S. 36f]

Die Umsetzung eines Suchfilters sowohl für den Filtertypen „$result-filter$“ sowie „$match-links$“ wurde bereits so umgesetzt, dass das geforderte Verhalten auftritt. Dazu sind bereits in die Klasse `Filter` die Funktionen `isMatchNothing()` und `isMatchEverything()` implementiert, die genau diese Bedingungen prüfen.

3.6 Bedingungen für den Suchalgorithmus

Am Suchalgorithmus gibt es keine grundlegenden Änderungen. Nur in zwei Punkten wurde genauer festgelegt, wie entsprechend zu verfahren ist.

Es wird nun konkret darauf hingewiesen, sofern die Attribute „$result - filter$“ oder „$match-links$“ nicht vorhanden sind, das die Ergebnisse auch nicht gefiltert werden dürfen.

Als logische Schlussfolgerung war dies schon selbstverständlich, jedoch im Falle des „$match-links$“ Attributs wurde nun eine Besonderheit eingeführt. Sofern der Attributwert einer leeren Zeichenkette entspricht, sollen alle Suchergebnisse gefiltert werden. In Konsequenz daraus erhält der Client nun ein leeres Suchergebnis unabhängig von der Anzahl tatsächlicher Treffer.

Zu einer Unterscheidung in ähnlicher Form beim Filtertypen „$result - filter$“ ist das geforderte Verhalten für beide Filtertypen bereits umgesetzt worden.

3.7 Behandlung von Sitzungen

Sofern ein Client mehrfach eine Verbindung zu einem MAP Server herstellt, ist jeweils nur die neueste Verbindung zu verwenden. Alle bisher bestehenden Verbindungen sind zu schließen oder zumindest darf über diese nicht mehr auf Anfragen positiv reagiert werden. Die einzige Antwort, die der Server in solch einem Falle zurückgeben darf, ist die Fehlermeldung $AccessDenied$.

Bisher wurde empfohlen eine Verbindung nach mindestens drei Minuten Inaktivität vom Server trennen zu lassen. Zusätzlich soll jetzt der in RFC 1122[29] spezifizierte Mechanismus „TCP Keep-Alive“ genutzt werden.[3, S. 49] Dabei wird, sofern eine Verbindung mindestens zwei Stunden inaktiv ist, ein TCP-Paket an den Client gesendet, auf welches dieser antwortet und somit signalisiert, dass die Verbindung noch besteht.

3.8 Zeitsynchronisation

Unter bestimmten Bedingungen kann es notwendig sein, die Uhren zwischen Server und Client abzugleichen. So kann beispielsweise ein PDP, welcher auf Ereignisse reagieren soll, diese unter Umständen aus Sicherheitsgründen verwerfen, da ein zu großer zeitlicher Versatz besteht. Es können aber auch falsche Entscheidungen auf Grund von Richtlinien zur zeitlichen Zugangsbeschränkung getroffen werden.

Eine korrekte Clientimplementation darf sich nicht auf die lokale Systemzeit verlassen, sondern muss seine Uhr mit dem MAP Server abgleichen. Dieser wiederum sollte seine Zeit auch von verlässlichen externen Quellen beziehen. Hier bietet sich die Nutzung des NTP[30] Protokolls an, welches weit verbreitet ist und bereits in

[29] http://tools.ietf.org/html/rfc1122#page-101

[30] http://en.wikipedia.org/wiki/Network_Time_Protocol

etlichen Umgebungen zuverlässig eingesetzt wird.

Zur Lösung dieses Problems schreibt die IF-MAP Spezifikation vor, dass die vom Client gesendeten Metadaten, welche Zeitstempel enthalten, diese vor der Publikation auf dem Server an die lokale Zeit des Servers anpassen. Daher ist unter den folgenden Bedingungen eine Anpassung an die Serverzeit notwendig:[3, S.54]

- Ein Client veröffentlicht Metadaten, welche Zeitstempel beinhalten.
- Ein Client empfängt Metadaten, welche Zeitstempel enthalten.
- Ein Client sucht oder abonniert Metadaten mit einem Zeitstempelfilter.

Es wird empfohlen einen automatischen Mechanismus in den Client zum Zeitabgleich zu implementieren, um die Abweichung zum Server zu kompensieren. Dabei muss der Client seine Zeitstempel an die lokale Zeit des Servers anpassen.[3, S.54]

3.9 Zusammenfassung

Zum Teil sind rein „kosmetische“ Änderungen in der Spezifikation vorgenommen worden, bei denen keine weitere Betrachtung notwendig ist. Neben den Angaben zur Größe von Metadaten und Bezeichnern, der Ausgabe von Fehlermeldungen und den Bedingungen für den Suchalgorithmus, ist insbesondere die Verwendung der administrativen Domäne bei Bezeichnern lediglich näher erläutert worden:

Bezeichner	Verwendung
$access-request$	Die Angabe der administrativen Domäne entfällt.
$ip-address$	Die adm. Domäne soll der Routing Domäne entsprechen
$mac-address$	Angabe der authentifizierenden Stelle als adm. Domäne.
$device$	„$aik-name$“ entfällt während „$name$“ weiter besteht.

Der Vorgehensweise zum Vergleichen von Identitätsbezeichnern wurde näher dargelegt. Einige Identitätstypen werden vor einem Vergleich normalisiert indem ihre Attributswerte in Kleinbuchstaben umgewandelt werden. Die Typen „$extended$“ und „$distinguished-name$“ müssen allerdings speziell behandelt werden. Dazu wurden die Vorgaben in den Abschnitten 3.1.4 und 3.1.5 detailliert untersucht.

Im Kapitel 3.2 sind die Einzelheiten zu allen Änderungen an der Verwendung von Metadaten analysiert worden. So müssen vom Client Metadaten ggf. normalisiert, die Generierung der Publisher ID geändert und eine XML Schema-Validierung durchgeführt werden. Außerdem sollen neue operative Metadaten eingeführt werden, die derzeit nur bei der Zeitsynchronisation Anwendung finden. Weitere konzeptionelle Überlegungen müssen noch zur Umsetzung der geforderten Kardinalitätsbedingungen sowie zur Normalisierung von Metadaten angestellt werden.

Zusätzlich soll in der ifmapj-Bibliothek die Erstellung herstellerspezifischer Metadaten ermöglicht und der MAP Server irond um eine Konfigurationseinstellung

zur Verwendung von TCP Keep-Alive erweitert werden.

Die Analyse ergab, dass die folgenden Punkte konzeptionell weiter bearbeitet und voraussichtlich implementiert werden müssen:

Funktion	notwendige Anpassung(en)
Administrative Domäne	Funktionalität den Vorgaben bei Bezeichnern anpassen
Bezeichner	Funktion zur Namenserzeugung bei „$device$“ und „$access - request$“ Bezeichnern umsetzen.
Gerätebezeichner	Die Änderungen der Vergleichsoperation beim irond übernehmen.
Identitätsbezeichner	Notwendige Normalisierung bei einigen Identitätstypen umsetzen.
Erweiterte Bezeichner	Vollständige Umsetzung zur Nutzung der neuen erweiterten Bezeichner.
Publisher ID	Überarbeitung der Generierung einer Publisher ID im irond.
Metadaten	Funktionalität schaffen zur Erstellung herstellerspezifischer und operativer Metadaten.
Schema-Validierung	Vollständige Umsetzung in irond notwendig.
Sitzungen	Nutzung der TCP Keep-Alive Option in irond.
Zeitsynchronisation	Vollständige Umsetzung in ifmapj notwendig.

Eine vollständige Übersicht aller Anpassungen und Erweiterungen inkl. der zugehörigen Abschnitte aus der IF-MAP 2.1 Spezifikation sind im Anhang unter A.8 zu finden.

4 Konzept zur Umsetzung

In diesem Kapitel werden die zur Implementierung notwendigen konzeptionellen Überlegungen und notwendigen Ansätze dargestellt, die sich aus der überarbeiteten IF-MAP 2.1 Spezifikation vom Mai 2012 ergeben.

4.1 Bezeichner

Bezeichner dienen zur Verknüpfung von Metadaten. So kann beispielsweise ein Bezeichner ein Netzwerk darstellen und weitere Bezeichner die Geräte innerhalb des Netzwerkes. Eine Verknüpfung der verschiedenen Bezeichner wird über zusätzliche Metadaten hergestellt. In der Praxis kann so ein DHCP Server, der für ein Subnetz verantwortlich ist, dieses als Bezeichner im MAP Graphen veröffentlichen. Jedes Gerät, dem vom DHCP Server eine IP Adresse zugewiesen wurde, kann nun über entsprechende Metadaten in Zusammenhang mit dem Subnetz dargestellt werden.

Es gibt eine Reihe vorgegebener Bezeichner, die vollständig von jeder IF-MAP Implementation umgesetzt werden müssen. In diesem Abschnitt werden konzeptionelle Überlegungen im Zusammenhang mit den Änderungen an Bezeichnerobjekten vorgenommen.

4.1.1 Bedingungen zur Nutzung einer administrativen Domäne

Bisher wurde es gestattet für einen „$access - request$“ Bezeichner eine administrative Domäne zu setzen. Diese Angabe ist nicht länger erlaubt und sollte in der ifmapj-Bibliothek als veraltet markiert werden. Es ist nicht sinnvoll die Angabe vollständig aus dem Funktionsaufruf zu entfernen, da dies Probleme mit Implementationen geben kann, die noch auf einer früheren Version der IF-MAP Spezifikation basieren. Um die Abwärtskompatibilität zur bisherigen Umsetzung zu erhalten, muss es zusätzlich möglich sein einen „$access - request$“ Bezeichner ohne die Angabe einer administrativen Domäne zu erstellen.

Die in der Analyse betrachteten und empfohlenen Anwendungsszenarien zur Nutzung der administrativen Domäne sind abhängig von der tatsächlichen Anwendung und daher nicht weiter Gegenstand der Untersuchung.

4.1.2 Änderungen am Gerätebezeichner

Es wird vorgesehen ein Gerät mittels eines „$device$“ Bezeichners im Graphen darzustellen. Hierbei war es bisher möglich entweder einen „$name$“ oder einen „$aik - name$“ zu setzen. Der Name eines Attestation Identity Key (aik-name) ist nicht zwingend eindeutig und daher zur Identifikation eines Gerätes nicht geeignet.

Aus diesen Grund ist die Verwendung des Attributs „$aik - name$“ zur Identifikation eines Gerätes nicht länger sinnvoll. Daher ist vorgesehen, dass ein Client bei

der Erstellung eines Gerätebezeichners keinen „$aik - name$“ mehr angeben darf. Zur Wahrung der Abwärtskompatibilität sollte der entsprechende Funktionsaufruf in der ifmapj-Bibliothek als veraltet markiert und nicht entfernt werden.

Ein MAP Server muss hingegen, zur Wahrung der Abwärtskompatibilität, weiterhin beide Angaben verarbeiten. Das bisherige Verhalten des irond war daher nur den Namen zu vergleichen und die Typen nicht zu berücksichtigen. Dies muss jetzt entsprechend angepasst werden, so dass Name und Typ bei einem Vergleich zusammen geprüft und nur bei einer vollständigen Übereinstimmung die Bezeichner als gleich angesehen werden.

4.1.3 Regeln zum Vergleich von Identitätsbezeichnern

Wie in der Analyse bereits festgestellt, müssen einige Identitätstypen vom Client vor der Übertragung an den Server normalisiert werden. Zur Minimierung des Implementationsaufwandes eines IF-MAP Clients liegt es nahe die notwendige Normalisierung weitestgehend in die ifmapj-Bibliothek zu integrieren. Für die in der Analyse bereits genannten Identitätstypen müssen die Namen ganz oder teilweise (Domain einer E-Mail) in Kleinschreibung überführt werden.

Im Falle der „$distinguished-name$“ Identität muss die Behandlung jedoch nach speziellen Kriterien erfolgen. Hierauf wird im nächsten Abschnitt näher eingegangen.

Zur einfachen Handhabung der unterschiedlichen Weisen zur Normalisierung sollte die notwendige Funktionalität in der ifmapj-Bibliothek für den Client transparent durchgeführt werden.

4.1.4 Gleichheitsbedingungen von „distinguished-name“ Identitäten

Die Java SE Standardklassenbibliothek bietet bereits seit Version 1.4 in der Klasse `javax.security.auth.x500.X500Principal`[31] eine einfache Möglichkeit zur Erstellung und Nutzung von eindeutigen X.500 Namen.

Die Klasse akzeptiert die beiden per RFC 1779[32] und RFC 2553 definierten Formate für eindeutige X.500 Namen und implementiert darüber hinaus die Vergleichsmethode `equals()`, die den Anforderungen in RFC 2459[33] entspricht.

Benötigt wird diese Funktionalität im MAP Server irond zur Erkennung gleicher Identitäten, aber auch die ifmapj-Bibliothek sollte diese Funktion bereitstellen.

[31] `http://docs.oracle.com/javase/1.4.2/docs/api/javax/security/auth/x500/X500Principal.html`

[32] `http://www.ietf.org/rfc/rfc1779.txt`

[33] `http://www.ietf.org/rfc/rfc2459.txt`

4.1.5 Erweiterte Bezeichner

Ein maßgeblicher Grundgedanke zur Umsetzung ist, bestehende Funktionalitäten zur Verarbeitung von XML Dokumenten zu nutzen. Hierbei ist eine vollständige und standardkonforme Unterstützung anzunehmen, da Java bereits in vielen Anwendungsbereichen zur Verarbeitung von XML Dokumenten genutzt wird.

So soll auch zur Umsetzung des in der Spezifikation beschriebenen Normalisierungsprozesses beim Schritt zur Entfernung eines ggf. vorhandenen Namensraumspräfixes auf die Technik der XSL Transformationen zurückgegriffen werden, anstatt eine einfache Ersetzung von Zeichenketten nach einem festgelegten Erkennungsmuster durchzuführen.

Bei der Untersuchung, ob eine XSL Transformation mit Java möglich ist und ob diese das erwartete Ergebnis liefert, kam eine Eigenheit von Java zu Tage, die in anderen Programmiersprachen und Umgebungen nicht auftritt. Dabei wurde, anstatt einen Namensraumpräfix zu entfernen, dieser durch einen Standardpräfix ersetzt. Um dieses Verhalten genauer zu untersuchen, wurde die gleiche Transformation einmal in PHP[34] unter Verwendung der libxslt-Bibliothek[35] als auch in der XML-Entwicklungsumgebung Altova XMLspy[36] durchgeführt. Beide verhielten sich, entgegen Java, wie erwartet und haben keinen eigenen Präfix erstellt.

Zum Vergleich des Verhaltens unter Java mit anderen Anwendungen kann die in der Datei `removeNamespaces.xslt` (siehe A.6.6) enthaltene XSL Transformation zur Entfernung eins Namensraumspräfix mit der in den ifmapj-examples enthaltenen XML Datei `example-ei.xml` (siehe A.6.1) für einen erweiterten Bezeichner manuell durchgeführt werden.

Listing 4: Beispiel zur XSL Transformation per Java `TransformerFactory`

```
1 try {
2    Document doc = DocumentBuilderFactory.newInstance().
         newDocumentBuilder().parse(new File("example-ei.xml"));
3    Transformer tf = TransformerFactory.newInstance().newTransformer(
         new StreamSource(new File("removeNamespaces.xslt")));
4    StringWriter buffer = new StringWriter();
5    tf.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
6    tf.transform(new DOMSource(doc.getFirstChild()), new StreamResult(
         buffer));
7    System.out.println(buffer.toString());
8 } catch (TransformerException e) { }
```

Zur Umgehung dieses Java-spezifischen Problems wurden mehrere Ansätze zur XSL Transformation ausprobiert. Alle lieferten unterschiedliche Ergebnisse, aber keiner das Gewünschte. Daher wird es nicht vermeidbar sein, nach der Transforma-

[34] http://www.php.net/manual/de/book.xsl.php
[35] http://xmlsoft.org/XSLT/
[36] http://www.altova.com/de/xmlspy.html

tion doch noch eine Zeichenersetzung zur Entfernung des Präfix durchzuführen. Hierbei kann auf reguläre Ausdrücke zurückgegriffen werden, die eine solche Ersetzung ohne großen Aufwand ermöglichen.

4.2 Metadaten

In diesem Abschnitt werden konzeptionelle Überlegungen und Schlussfolgerungen zur Anwendung von Metadaten im Zusammenhang mit der IF-MAP Spezifikation in der Version 2.1 sowie den dazugehörigen Rahmenbedingungen angestellt.

4.2.1 Normalisieren von Metadaten

Die empfohlene Normalisierung für Metadaten umfasst sowohl die an einen MAP Server gesendeten Metadaten als auch die von einem Server empfangenen. Hierzu müssen ggf. die übertragenen Metadaten in Kleinschreibung umgewandelt werden.

Eine Funktion zur Umwandlung in Kleinschreibung ist in Java bereits vorhanden. Der Aufruf dieser Funktion ist genauso einfach wie ein zusätzliches Attribut als Indikator zur Umwandlung in Kleinbuchstaben beim Funktionsaufruf zu setzen. Daher wird hier in der Implementation darauf verzichtet dieses zusätzliche Attribut einzuführen und Funktionen damit unnötig weiter zu überladen.

4.2.2 Erstellung der Publisher ID

Das Format einer Publisher ID ist nicht definiert und daher wird hier ein eigenes Konzept zu dessen Erstellung entwickelt. Die Berechnung einer eindeutigen Publisher ID soll, je nach Authentifizierungsart, jeweils anhand der in der Analyse festgelegten Parameter erfolgen. Die entsprechenden Parameter können zur weiteren Verwendung in eine möglichst eindeutige Kennung mit fester Länge konvertiert werden. Hierzu werden diese als Zeichenkette zusammengefügt und anschließend wird ihre MD5[37] Prüfsumme berechnet. Dieser wird der Benutzername als Präfix vorangestellt und zusammen ergibt das dann eine unverwechselbar eindeutige Publisher ID.

Beispiel einer Publisher ID für den Benutzer „publisher“:

```
publisher-d41d8cd98f00b204e9800998ecf8427e
```

Das bisherige Konzept, welches noch einen Zähler an die Publisher ID angefügt hat, wird nicht weiter benötigt, da der gleiche Client immer dieselbe Identifikation haben soll, sofern er nicht mehrere IP Adressen nutzt. In diesem Fall und nur dann soll die IP Adresse des Clients ebenfalls in der Berechnung der ID Berücksichtigung finden. Die notwendigen Parameter und ggf. die IP Adresse werden in der Klasse `ClientIdentifier` festgelegt und zur Laufzeit im Arbeitsspeicher vorgehalten.

[37] `http://de.wikipedia.org/wiki/Message-Digest_Algorithm_5`

Sofern sich die zu berücksichtigenden Parameter für einen Client nicht ändern, ist sichergestellt, dass ein Client dauerhaft die gleiche Identifikation erhält. Daher wäre eine persistente Speicherung der Publisher ID nicht notwendig. Diese ist jedoch bereits implementiert und kann somit einfach weiter genutzt werden. So ist außerdem die Möglichkeit gegeben, sofern sich der Benutzername eines Clients ändert, die Zuordnung manuell anzupassen um die gleiche ID weiter zu verwenden.

4.2.3 Kardinalität von Metadaten

Die Prüfung der bestehenden Implementation des MAP Servers irond zur Umsetzung des geforderten Verhaltens hat ergeben, dass dieses bereits in abgewandelter Form implementiert wurde und es empfehlenswert ist, daran keine Änderungen vorzunehmen.

Bei der Veröffentlichung von Metadaten gleichen Typs mit unterschiedlicher Kardinalität soll eine entsprechende Fehlermeldung vom Server an den Client ausgegeben und die Metadaten nicht in den Graphen aufgenommen werden. Hierbei ist es unerheblich, ob diese Metadaten überhaupt in einem Zusammenhang zueinander stehen. Sobald Metadaten eines bestimmten Typs einmal auf dem Server veröffentlicht wurden, gilt die dort angegebene Kardinalität für alle zukünftigen Veröffentlichungen des gleichen Typs.

Sofern eine XML Schema Validierung von Metadaten erfolgt (siehe 4.3), greift die im Schema definierte Kardinalität und die Entscheidung obliegt nicht den erstmalig veröffentlichten Metadaten. Eine korrekte Unterscheidung ist ohne Schema-Validierung nicht möglich. Daher bleibt als einzige Möglichkeit, sich auf einen der beiden möglichen Werte festzulegen. Es liegt nahe, den Wert der zuerst veröffentlichten Metadaten als korrekt anzunehmen.

In der Spezifikation wird jedoch darauf hingewiesen, dass die Kardinalität nur von Metadaten mit gleicher Zuordnung zu Bezeichnern berücksichtigt werden soll. Dies steht im Widerspruch zur Definition des Kardinalitätsattributs im Basisschema. Es besteht keine Möglichkeit ein Element mit beiden Kardinalitätsangaben zuzulassen. Daher ist es nur sinnvoll, das erzwungene Verhalten auch im MAP Server strikt beizubehalten und nicht von den eingeschränkten Möglichkeiten, die sich aus der Schema-Definition ergeben, abzuweichen.

4.2.4 Größe von Metadaten

In diesem Fall ergab die Prüfung, dass die geforderten Bedingungen an die Mindestgröße von Metadaten bereits vollständig erfüllt sind. Hierzu wurde der Quellcode des MAP Servers irond untersucht, insbesondere die Funktionen zum „Marshalling“ und die korrespondierenden Java-Objekte. An keiner Stelle wurde eine ungewollte Beschränkung der Größe festgestellt.

4.2.5 Neue operative Metadaten

Es ist notwendig, zur Berücksichtigung des eigenen XML Namensraums für operative Metadaten, eine neue Funktion in der ifmapj-Bibliothek zu schaffen, die ein einfaches XML Element im entsprechenden Namensraum erstellt.

Dem MAP Server irond muss das Schema für den Namensraum bekannt gemacht werden, damit dieser es zur XML Validierung von Metadaten verwenden kann. Näheres dazu im Abschnitt 4.3. Das hierzu von der TCG festgelegte XML Schema mit dem angepassten „$current - timestamp$“ Attribut befindet sich im Anhang unter Abschnitt A.6.4.

4.2.6 Herstellerspezifische Metadaten

Zur einfachen Erstellung herstellerspezifischer (*engl.* vendor-specific) Metadaten ist bisher in der ifmapj-Bibliothek keine Funktion vorgesehen. Die Nutzung erweiterter Bezeichner bringt jedoch fast schon zwangsläufig die Notwendigkeit mit sich entsprechend angepasste Metadaten zu verwenden.

Es soll die Möglichkeit geschaffen werden auf einfache Weise Metadaten mit einem oder mehreren Attributen zu erstellen. Hierbei muss es die Möglichkeit geben den Zielnamensraum eines XML Schemas mittels URI anzugeben. Die optionale XML Schema Validierung der herstellerspezifischen Metadaten wird im nächsten Punkt behandelt.

4.3 XML Schema-Validierung

Zur Validierung der übertragenen Metadaten und erweiterten Bezeichner benötigt der MAP Server die zugehörigen XML Schemata. Diese sollen im Dateisystem unterhalb des Installationsordners im Verzeichnis `schema` abgelegt werden. In diesem Verzeichnis befindet sich bereits das zur Validierung von XML Nachrichten zwischen Client und Server notwendige Basisschema `ifmap-base-2.0v17.xsd`.

Hierbei werden die Basisschemata für erweiterte Bezeichner und für sämtliche Metadaten benötigt. Für alle von der TCG spezifizierten Metadaten ist das Schema `ifmap-metadata-2.0v8.xsd` zuständig. Die neu eingeführten operativen Metadaten besitzen einen eigenen Namensraum und sind aus diesem Grund in der Datei `ifmap-opmeta-2.1v1.xsd` zu finden.

Um alle Namensräume und die entsprechenden Schemadateien dem Server bekannt zu machen, bedarf es einer Erweiterung der `ifmap.properties` Konfigurationsdatei. Hier muss festgelegt werden können, ob Metadaten schema-validiert werden und welche Schemata hierbei zu verwenden sind. Zusätzlich soll der Server in einem abgeschotteten (*engl.* locked-down) Modus betrieben werden können bei dem Metadaten abgelehnt werden, für die kein XML Schema bekannt ist. Auch dies muss in der Konfiguration einstellbar sein.

Nach dem gleichen Prinzip soll eine Möglichkeit zur Angabe von Namensräumen und Schemata für erweiterte Bezeichner geschaffen werden. Auch hier soll es konfigurierbar sein, ob eine Schema-Validierung durchgeführt wird. Die Überlegungen zu einem abgeschotteten Modus für Metadaten sind auch anwendbar auf erweiterte Bezeichner. Hier kann auf die gleiche Weise der Bezeichner vom Server abgelehnt werden, sofern kein Schema zur Validierung vorliegt.

Im Gegensatz zum MAP Server irond, der durch das Prinzip per JAXB-RI XML Nachrichten zu erstellen nur valides XML erzeugt, ist das hingegen in der ifmapj-Bibliothek abhängig von der korrekten und vollständigen Implementation des Clients sowie der Bibliothek selbst. Es wären hier große Teile der Kommunikationsschicht in der Bibliothek neu zu entwickeln. Ein Bruch mit der Abwärtskompatibilität wäre zwar vermeidbar, würde aber etliche Funktionen mit einem Validierungsparameter überladen. Gleichzeitig würde die Bibliothek einen Mechanismus zur Bekanntmachung von XML Schemata benötigen. Somit würde man hier jedoch einiges an Leichtgewichtigkeit einbüßen oder umfangreiche Teile, wie die Verwaltung der bekannten Schemata, dem Client überlassen müssen. Darüber hinaus bringt die Validierung auch keinen nennenswerten Vorteil, da das Ergebnis der Validierung vom Server nicht verwendet werden darf.

4.4 Behandlung von Sitzungen

Zur Kommunikation zwischen Client und Server werden zwei Kanäle verwendet, der SSRC (*engl.* Synchronous Send and Receive Channel) und optional der ARC (*engl.* Asynchronous Receive Channel). Über den SSRC werden direkte Anfragen vom Client an den Server abgewickelt, wobei hingegen der ARC nur zur Übermittlung von Daten im Polling-Verfahren an den Client dient.

Die Möglichkeit zur Nutzung der TCP Keep-Alive[38] Option sollte in den Einstellungen des MAP Servers konfigurierbar sein. Zusätzlich sollte ein Zeitüberschreitungswert für den Abbruch blockierender Netzwerkanfragen innerhalb des TCP Stapels festlegbar sein, um den Prozess durch fehlerhafte Übertragungen nicht zu lange auszusetzen. Dies wird in der Spezifikation nicht explizit erwähnt, bringt jedoch den Vorteil, dass eine Anfrage maximal nur für den festgelegten Zeitraum die Kommunikation mit dem Client behindern kann. Sollte die Bearbeitung den festgelegten Zeitraum überschreiten, wird eine `InterruptedIOException` ausgelöst. Die Verbindung bleibt jedoch bestehen und kann anschließend weiter verwendet werden.

Beide Optionen sollen über die Konfigurationsdatei `ifmap.properties` einstellbar sein und sinnvolle Standardwerte definiert haben. Als sinnvoll wird erachtet, was die TCP Implementation Javas vorgibt. Somit ist TCP Keep-Alive standardmäßig ausgeschaltet und der Zeitraum für blockierende Anfragen unendlich.

[38] `http://tldp.org/HOWTO/TCP-Keepalive-HOWTO/overview.html`

4.5 Zeitsynchronisation

Es wird empfohlen einen automatischen Mechanismus zum Abgleichen der Client- und Serverzeit zu implementieren. Hierbei muss die Abwärtskompatibilität zur bestehenden Version 0.1.4 der ifmapj-Bibliothek gewährleistet bleiben. Daher muss der Referenzmechanismus zur Zeitsynchronisation optional in die Bibliothek implementiert werden.

Der automatische Abgleich erfolgt in drei Schritten, welche in Abbildung 3 dargestellt sind. Hierbei veröffentlicht der Client nach dem Verbindungsaufbau seine lokale Zeit auf dem MAP Server. Anschließend stellt er eine Suchanfrage nach der soeben veröffentlichten Zeit. In der Antwort auf die Suchanfrage befinden sich nun zwei Zeitangaben. Hierbei handelt es sich um die zuvor vom Client veröffentlichte Zeit, wie auch ein zusätzlich vom Server für alle Metadaten eingefügter Zeitstempel. Anhand der Differenz beider Uhrzeiten kann nun der zeitliche Versatz zwischen Client und Server ermittelt werden.

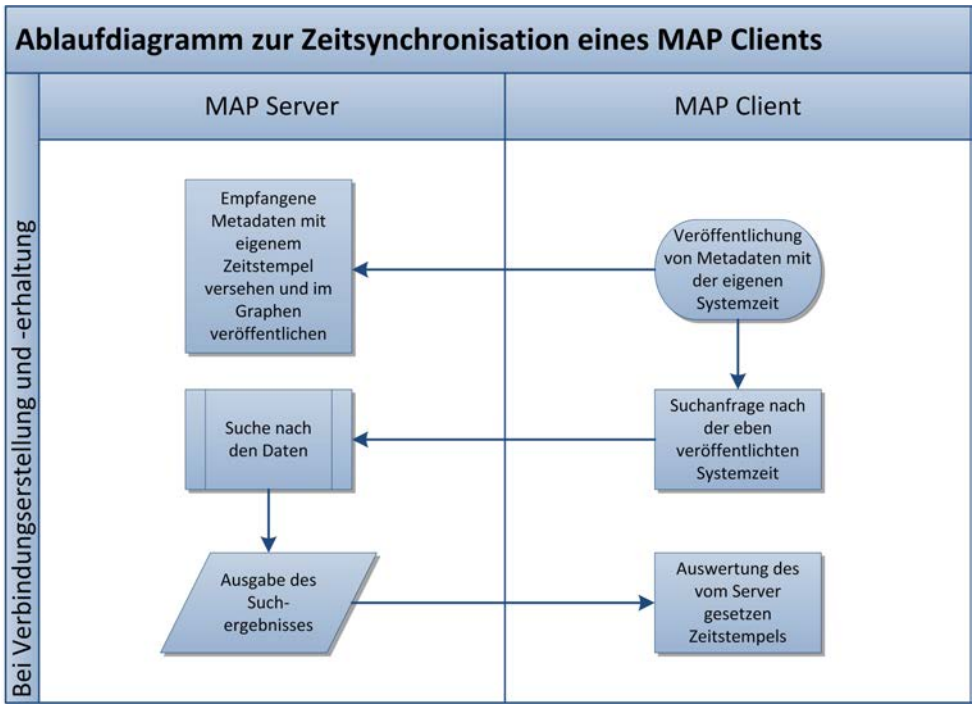


Abbildung 3: Schematischer Ablauf des automatischen Mechanismus zum Zeitabgleich eines MAP Client durch Veröffentlichung von Metadaten

Um die notwendigen Schritte der Referenzimplementation in der Bibliothek zu kapseln, wird der Konstruktor der Klasse `SSRCImpl` mit einer weiteren Möglichkeit zum Aufruf unter Angabe eines Gerätebezeichners überladen.

Bisher schon obliegt es dem Client eine Sitzung in regelmäßigen Abständen zu

erneuern. Sofern die Klasse unter Angabe eines Gerätebezeichners instanziiert wurde, wird bei jedem Aufruf von `renewSession()` auch ein erneuter Abgleich mit der Serverzeit erfolgen.

Damit eine einfache Erstellung von, auf die Serverzeit abgestimmten, Zeitstempeln für den Client ermöglicht wird, soll eine Funktion zur Rückgabe der aktuellen Serverzeit zur Verfügung gestellt werden.

4.6 Zusammenfassung

Die konzeptionellen Überlegungen haben ergeben, dass bei der Umsetzung einzelne nicht mehr erlaubte bzw. von jetzt an unnötige Funktionsaufrufe zur Wahrung der Abwärtskompatibilität erhalten bleiben müssen. So muss es weiterhin möglich sein in der ifmapj-Bibliothek, einen „$access - request$“ Bezeichner unter Angabe einer administrativen Domäne zu erstellen. Ebenso muss auch das Attribut „$aik - name$“ bei Gerätebezeichnern weiter verwendet werden können. Auch der MAP Server irond muss das Attribut weiterhin verarbeiten können und zwischen den Attributen „$aik - name$“ und „$name$“ bei Vergleichsoperationen unterscheiden.

Es wurden weitere Bedingungen zur Vergleichbarkeit von Identitätsbezeichnern formuliert. Diese sollen transparent von der eigentlichen Anwendung durchgeführt werden. Insbesondere die eindeutigen X.500 Namen sind dabei speziell zu behandeln und sollen daher in eine Instanz der Klasse `X500Principal` überführt werden, da so eine bereits bestehende RFC-konforme Vergleichsfunktion aus der Klasse genutzt werden kann.

Neu hinzugekommen sind die erweiterten Bezeichner, die zur Wahrung der Abwärtskompatibilität in einem Identitätsbezeichner übertragen werden. Dabei soll bei den XML Dokumenten vor ihrer Übertragung als erweiterter Bezeichner ein ggf. vorhandener Namensraumpräfix entfernt werden. Die neuen Vorgaben zur Berechnung einer eindeutigen Publisher ID erfordern eine neue Klasse, die die entsprechenden Anforderungen, unter Berücksichtigung der Authentifizierungsmethode und weiterer Parameter, umsetzen soll.

Eine Funktion zur Normalisierung von Metadaten soll nicht implementiert werden, da diese bereits auf simple Weise normalisiert werden können. Das Verhalten bei der Veröffentlichung von Metadaten gleichen Typs mit unterschiedlicher Kardinalität soll beibehalten werden um eine stringente Schema-Validierung zu ermöglichen.

Größenbeschränkungen werden weder in der ifmapj-Bibliothek noch im MAP Server irond fest vorgegeben. Daher besteht auch hier kein Bedarf zur Anpassung. Zur Verwendung des Referenzmechanismus zum Zeitabgleich zwischen MAP Client und Server müssen neue operative Metadaten eingeführt und ein entsprechendes XML Schema erstellt werden. Zusätzlich soll die Möglichkeit zur einfachen

Erstellung herstellerspezifischer Metadaten unter Verwendung eines eigenen Namensraums geschaffen werden.

Die optionale Möglichkeit zur Schema-Validierung von Metadaten und erweiterten Bezeichnern benötigt die Möglichkeit zur Bekanntgabe aller verwendeten Schemata im MAP Server irond. Hierzu soll die Konfigurationsdatei mit neuen Parametern versehen werden. Zusätzlich soll der Server in einem abgeschotteten Modus betrieben werden können, in dem er nicht nur die Validität der XML Daten prüft, sondern auch Metadaten, für die kein Schema bekannt ist, ablehnt. Für erweiterte Bezeichner bietet es sich ebenfalls an einen abgeschotteten Modus einzuführen, der die Bezeichner ablehnt, sofern keine erfolgreiche Validierung durchgeführt werden kann.

Bei der Sitzungsverwaltung im MAP Server irond sollen die konfigurierbaren Einstellungen zur Erhaltung von TCP-Sitzungen sowie zum Abbruch bei blockierenden Netzwerkoperationen eingeführt werden. Der Referenzmechanismus zur Zeitsynchronisation, bestehend aus dem Ablauf der Veröffentlichung von Zeitinformationen mit direkt folgender Suchanfrage nach derselben zur Erkennung einer Zeitabweichung und anschließender Synchronisation der Client-Zeit soll vollständig als automatische und manuelle Funktion in die ifmapj-Bibliothek implementiert werden. Folgende Funktionalitäten sind zu implementieren oder anzupassen:

Funktionalität	Anw.	Art der Umsetzung
Adm. Domäne	ifmapj	Nicht mehr zu verwendende Funktionsaufrufe als veraltet (*engl.* deprecated) markieren.
Bezeichner	ifmapj	Funktionalität zur Namenserzeugung bei „$device$“ und „$access-request$“ Bezeichnern bereitstellen.
Gerätebezeichner	irond	Strikt zwischen „$aik-name$“ und „$name$“ bei Gerätebezeichnern unterscheiden.
Identitätsbezeichner	ifmapj	Normalisierung notwendig bei einigen Identitätstypen.
Erw. Bezeichner	ifmapj	Implementation von Funktionen zur einfachen Verwendung erweiterter Bezeichner.
Publisher ID	irond	Den beschriebenen Mechanismus zur Generierung in irond implementieren.
Metadaten	ifmapj	Funktionen bereitstellen zur Erstellung herstellerspezifischer und operativer Metadaten.
Schema-Validierung	irond	Vollständige Umsetzung zur Validierung von Metadaten und erweiterten Bezeichnern.
Sitzungen	irond	TCP Keep-Alive und TCP Timeout als konfigurierbare Option implementieren.
Zeitsynchronisation	ifmapj	Den spezifizierten Referenzmechanismus vollständig implementieren.

5 Implementierung der Software

In diesem Kapitel werden die einzelnen Implementationen betrachtet, die zur Umsetzung der IF-MAP 2.1 Spezifikation notwendig waren. Im Rahmen der Umsetzung wurde die Version 0.1.5 der ifmapj-Bibliothek sowie den dazugehörigen ifmapj-examples erstellt. Diese ist vollständig abwärtskompatibel und kann somit ohne Anpassungen in bestehenden Umgebungen genutzt werden.

Des Weiteren wurde der MAP Server irond durch die Implementation in der Version 0.3.5 sowie das Visualisierungstool irongui in der Version 0.4.1 erstellt.

5.1 Bedingungen zur Nutzung einer administrativen Domäne

Um auch weiterhin die Angabe einer administrativen Domäne am MAPS zu ermöglichen, bedarf es dort nur einer minimalen Änderung. In der ifmapj-Bibliothek hingegen sind kleinere Anpassungen notwendig.

5.1.1 Anpassungen ifmapj

Hier wurde in der bestehenden Klasse `AccessRequest` der Konstruktor überladen, um eine Instanziierung ohne Angabe einer administrativen Domäne zu ermöglichen. Hierbei wäre es auch denkbar die Vererbung von `Identifier` direkt, anstatt über `IdentifierWithAd` vorzunehmen. Da es jedoch weiterhin möglich sein soll eine Domäne anzugeben, wird diese bei fehlender Angabe standardmäßig als NULL-Wert gespeichert. Dies sichert die Abwärtskompatibilität zu bisherigen Clientimplementationen, bei denen die Angabe als notwendig erachtet wurde. Zusätzlich wurde der bisherige Konstruktor der Klasse sowie die bestehende Funktion `setAdministrativeDomain()` per Annotation als „deprecated“ (veraltet) gekennzeichnet.

5.1.2 Anpassungen ifmapj-examples

Bei den ifmapj-examples wurde, um die Anforderung umzusetzen, in den Klassen `PublishExample` sowie `PublishExample2` bei den Aufrufen der Funktion `Identifiers.createAr()` die administrative Domäne entfernt.

5.1.3 Anpassungen irond

Die Klasse `AccessRequest` wurde mit einem weiteren Konstruktor überladen bei dem die Angabe der administrativen Domäne fehlt. Gleichzeitig wurde der bisherige Konstruktor als veraltet markiert.

5.2 „Device“ und „AccessRequest“ Bezeichner

Die Bedingungen zur Festlegung des Namens eines `Device` und `AccessRequest` Bezeichners wurden genauer festgelegt. Zur Umsetzung der Spezifikation wurde die ifmapj-Bibliothek sowie der MAP Server irond angepasst.

5.2.1 Anpassungen ifmapj

Um den Anforderungen für einen Namen eines Gerätebezeichners zu entsprechen sind neue Funktionen in den Klassen `AccessRequest` und `Device` erstellt worden. Beide Klassen müssen die gleiche Funktionsweise zur Erstellung von Namen aufweisen. Hier die Übersicht der identischen Funktionen beider Klassen:

Methode	Funktionsweise
`setName(String)`	Setzen des angegebenen Namens.
`setNameUUID()`	Setzen einer eindeutigen UUID als Name.
`setNameRandom()`	Ein Name basierend auf einer 128-bit Zufallszahl.

Zur Generierung einer eindeutigen UUID wird die Funktion `randomUUID()` aus der Java-Klasse `java.util.UUID` genutzt. Diese erstellt eine Zeichenkette, die eine eindeutige UUID darstellt. Die UUID wird als Name des Bezeichners gesetzt.

Um einen Namen basierend auf einer 128-bit Zufallszahl zu generieren, wurde in der Klasse `StringHelpers` die Funktion `getRandomString(int)` erstellt. Die Funktion akzeptiert als Argument einen Ganzzahlwert, der die Länge der Zeichenkette repräsentiert, die ausgegeben werden soll. In Falle einer 128-bit Zufallszahl muss, da jedes Zeichen acht Bit enthält, eine Zeichenkette mit einer Länge von 16 Zeichen erstellt werden. Um nicht-druckbare Zeichen auszuschließen wird von der zufälligen Zeichenkette eine MD5 Prüfsumme ermittelt und diese als Zufallszahl verwendet. Sofern MD5 in der Laufzeitumgebung nicht zur Verfügung steht, wird die Zeichenkette alternativ Base64[39]-encodiert.

5.2.2 Anpassungen irond

Zur Wahrung der Abwärtskompatibilität wurden am MAP Server irond im Zusammenhang mit dem `AccessRequest` Bezeichner nur die bereits erwähnten Änderungen im Zusammenhang mit der Angabe einer administrativen Domäne vorgenommen. Dieser akzeptiert, wie bisher, jede beliebige Zeichenkette als administrative Domäne für den Bezeichner.

Der Gerätebezeichner wurde im Zusammenhang mit der Überprüfung auf Gleichheit der Spezifikation entsprechend angepasst. Es wird nun zwischen den Attributen „$aik - name$“ und „$name$“ strikt unterschieden und diese nicht mehr gleichgesetzt.

[39] `http://en.wikipedia.org/wiki/Base64`

Listing 5: Die Funktion `equals()` berücksichtigt jetzt das Typenattribut

```
@Override
public boolean equals(Object o) {
   if (this == o) return true;
   if (!(o instanceof Device)) return false;
   Device d = (Device) o;
   return mValue.equals(d.getValue()) &&
         mType.equals(d.getDeviceType());
}
```

5.3 Regeln zum Vergleich von Bezeichnern

Die notwendigen Unterscheidungen bei der Vergleichsoperation zweier Bezeichner wurden vollständig implementiert. Erweiterte Bezeichner und eindeutige X.500 Identitätsbezeichner werden jetzt gemäß der Spezifikation besonders behandelt. Die notwendigen Änderungen sind eigentlich auf die Objekte vom Typ `Identity` beschränkt, da erweiterte Bezeichner ebenfalls Identitäten vom Typen „$other$“ sind. Es war möglich, die erstellte Implementation zum Vergleich in der ifmapj-Bibliothek für den MAP Server irond zu übernehmen.

5.3.1 Anpassungen ifmapj

Zum Vergleich zweier Identitätsbezeichner wurde in der ifmapj-Bibliothek in der Klasse `Identity` die Funktion `equals(Identity)` hinzugefügt, welche zwei Identitäten beliebigen Typs miteinander vergleicht.

Listing 6: Die Funktion `equals()` vergleicht Identitäten je nach Typ unterschiedlich

```
@Override
public boolean equals(Object o) {
   if (!(o instanceof Identity))
      return false;
   if(this == o)
      return true;
   Identity i = (Identity) o;
   if (!mType.equals(i.getType()))
      return false;
   // check administrative domain in super-class
   if (!super.equals(i))
      return false;
   if (mType == IdentityType.other && !mOtherTypeDefinition.equals(i.
       getOtherTypeDefinition()))
      return false;
   if (mType == IdentityType.other && mOtherTypeDefinition.equals(
       IfmapStrings.OTHER_TYPE_EXTENDED_IDENTITY) && !
       extendedIdentifierEquals(mName, i.getName()))
```

```
16        return false;
17    else if (mType == IdentityType.distinguishedName && !
          distinguishedNameEquals(mName, i.getName()))
18        return false;
19    else if (!mName.equals(i.getName())) return false;
20        return true;
21 }
```

5.3.2 Anpassungen irond

Hier wurde die `equals()`-Funktion in der Klasse `Identity` auf die gleiche Weise erweitert wie in der ifmapj-Bibliothek. Zusätzlich wurde hier jedoch auch eine `hashCode()`-Funktion implementiert, damit bei großen Datenmengen die Suche performant bleibt.

Einige Bezeichnertypen werden bei Vergleichen speziell behandelt. Dies muss auch die `hashCode()`-Funktion wiederspiegeln. Hierzu wurde in allen von `Identifier` geerbten Klassen die Funktion nachträglich implementiert. Dies beinhaltet neben der `Identity`-Klasse auch die Klassen `AccessRequest`, `Device`, `IpAddress` und `MacAddress`.

Insbesondere werden IP-Adressen jetzt nicht mehr nur als Zeichenkette repräsentiert, sondern ebenfalls als Objekte vom Typ `InetAddress`. Dies hätte den Vorteil, dass gleiche Adressen in unterschiedlicher Notation bei einem Vergleich korrekt erkannt werden würden. Allerdings gibt die Spezifikation vor, dass ein Server eine Adresse in falscher Notation strikt abweisen soll.[3, S. 19] Um die korrekte Notation zu erkennen war es notwendig, den Prozess zur Normalisierung ebenfalls im MAPS zu implementieren. Technisch wird die IP-Adresse als Zeichenkette an eine neue Instanz vom Typ `Inet6Address` oder `Inet4Address` übergeben. Dabei wird die entsprechende kanonische Schreibweise automatisch erzeugt und kann zum Vergleich mit der vom Client erhaltenen Zeichenkette herangezogen werden. Sofern der Vergleich fehlt schlägt, wurde die Adresse nicht in der korrekten Notation übergeben und die Erstellung des `IpAddress`-Objektes wird durch das Auslösen einer `InvalidIdentifierException` unterbrochen.

Die bisherige Implementation zur Zwischenspeicherung bereits berechneter Hash-Werte wurde an die zusätzlichen Funktionen angepasst. Dabei wird in der Methode `getHashCode()` der Hash-Wert berechnet und in einer Instanzvariable gespeichert. Der Abruf über die Methode `hashCode()` erspart so die mehrfache Ausführung der Berechnung. Die spezielle Behandlung von erweiterten Bezeichnern und eindeutigen X.500 Namen erfordert in der Klasse `Identity` eine etwas komplexere `getHashCode()`-Funktion:

Listing 7: Berechnung eines Hashcodes zum schnellen Auffinden von Einträgen

```
1 protected final int getHashCode() {
2     int hash = 11 * 97 + (mType != null ? mType.hashCode() : 0);
```

```
3     hash = 11 * hash + mOtherTypeDefinition.hashCode();
4     if (mOtherTypeDefinition.equals(IfmapConstStrings.ID_OTHER_EXT))
5        hash=11 * hash + DomHelpers.calculateHashCode(
             mExtendedIdentifier);
6     else if (mType == IdentityTypeEnum.distinguishedName &&
          mDistinguishedName != null)
7        hash = 11 * hash + mDistinguishedName.getName().hashCode();
8     else
9        hash = 11 * hash + ((mCaseSensitive) ? mName.hashCode() : mName.
             toLowerCase().hashCode());
10    return 11 * hash + super.getHashCode();
11 }
```

5.4 Gleichheitsbedingungen von „distinguished-name“ Identitäten

Die Behandlung von eindeutigen X.500 Namen sieht vor, diese nach RFC 3280[40] zu vergleichen. In Java ist bereits ein äquivalenter Mechanismus zum Vergleich zweier Objekte vom Typ `X500Principal` enthalten, welcher für diesen Zweck verwendet wird.

5.4.1 Anpassungen ifmapj

In die Bibliothek wurde bei der neu hinzugekommenen Funktion `equals()` in der Klasse `Identity` die Vergleichsoperation nach RFC 2459[41] implementiert, um Clients eine einfache Möglichkeit zum Vergleich zweier „$distinguished - name$“ Identitäten zu ermöglichen.

Zur Wahrung der Abwärtskompatibilität ist es immer noch möglich eine beliebige Zeichenkette als eindeutigen X.500 Namen in einer Identität festzulegen. Allerdings wird in diesem Fall die Vergleichsoperation per `equals()`-Methode mit einer Ausnahme (*engl.* exception) enden, sofern die Zeichenkette kein gültiger Name ist.

Listing 8: Vergleichsoperation zweier eindeutiger X.500 Namen

```
1  /**
2   * Compare two DSN
3   * @param dsn1 Distinguished name (X.500 DSN) string
4   * @param dsn2 Distinguished name (X.500 DSN) string
5   * @return boolean true if both DSN are equal
6   * @since 0.1.5
7   */
8  private boolean distinguishedNameEquals(String dsn1, String dsn2) {
9      return new X500Principal(dsn1).equals(new X500Principal(dsn2));
10 }
```

[40] http://www.ietf.org/rfc/rfc3280.txt
[41] http://www.ietf.org/rfc/rfc2459.txt

5.4.2 Anpassungen irond

Beim MAP Server irond werden die „$distinguished - name$“ Identitäten in Objekten vom Typ `javax.X500.X500Principal` gespeichert. Dies ermöglicht einen schnellen und einfachen Vergleich zweier DSN per `equals()`-Funktion, da die `hashCode()`-Funktion bereits korrekt implementiert ist.

Zur Wahrung der Abwärtskompatibilität zu IF-MAP 2.0 werden weiterhin ungültige Angaben akzeptiert, welche sich nicht in ein `X500Principal` Objekt überführen lassen. Sofern dieser Fall eintritt, wird lediglich ein einfacher Vergleich von Zeichenketten durchgeführt.

Die Möglichkeit beim MAP Server irond eine Identität unabhängig von ihrer Groß- und Kleinschreibung zu vergleichen wurde so umgesetzt, dass intern vor der Konvertierung in ein `X500Principal` Objekt alle Großbuchstaben der Zeichenkette per `toLowerCase()`-Methode in Kleinbuchstaben umgewandelt werden.

5.5 Erweiterte Bezeichner

Die Spezifikationen sehen die Möglichkeit zur Nutzung erweiterter Bezeichner (*engl.* extended identifier) vor. Hierbei handelt es sich um XML Daten, die in ein `Identity`-Objekt gekapselt werden. Diese müssen auf dem von der TCG festgelegtem XML Schema für erweiterte Bezeichner basieren.

5.5.1 Anpassungen ifmapj

Die Funktion `createIdentity()` in der Klasse `IdentifierFactoryImpl` wurde überladen, um XML Daten vom Typ `org.w3c.dom.Document` direkt als Parameter zu akzeptieren. Hierbei wird der `Type` des Bezeichners automatisch auf „other“, und dessen Definition entsprechend auf „extended“ gesetzt. Zusätzlich wurde in der Klasse `Identifiers` eine überladene Funktion zur Erstellung erweiterter Bezeichner hinzugefügt. Diese akzeptiert entweder ein XML Dokument als `String`, `InputStream` oder als `org.w3c.dom.Document`.

Die XML Daten werden durch den Aufruf von `normalize()` normalisiert und für die Übertragung an den MAPS vorbereitet. Hierbei wird ein ggf. vorhandener Namensraum und die XML Kopfzeile entfernt.

Zur Entfernung des Namensraums wird eine XSL Transformation (siehe Listing A.6.6 XSLT Remove Namespace) durchgeführt, welche einen beliebigen Namensraum in einen fortlaufend nummerieren Standardnamensraum „nsX“ überführt. Im Anschluss wird mittels einer einfachen Zeichenkettenersetzung, unter Nutzung eines regulären Ausdrucks, dieser Namensraum noch entfernt.

5.5.2 Anpassungen irond

Durch die Kapselung der erweiterten Bezeichner in den Typen „*other*“ ist dessen Nutzung grundsätzlich schon möglich gewesen. Jedoch war kein entsprechender Vergleichsalgorithmus für XML Dokumente enthalten.

Um den Vergleich sowie eine eventuelle Schemavalidierung aus der Sichtweise des DOM[42] zu ermöglichen, werden die erweiterten Bezeichner in Objekte vom Typ `org.w3c.dom.Document` überführt. Die hierfür notwendige Funktion befindet sich in der Hilfsklasse `DOMHelpers`.

Listing 9: Verarbeitung von erweiterten Bezeichnern

```
1 private Document prepareExtendedIdentifier(String xml)
2       throws SAXException, IOException {
3    if (!mCaseSensitive)
4       xml = DomHelpers.unescapeXml(xml.toLowerCase());
5    else
6       xml = DomHelpers.unescapeXml(xml);
7    return DomHelpers.toDocument(xml, null);
8 }
```

Die Notwendigkeit zum Vergleich zweier erweiterter Bezeichner wurde ermöglicht indem zwei weitere Hilfsfunktion zum Vergleich von XML Dokumenten sowie zur Berechnung eines Hashcodes zu einem `org.w3c.dom.Document` erstellt wurden.

Vor dem Vergleich werden die XML Dokumente normalisiert um unerhebliche Abweichungen zu entfernen. Anschließend wird der Vergleich per `isEqualNode()`[43]-Funktion durchgeführt.

Listing 10: Hilfsfunktion zum Vergleich von XML Dokumenten

```
1 /**
2  * Compare two DOM documents
3  * @param d1 First DOM document
4  * @param d2 Second DOM document
5  * @return true if both are equal
6  */
7 public static boolean compare(Document d1, Document d2) {
8    d1.normalize();
9    d2.normalize();
10   return d1.isEqualNode(d2);
11 }
```

Die Berechnung des Hashcodes erfolgt indem das XML Dokument in ein `String`-Objekt überführt wird. Dieses stellt über die interne `hashCode()`-Funktion den

[42] http://de.wikipedia.org/wiki/Document_Object_Model

[43] http://www.w3.org/TR/2003/WD-DOM-Level-3-Core-20030226/DOM3-Core.html#core-Node3-isEqualNode

Hashcode bereit.

Listing 11: Berechnung des Hashcodes eines erweiterten Bezeichners

```
1  public static int calculateHashCode(Document doc) {
2     try {
3        doc.normalize();
4        Transformer t=TransformerFactory.newInstance().newTransformer();
5        t.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
6        t.setOutputProperty(OutputKeys.INDENT, "no");
7        StringWriter sw = new StringWriter();
8        DOMSource source = new DOMSource(doc);
9        t.transform(source, new StreamResult(sw));
10       Integer hc = sw.getBuffer().toString().hashCode();
11       logResult(hc.toString(), null);
12       return (int)hc;
13    } catch (TransformerException e) { return 0; }
14 }
```

5.5.3 Anpassungen irongui

Es wurde ein neues Dialogfenster erstellt, welches das bisherige Kontextmenü zur Erstellung eines Abonnements (*engl.* subscription) ablöst. Diese Änderung wurde notwendig da in einem Kontextmenü Steuerelemente, wie in diesem Fall Knöpfe, nicht richtig funktionieren. Hierzu wurde die Menüstruktur in ein Fenster mit Reitern überführt.

Sobald der Identitätstyp „$extended$“ ausgewählt wurde, wird die „$other-type-definition$“ auf diesen Wert festgesetzt. Außerdem kann nun über den Knopf *„Load XML...“* eine XML Datei, die einen erweiterten Bezeichner repräsentiert, geladen werden. Der Inhalt der XML Datei wird in dem Feld *„name“* eingetragen.

Zur Übermittlung zwischen MAP Client und Server werden die erweiterten Bezeichner innerhalb von XML Anfragen gekapselt. Dazu müssen XML Steuerzeichen maskiert werden. Die gleiche Maskierung ist zur Darstellung des Bezeichners durch irongui notwendig, da in einer Kurzinfo (*engl.* tooltip) HTML Sequenzen erlaubt sind, die die gleichen Steuerzeichen zur Markierung von Elementen verwenden. Da die, sich durch die Maskierung ergebenen, XML Entitäten ihren Pendants in HTML entsprechen, ist eine Darstellung von XML innerhalb von HTML durch einfaches maskieren der Steuerzeichen möglich. Hierzu wurden in der Klasse `IfmapUtils` die zwei neuen Funktionen `escapeXml()` und `unescapeXml()` geschaffen. Genutzt werden die beiden Funktionen in den Klassen `TreePanel`, `GraphPanel` und `IdentifierData`.

Der neue Dialog zur Erstellung eines Abonnements befindet sich in der Klasse `SubscriptionDialog`. Die beiden obsoleten Klassen `SubscribeDialog` und `Subscription_GUI` wurden konsequenterweise entfernt. Beide enthielten unterschiedliche Ansätze eines Benutzerdialogs zur Erstellung eines Abonnements. Die-

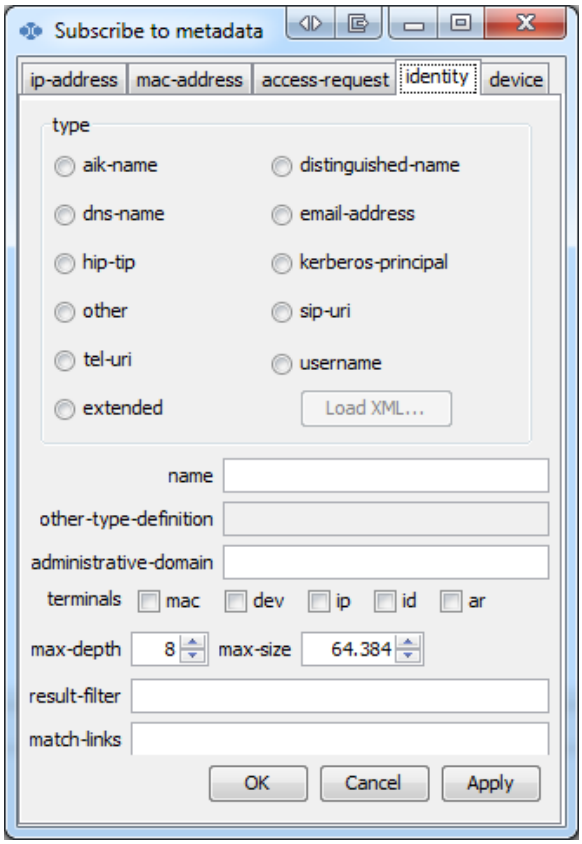


Abbildung 4: Neu erstelltes Dialogfenster „Subscribe to metadata“ in „irongui“

se wurden nun in einem Dialogfenster kombiniert. Dabei bleiben bestehende Klassen erhalten, die Teile des Dialogs darstellen. Nur die Zusammensetzung der Elemente wurde neu angeordnet und in einem Fenster anstatt im Kontextmenü untergebracht.

Die Klasse `ViewController` wurde angepasst, um die neuen Schaltflächen mit ihren entsprechenden Funktionen zu belegen. Gleichzeitig ist nicht mehr verwendeter Code zur Steuerung der alten Dialoge entfernt worden.

5.5.4 Beispielimplementation

Beispiele zur Nutzung und Übertragung von erweiterten Bezeichnern wurden den ifmapj-examples hinzugefügt. Hierzu wurde zu den bestehenden Beispielen die Klassen `ExtendendIdentityExample` und `ExtendendIdentityExample2` hinzugefügt.

In dem Beispiel wird eine XML Datei aus dem Dateisystem geladen und an den MAPS übertragen. Hierbei ist das übergebene XML Dokument vom Typ

`InputStream` und kann daher per `getResourceAsStream()` direkt eingelesen werden.

Listing 12: Erstellung eines erweiterten Bezeichners und Veröffentlichung

```
// create IF-MAP 2.1 compliant extended identifier from
// any XML document (see spec. 3.2.3.2 for details)
Identifier extId = Identifiers.createExtendedIdentity(getClass().
    getResourceAsStream(Config.EXTENDED_IDENTITY_XML));

// Create a PublishUpdate object containing the extended identifier
PublishUpdate pu = Requests.createPublishUpdate(dev, extId, mF.
    createAuthAs());
pu.setLifeTime(MetadataLifetime.forever);
ssrc.publish(Requests.createPublishReq(pu));
```

5.6 Erstellung der Publisher ID

Die Erstellung einer Publisher ID für einen Client obliegt dem MAP Server. Es sind somit keine Änderungen an der ifmapj-Bibliothek notwendig, da diese nur ihre jeweils zugeteilte Identifikation in Serveranfragen verwendet.

Beim MAP Server irond wurde es notwendig die Klasse `ClientIdentifier` zu erweitern sowie die Klasse `PublisherIdGeneratorImpl` vollständig umzuschreiben. In beiden Klassen wurde der Konstruktor geändert, um alle notwendigen Informationen zur Berechnung der Publisher ID zu erhalten.

Zusätzliche wurde eine Funktion geschaffen, die eine lesbare Identifikation des Clients erstellt, um die Zuordnung in der Konfigurationsdatei `publisher.properties` abzuspeichern. Dies ermöglicht, durch manuelle Anpassung in der Konfigurationsdatei, die Beibehaltung der bisherigen Publisher ID, auch wenn sich einer der relevanten Parameter zur Berechnung der Identifikation ändern sollte.

Listing 13: Erzeugen einer lesbaren Identifikation eines Clients

```
public String getIdentificationString() {
   String suffix = "";
   if (!mIpAddress.isEmpty())
      suffix = ";" + mIpAddress;
   if (mSubject.isEmpty())
      return mUsername + suffix;
   else
      return "\"" + mSubject + "\";\"" + mIssuer + "\"" + suffix;
}
```

Die Zuordnung der Identifikation eines Clients zu seiner Publisher ID wird lesbar in der Konfigurationsdatei `publisher.properties` gespeichert. Bei Bedarf können dort Änderungen an der Zuordnung vorgenommen werden. Beispielsweise, wenn sich der Benutzername eines bestehenden Clients ändern sollte.

Listing 14: Die Funktion zur Erstellung einer eindeutigen Publisher ID

```
1 public String generatePublisherIdFor(ClientIdentifier cid) {
2    return cid.getUsername() + "-" + MD5Provider.getMD5("iron." + cid.
         getUsername() + "." + cid.getSubject() + "." + cid.getIssuer()
         + "." + cid.getIpAddress());
3 }
```

Unverändert zur bisherigen Implementation wird die erstellte Publisher ID dem MAP Client zur Nutzung bei Veröffentlichungen auf dem Server mitgeteilt. Hierbei wird vom Server die zugeteilte ID bei zukünftigen Anfragen erwartet und alle Anfragen auf ihr Vorhandensein hin geprüft.

5.7 Herstellerspezifische Metadaten

Zur Erweiterung der Möglichkeiten in der Verwendung der ifmapj-Bibliothek wurden drei neue Methoden erstellt, die es ermöglichen herstellerspezifische Metadaten zu erzeugen. Um die Funktionalität bereitzustellen, wurde die Klasse `StandardIfmapMetadataFactoryImpl` um drei Aufrufe der `create()`-Funktion erweitert.

An diese Funktion kann optional ein oder mehrere Attributnamen und -werte übergeben werden. Der `qualifiedName` wird als Präfix für den anzugebenden Namespace URI genutzt.

Auszugsweise hier die Methode mit der entsprechenden Funktionalität zur Erstellung herstellerspezifischer Metadaten. Die weiteren Überladungen der Methode ändern die Übergabeparameter zu den Attributen entsprechend ab und rufen anschließend diese Funktion auf:

Listing 15: Die Funktion „create()“ zur Erstellung herstellerspezifischer Metadaten

```
1  @Override
2  public Document create(String elementName, String qualifiedName,
       String uri, Cardinality cardinality, HashMap<String, String>
       attributes) {
3     Document doc = mDocumentBuilder.newDocument();
4     Element e=doc.createElementNS(uri, qualifiedName+":"+elementName);
5     e.setAttributeNS(null,"ifmap-cardinality",cardinality.toString());
6     for (Map.Entry<String, String> attr : attributes.entrySet())
7         e.setAttributeNS(uri, attr.getKey(), attr.getValue());
8     doc.appendChild(e);
9     doc.createAttributeNS(uri, qualifiedName);
10    return doc;
11 }
```

Zusammenfassend eine kurze Übersicht zu den möglichen Funktionsaufrufen zur Erstellung herstellerspezifischer Metadaten:

Funktion	Metadaten als
create(elementName, qualifiedName, uri, cardinality)	keine
create(elementName, qualifiedName, uri, cardinality, attrName, attrValue)	String
create(elementName, qualifiedName, uri, cardinality, attrName[], attrValue[])	HashMap

5.8 XML Schema-Validierung

Zur Validierung werden beim irond XML Dokumente von der übertragenen Zeichenkette in eine Instanz der Klasse `org.w3c.dom.Document` überführt. Anschließend können die Dokumente standardkonform über die Klassenmethode `isEqualNode()` miteinander verglichen oder per `SchemaFactory` gegen ein XML Schema validiert werden.

Im Rahmen dieser Arbeit ist die Hilfsklasse `DomHelpers` erstellt worden. Hier sind bereits die Funktionen zum Vergleich und zur Überführung in eine `Document`-Instanz bereitgestellt worden. Dazu wurde jetzt noch die Funktion `validate()` geschaffen, welche die Validierung eines XML Dokumentes gegen ein XML Schema ermöglicht. Als Parametersatz für die Funktion wird das `Document` sowie eine `StreamSource` zum entsprechenden XML Schema (XSD) erwartet. Diese Funktionalität kann nun zur Validierung von Metadaten und erweiterten Bezeichnern verwendet werden.

Um eintreffende Metadaten zu validieren, ist die Klasse `MetaDataFactory` und dessen Implementation `MetaDataFactoryImpl` angepasst worden. Insbesondere wurde der Konstruktor erweitert, um Informationen über die ggf. vollzogene Validierung bei Objekterstellung nach der Übertragung zu erhalten.

Die Klasse `Identity` beinhaltet jetzt eine Möglichkeit die erweiterten Bezeichner ebenfalls gegen ein XML Schema zu validieren. Hierbei war es notwendig, der Klasse die Serverkonfiguration bekannt zu machen, da in dieser die Zuordnung eines XML Namespace URI zu der dazugehörigen XSD-Datei im Unterverzeichnis `schema` erfolgen kann. Über die bereits erwähnte `DomHelpers`-Klasse wird die Validierung gegen das Schema durchgeführt.

Listing 16: Funktion zum Heraussuchen der passenden Schemadatei für einen Namespace URI

```
1 @Override
2 public StreamSource getExtendedIdentifierSchema(String uri) {
3    if (getExtendedIdentifierSchemaFileNames().containsKey(uri))
4       return new StreamSource(new File(
            getExtendedIdentifierSchemaFileNames().get(uri)));
5    else
6       return null;
7 }
```

5.9 Behandlung von Sitzungen

Die Kommunikation zwischen MAP Client und Server erfolgt über das HTTPS-Protokoll, welches auf TCP/IP Verbindungen basiert. Hierbei wird eine ständige Verbindung zwischen den Kommunikationspartnern aufrecht erhalten.[44] Unter Umständen kann die Verbindung unterbrochen werden, ohne dabei korrekt geschlossen worden zu sein. Um den Status einer Verbindung periodisch prüfen zu können, ohne eine entsprechende Funktion auf höheren Protokollschichten implementieren zu müssen, beinhaltet das TCP Protokoll die Funktion Keep-Alive.

Einige Funktionsaufrufe in Javas TCP-Implementation können andere Programmfunktionen blockieren solange diese auf Antworten der darunter liegenden Netzwerkschichten warten. Zur Vermeidung lang anhaltender Wartezeiten, bis ein Fehler an die darüber liegende Protokollschicht durchgereicht wird, kann ein Zeitüberschreitungswert festgelegt werden, ab wann eine Netzwerkoperation erfolglos abgebrochen werden soll.

Zur Umsetzung der Möglichkeit die TCP Socket Option Keep-Alive zu verwenden, sowie zur Anpassung der Zeitüberschreitung für blockierende Funktionsaufrufe, wurde die Konfiguration des MAP Servers irond erweitert. Hierzu wurden zwei neue Einstellungen in der Konfigurationsdatei `ifmap.properties` geschaffen. Dies sind die neu hinzugekommen Einstellungen:

Funktion	Beschreibung
`irond.comm.socket.keep-alive`	TCP Keep-Alive aktivieren? (true/false)
`irond.comm.socket.timeout`	Zeitüberschreitung in Millisekunden

Die Klasse `ServerConfigurationProvider` sowie die dazugehörige Implementation `ServerConfigurationProviderImpl` dienen zum Auslesen und Setzen der Standardwerte in der Konfigurationsdatei `ifmap.properties`. Hier wurden die zwei neuen Funktionen `getSocketKeepAlive()` und `getSocketTimeout()` eingefügt. Beide Funktionen liefern dem Programm die jeweils in der Konfiguration eingestellten Werte oder den Standardwert.

Der Thread zur Etablierung von Clientverbindungen befindet sich in der Klasse `ChannelAcceptor` und wird in der `run()`-Methode durch eine Endlosschleife implementiert. Innerhalb der Funktion wird, sobald eine neue Verbindung von einem Client aufgebaut wird, diese jeweils an eine eigene Instanz der Java-Klasse `SSLSocket` gebunden. Dabei werden die in der Konfiguration gesetzten Parameter oder die festgelegten Standardwerte an die Instanz übergeben.

Listing 17: Anpassungen in der Klasse `ChannelAcceptor`

```
1 SSLSocket clientSocket = (SSLSocket) mSocket.accept();
2 clientSocket.setKeepAlive(mConfReader.getSocketKeepAlive());
3 clientSocket.setSoTimeout(mConfReader.getSocketTimeout());
```

[44] `http://www.ietf.org/rfc/rfc2616.txt`

Sinnvoll kann die TCP Keep-Alive Funktion nur genutzt werden, sofern die Zeitüberschreitung einer Sitzung mehr als zwei Stunden beträgt, da sonst die Verbindung bevor das erste Keep-Alive Paket gesendet wird, schon vom Server geschlossen wurde.

5.10 Zeitsynchronisation

Einige Metadaten können einen zeitlichen Bezug aufweisen. Eine Abstimmung der Client- und Serverzeit ist in diesen Fällen unerlässlich, um einen zuverlässigen Betrieb zu gewährleisten. In Abschnitt 4.6.1 der IF-MAP 2.1 Spezifikation wird hierfür eine Referenzimplementation beschrieben. Diese sieht die Übertragung spezieller Metadaten vor, die einen Zeitstempel des Client enthalten. Dabei muss ein Gerätebezeichner erzeugt und zusammen mit den Metadaten an den Server übertragen werden. Im Anschluss an die Übertragung erfolgt eine Suchanfrage an den Server, welche die soeben übertragenen Metadaten abfragt. In der Antwort des Servers sind nun zwei Zeitstempel enthalten, da der MAPS bei der Veröffentlichung der Metadaten seinen eigenen Zeitstempel eingefügt hat. Jetzt vergleicht der Client den zeitlichen Abstand beider Zeitstempel des Suchergebnisses, um seine Zeitabweichung zum MAP Server festzustellen. Es obliegt von nun an dem Client, seine Zeitangaben in Metadaten an die Uhr des Servers anzupassen.[3, S. 54ff]

5.10.1 Anpassungen ifmapj

Die in der Spezifikation vorgeschlagene Referenzimplementation zur Erkennung von Zeitabweichungen wurde in die ifmapj-Bibliothek implementiert. Zur Nutzung muss bei der Erstellung eines SSRC lediglich ein Gerätebezeichner übergeben werden. Der anschließende Aufruf von `newSession()` führt den in der Spezifikation genannten Prozess für den Client transparent durch und es stehen anschließend die folgenden Funktionsaufrufe zur Verfügung:

- **`getClockSynchronized()`** - Konnte die Uhrzeit erfolgreich mit dem Server synchronisiert werden? (boolean)
- **`getClockSkewMilliseconds()`** - Zeitabweichung in Millisekunden (long)
- **`getClockSkewSeconds()`** - Zeitabweichung in Sekunden (int)
- **`getClockFromServer()`** - Gibt die Serverzeit zurück (Calendar)
- **`getClockLastSynchronization()`** - Gibt den Zeitpunkt der letzten Synchronisation zurück (Calendar)
- **`performClockSynchronization()`** - Auslösen eines neuen Synchronisationsvorgangs (void)

- **performClockSkewDetection()** - Manuelle Durchführung einer Synchronisation unter Angabe eines Gerätebezeichners und optional einer Zeit (long)

Weitere Details zu den Rückgabe- und Aufrufparametern der einzelnen Funktionsaufrufe können in der JavaDoc Dokumentation eingesehen werden. Es wird empfohlen, die Synchronisationen in regelmäßigen Abständen erneut durchzuführen, um ein Abweichen der Uhren zwischen Server und Client dauerhaft zu vermeiden.[3, S. 54]

Es reicht aus, die Funktion `performClockSynchronization()` oder aber alternativ gleich `renewSession()` aufzurufen. Dabei werden die operativen Metadaten erneut publiziert, vom Server abgefragt und ausgewertet.

Listing 18: Erstellen eines SSRC Kanals mit automatischer Zeitsynchronisation durch die zusätzliche Angabe eines Gerätebezeichners

```
1  /*
2   * Use the new createSSRC() function and supply a device
3   * identifier for automated time synchronization on
4   * newSession() and renewSession() calls.
5   */
6  SSRCImpl ssrc = IfmapJ.createSSRC(
7      Config.BASIC_AUTH_SERVER_URL,
8      Config.BASIC_AUTH_USER, Config.BASIC_AUTH_PASSWORD,
9      IfmapJHelper.getTrustManagers(getClass().getResourceAsStream(
10             Config.TRUST_STORE_PATH), Config.TRUST_STORE_PASSWORD),
11     Identifiers.createDev());
```

Innerhalb der Klasse `SSRCImpl` wurden zwei weitere, als privat deklarierte Funktionen implementiert. Die Funktion `publishClientTime()` dient der Veröffentlichung der Clientzeit, während die Funktion `searchClientTime()` eine Suchanfrage nach der veröffentlichten Zeit an den Server stellt und das Suchergebnis auswertet. Die Clientzeit wird mit Sitzungslebensdauer veröffentlicht und verfällt sobald eine geöffnete Verbindung zum Server geschlossen wird. Die Funktion zur Suchanfrage wertet diese auch direkt aus und gibt das XML Element zurück, welches die Server- und Clientzeit aus den empfangenen Metadaten beinhaltet.

Der Aufruf von `performClockSkewDetection()` wie auch der Aufruf der Funktion `performClockSynchronization()` stößt die Veröffentlichung und Auswertung der Clientzeit erneut an. Hierbei werden die beiden privaten Funktionen entsprechend verwendet. Die erhaltenen Zeitstempel im ISO-8601[45] Format werden intern in Millisekunden umgerechnet und dann voneinander subtrahiert. Die so erhaltene Differenz gibt den Versatz beider Zeiten zueinander an.

Es reicht also aus, den ermittelten Wert zu der aktuellen Systemzeit des Client zu addieren, um die korrekte Serverzeit zu erhalten.

[45] `http://lists.ebxml.org/archives/ebxml-core/200104/pdf00005.pdf`

Wert (ms)	Bedeutung
< 0	Die Clientzeit ist der Serverzeit voraus.
0	Die Client- und Serverzeit stimmt überein.
> 0	Die Serverzeit ist der Clientzeit voraus.

Sofern bei der Instanziierung von `SSRCImpl` ein Gerätebezeichner übergeben wurde, aktualisiert die Funktion `performClockSynchronization()` zusätzlich die interne Klassenvariable `mClockSkew` und stellt den Wert somit den restlichen öffentlichen Funktionen zur Verfügung. Diese können dann in der Clientimplementation zur einfachen Erstellung von Zeitstempeln verwendet werden.

5.10.2 Anpassungen irond

Es waren keinen Anpassungen notwendig, da der implementierte Mechanismus rein client-seitig operiert und die Spezifikation bisher schon die Übertragung beliebiger Metadaten zulässt.

Es wurde lediglich das XML Schema `ifmap-opmeta-2.1v1.xsd` erstellt und dem Paket hinzugefügt. Das Schema befindet sich im Anhang unter A.6.4. Dieses wird benötigt, wenn der Server im abgeschotteten Modus betrieben wird, in dem für sämtliche Metadaten zwingend eine positive Schema-Validierung erforderlich ist.

5.10.3 Beispielimplementation

Die exemplarische Verwendung der Zeitsynchronisation wurde mit einer eigenen Klasse in die ifmapj-examples integriert. Die Klasse `ClockSkewExample` zeigt die beiden Möglichkeiten zur Nutzung auf. Sofern bei Erstellung des SSRC Kommunikationskanals ein Gerätebezeichner übergeben wurde, stehen `die getClock...()` Funktionen vollständig zur Verfügung.

Sollte die Erstellung eines Gerätebezeichners zu diesem Zeitpunkt nicht möglich sein, so kann später eine manuelle Synchronisation durch den Aufruf von `performClockSkewDetection()` unter Angabe eines Gerätebezeichners den Versatz in Millisekunden feststellen. Außerdem besteht hier die Möglichkeit auch die Zeit des Client selbst anzugeben, falls diese von der lokalen Systemzeit abweicht.

Listing 19: Manueller Abgleich mit der Serverzeit unter Angabe eines Gerätebezeichners und optional eines Zeitstempels.

```
1 Calendar clientTime = Calendar.getInstance();
2 long clockSkew = ssrc.performClockSkewDetection(
3       Identifiers.createDev(), clientTime);
```

5.11 Zusammenfassung

Die erfolgten Implementationen und Anpassungen zur Umsetzung des entwickelten Konzepts wurden in diesem Kapitel detailliert beschrieben. Neben den Anpassungen an der ifmapj-Bibliothek und dem MAP Server irond wurde auch die Implementation der zur Bibliothek zugehörigen ifmapj-examples an die neuen Begebenheiten angepasst. Hier wurden Funktionsaufrufe angepasst und neue Beispiele zur Zeitsynchronisation und Verwendung erweiterter Bezeichner eingeführt. Außerdem wurde das Visualisierungswerkzeug irongui um die Funktionalität zur Verwendung von erweiterten Bezeichnern in Abonnementanfragen an einen MAP Server erweitert. Dazu wurde ein Dialogfenster neu gestaltet und die Darstellung von Bezeichnern für die Nutzung der erweiterten Bezeichner angepasst. Zur einfachen Handhabung können erweiterte Bezeichner jetzt als XML Dokumente geladen und in Suchanfragen verwendet werden. Die mühselige Eingabe von XML Dokumenten in ein Textfeld ist nicht notwendig, aber optional weiterhin möglich.

Bei den Geräte- und AccessRequest-Bezeichnern wurden neue Funktionen zur Erstellung eindeutiger Namen nach Vorgabe der IF-MAP Spezifikation erstellt. Dazu wurden in der ifmapj-Bibliothek die jeweiligen Klassen entsprechend erweitert. Beim MAP Server irond wurde die Vergleichsoperation eines Gerätebezeichners angepasst.

Die Funktionalität zum Vergleichen von Klasseninstanzen per `equals()` -Methode wurde für Identitäten um die Berücksichtigung des jeweiligen Identitätstypen erweitert. Sowohl der MAP Server irond als auch die ifmapj-Bibliothek bietet jetzt die gleiche Möglichkeit zum Vergleich von Identitätsbezeichnern. Dies garantiert bei konsequenter Nutzung eine gleiche Behandlung der Objekte sowohl vom MAP Server als auch vom MAP Client. Insbesondere sind hier die erweiterten Bezeichner und die eindeutigen X.500 Namensidentitäten zu nennen, bei denen die Vergleichsoperation unter Umständen vor dem Vergleich eine komplexe Normalisierung ihrer Daten erfordert.

Zur Beschleunigung der Suche, bei sehr vielen veröffentlichten Metadaten und Bezeichnern, wurde für die relevanten Objekte konsequent die `hashCode()`-Funktion implementiert, die die Suche nach Objekten über eine Ganzzahl als Index erheblich beschleunigt.

Die Erzeugung herstellerspezifischer Metadaten ist jetzt mit einem einzigen Funktionsaufruf in der ifmapj-Bibliothek möglich. Unter Angabe eines korrespondierenden Schemas kann der MAP Server auch diese Metadaten validieren und nötigenfalls als ungültig abweisen.

Zusammenfassend konnten die folgenden notwendigen Änderungen und Erweiterungen in der ifmapj-Bibliothek sowie dem MAP Server irond vorgenommen und erfolgreich getestet werden:

Funktion	Anw.	Umsetzung
Adm. Domäne	ifmapj	Weitere Konstruktoren für Klassen erstellt und nicht mehr zu verwendende Aufrufe als veraltet markiert.
Adm. Domäne	irond	Konstruktor der Klasse `AccessRequest` an die Verwendung ohne adm. Dom. angepasst.
Bezeichner	ifmapj	Funktionen zur Erstellung von Namen für `Device`- und `AccessRequest`-Bezeichner hinzugefügt.
Gerätebezeichner	ifmapj	Die notwendigen Vergleichsoperationen und Hash-Funktionen wurden implementiert.
Gerätebezeichner	irond	Die genutzte Vergleichsoperation wurde angepasst.
Identitätsbezeichner	ifmapj	Die notwendigen Normalisierungen und Vergleichsoperationen einiger Typen wurde implementiert.
Identitätsbezeichner	irond	Die notwendige Normalisierung einiger Typen wurde implementiert.
Erw. Bezeichner	ifmapj	Neue Funktionen zur Erstellung und Übertragung von erweiterten Bezeichnern wurde implementiert.
Erw. Bezeichner	irond	Anpassung der Vergleichsoperation und Gültigkeitsprüfung für erweiterte Bezeichner.
Erw. Bezeichner	irongui	Neues Dialogfenster mit zusätzlicher Funktion zur Verwendung von erweiterten Bezeichnern eingeführt.
Publisher ID	irond	Die neue Klasse zur Erstellung einer Publisher ID wurde implementiert.
Metadaten	ifmapj	Die Funktionen zur Erstellung herstellerspezifischer und operativer Metadaten in einem eigenen Namensraum wurden implementiert.
Schema-Validierung	irond	Eine Schema-Validierung aller Daten und das Setzen des entsprechenden Validierungsattributs wurde vollständig umgesetzt.
Sitzungen	irond	Das Verbindungsmanagement wurde um die Optionen zur Verwendung von TCP Keep-Alive und Socket Timeout erweitert.
Zeitsynchronisation	ifmapj	Der Referenzmechanismus wurde als automatische und manuelle Option implementiert.

6 IF-MAP 2.0 Nachweistest irond

Ein Nachweistest (*engl.* compliance test) dient der Prüfung einer Software auf die Erfüllung aller gestellten Anforderungen durch die zugrundeliegende Spezifikation. Die TCG stellt seinen Partnern eine Testumgebung zur Prüfung eigener IF-MAP Server- und Clientimplementationen bereit. Diese liegt dem Autor in der Revision r3 vom 12.März 2012 vor und ist nicht öffentlich im Internet erhältlich. Zu diesem Zeitpunkt war die IF-MAP Spezifikation noch in der Version 2.0 aktuell. Folglich waren auch keine Tests auf die im Mai veröffentlichte Überarbeitung der Spezifikation ausgelegt. Daher bietet der Nachweistest in der vorliegenden Ausgabe lediglich die Möglichkeit zur Kontrolle der Abwärtskompatibilität zu IF-MAP 2.0.

Der Nachweistest besteht aus mehreren Klassen, die als Testsuite zusammengefasst worden sind. Es besteht die Möglichkeit einzelne Tests auszuwählen, über die Kommandozeile gezielt Anfragen an einen Server zu senden, oder aber einen kompletten Durchlauf aller Servertests durchzuführen. Es wird empfohlen die Testsuite mit Java 6 zu verwenden, da einige Tests mit der derzeit aktuellen Java Version 7 scheitern.

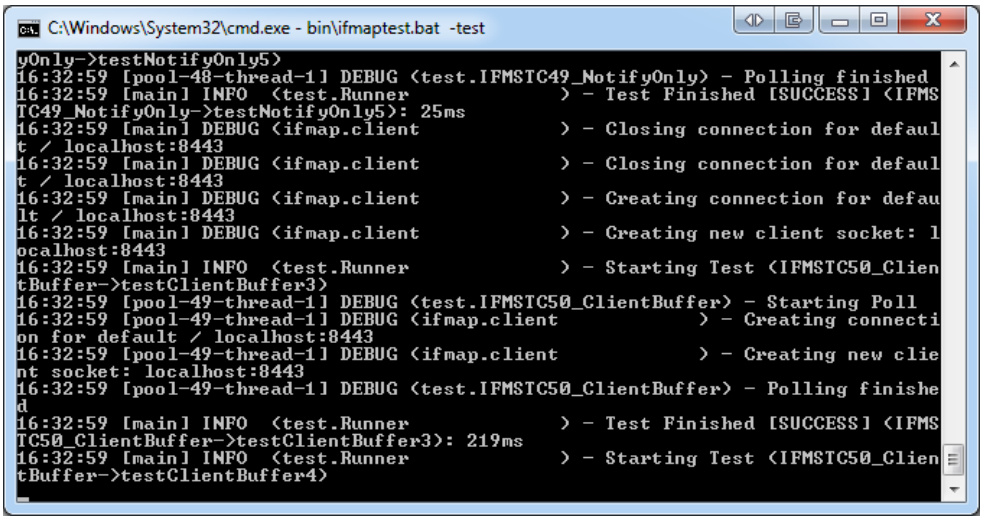


Abbildung 5: Ausführen des IF-MAP 2.0 Nachweistests auf der Kommandozeile

Zur Prüfung des überarbeiteten MAP Server irond wurde der Durchlauf aller Tests gewählt. Es wurden sämtliche Prüfungen des Tests erfolgreich durchgeführt. Somit kann die Abwärtskompatibilität der Änderungen als vollständig und korrekt angenommen werden. Das Ergebnis kann im Anhang unter A.7 eingesehen werden.

7 Komponententests

Die vorhandenen Komponententests (*engl.* Unit-Tests), die mittels JUnit[46] durchgeführt werden können, wurden an den notwendigen Stellen korrigiert und erweitert. Insbesondere wurden die Tests betreffend des strikteren Verhaltens einiger Bezeichnertypen angepasst, welches sich aus der IF-MAP 2.1 Spezifikation ergibt.

7.1 Anpassungen ifmapj

- **`DateHelpersTest`** - Tests zur Prüfung der korrekten Funktionsweise der Hilfsmethoden zur Datumserstellung und -konvertierung.

- **`AccessRequestHandlerTest`** - Die NULL-Prüfung bei Instanzerstellung eines neuen Objektes wurde dahingehend geändert, dass nun ein Objekt mit einem zufällig generierten Namen erstellt wird, anstatt NULL zurückzugeben.

- **`AccessRequestTest`** - Die Tests zur Erstellung eines AccessRequest mit UUID als Namen und einer Kombination aus Publisher ID und eines beliebigen Wertes wurden hinzugefügt.

- **`DeviceHandlerTest`** - Anpassungen an das geänderte Verhalten bei Übergabe eines NULL-Wertes an den Konstruktor.

- **`DeviceTest`** - Neuer Test zur Prüfung auf das korrekte Format eines UUID-Namens sowie eines Namens bestehend aus Publisher ID und eines beliebigen Wertes.

- **`IdentifiersTest`** - Zusätzliche Tests zur Prüfung der korrekten Erstellung eines erweiterten Bezeichners.

- **`IpAddressHandlerTest`** - Erweiterung der Tests um mehrere Gültigkeitsprüfungen von IPv4 Adressen sowie zur Kontrolle des Verhaltens bei der Umwandlung von Adressen in das spezifizierte Format.

- **`IpAddressTest`** - Anpassungen an den Test zur Erstellung eines Bezeichners mit einem ungültigen Wert als IP Adresse.

- **`MacAddressHandlerTest`** - Neue Tests zur Erstellung eines Bezeichners mit `byte[]`-Werten.

- **`MacAddressTest`** - Test zur kanonischen Schreibweise von MAC Adressen (Umwandlung in Kleinbuchstaben).

[46] `http://en.wikipedia.org/wiki/JUnit`

7.2 Anpassungen irond

- **EventProcessorSetup** - Die Instanziierung von `MetadataFactoryImpl` wurde um die Angabe einer Serverkonfiguration, zur Ermittlung der Einstellungen zur XML Validierung, angepasst.
- **DummydataModelConf** - Anpassungen für die neuen Konfigurationseinstellungen und ihrer Standardwerte.
- **IdentityTest** - Verwendung gültiger eindeutiger X.500 Namen in den jeweiligen Tests.
- **MetadataTest** - Aufruf von `MetaDataFactoryImpl` mit der Angabe einer Testkonfiguration des Servers erweitert.
- **StubProvider** - Erweiterung für die neuen Konfigurationseinstellungen mit entsprechenden Standardwerten.
- **PrefixNamespaceMapTest** - Aufruf von `MetaDataFactoryImpl` mit der Angabe einer Testkonfiguration des Servers erweitert.

8 Erweiterung der „ifmapj-examples“

Den ifmapj-examples wurden mehrere Beispiele zur Erstellung und Verwendung erweiterter Bezeichner sowie zur Zeitsynchronisation hinzugefügt.

Hierzu wurden die folgenden Klassen neu erstellt und in den Programmablauf integriert:

- **`ExtendedIdentityExample`** - Einfaches Beispiel zur Erstellung eines erweiterten Bezeichners.
- **`ExtendedIdentityExample2`** - Komplexes Beispiel mit herstellerspezifischen Metadaten zur Erstellung eines erweiterten Bezeichners.
- **`ClockSkewExample`** - Beispielhafte Vorgehensweise zum automatischen und manuellen Zeitabgleich mit einem MAP Server.

8.1 Klasse: ExtendedIdentityExample

Die Verwendung der Methoden zur Erstellung eines erweiterten Bezeichners wird in dieser Klasse anhand eines simplen Beispiels dargestellt. Hierbei wird eine aktive Verbindung zum MAP Server vorausgesetzt. Das XML Dokument des erweiterten Bezeichners wird als `InputStream` über die Funktion `getResourceAsStream()` bereitgestellt. Dazu ist die XML Datei als Resource dem `Package` vorher hinzugefügt worden.

Listing 20: Einfaches Beispiel zur Erstellung eines erweiterten Bezeichners unter Angabe einer XML Datei

```
1 StandardIfmapMetadataFactory mF =
2       IfmapJ.createStandardMetadataFactory();
3 Identifier extId = Identifiers.createExtendedIdentity(
4       getClass().getResourceAsStream(Config.EXTENDED_IDENTITY_XML));
```

8.2 Klasse: ExtendedIdentityExample2

Diesem Beispiel liegt ein klarer Anwendungsfall zugrunde. Es wird gezeigt, wie ein DHCP Server in einem Netzwerk Informationen über die vergebenen IP Adressen an den MAP Server veröffentlicht. Dazu werden vom DHCP Server zwei Arten von Bezeichnern erzeugt. Zuerst wird ein „$mac - address$“ Bezeichner angelegt und anschließend über Metadaten mit einem Bezeichner vom Typ „$ip - address$“ verknüpft. In den Metadaten werden Informationen zur Ablauf- und Vergabezeit vermerkt. Drei weitere Endgeräte werden so ebenfalls durch den DHCP Server im Graphen des MAP Server bekannt gemacht.

Ein Zusammenhang zwischen den vergebenen Adressen ist bisher jedoch nicht

nachvollziehbar gewesen. Als Lösung bieten sich erweiterte Bezeichner in Verbindung mit herstellerspezifischen Metadaten an. Diese können die einzelnen IP-Adressen, die alle zu einem bestimmten Subnetz gehören, miteinander verknüpfen.

Hierzu wird einfach ein erweiterter Bezeichner mit entsprechenden Informationen zu dem Subnetz veröffentlicht. Dies geschieht in dem durch herstellerspezifische Metadaten ein Zusammenhang zu allen vergebenen IP Adressen hergestellt wird. Diese Metadaten bestehen aus dem Element „$part - of$“ und dem Attribut „$link - status$“. Zusätzlich werden über „$location$“-Metadaten Informationen zum Standort des Netzwerkes mit dem Bezeichner verknüpft.

Als erweitertes Szenario wäre denkbar, dass der Status durch einen Switch aktualisiert wird, sobald die physikalische Verbindung getrennt wurde. Die IP Adresse bleibt weiterhin, bis zur Ablaufzeit, für das Gerät reserviert und kann somit bis dahin im Graphen bestehen bleiben. Nach Ablauf der Gültigkeit kann der DHCP Server die Informationen über die Adresse vom MAP Server löschen.

Sinnvoll ist dies in der Praxis zur Unterstützung administrativer Tätigkeiten in großen, über mehrere Standorte verteilten, Netzwerken. Es können Probleme im Netzwerk schneller und einfacher festgestellt werden. Denkbar wäre hier auch eine Netzwerküberwachung anhand der Daten eines MAP Servers, bei dem Nagios[47] die Zusammenhänge der Netzwerkkomponenten aus dem Graphen übernehmen kann.

8.3 Klasse: ClockSkewExample

In diesem Beispiel werden beide Möglichkeiten zur Erkennung von Zeitabweichungen verwendet. Es gibt einerseits die Möglichkeit, die Synchronisation der Bibliothek zu überlassen oder aber die Bibliothek nur hilfsweise zur Erkennung der Zeitabweichung zu verwenden. Beide Mechanismen werden in der Klasse exemplarisch behandelt. Zuerst wird der automatische Abgleich, im Anschluss der manuelle Abgleich durchgeführt.

8.3.1 Automatische Zeitsynchronisation

Der automatische Mechanismus zum Zeitabgleich mit einem MAP Server wird initiiert indem bei der Erstellung des SSRC Kanals ein Gerätebezeichner an den Methodenaufruf `createSSRC()` übergeben wird. Anschließend wird überprüft, ob die Zeit erfolgreich synchronisiert wurde und der zeitliche Versatz wird ausgegeben. Nach der Synchronisation steht ein Objekt vom Typ `Calendar` bereit, welches auf die Serverzeit eingestellt ist.

Nun wird mit einer Unterbrechung von jeweils einer Sekunde die Serverzeit erneut ausgegeben. So lässt sich erkennen, dass die Zeit weiterläuft und das `Calendar`-Objekt anstatt der lokalen Systemzeit genutzt werden kann.

[47] `http://www.nagios.org/`

Jetzt wird die Sitzung durch einen Aufruf von `renewSession()` erneuert und die Zeit dabei ein weiteres Mal im Hintergrund synchronisiert. Der Zeitpunkt der letzten Synchronisation wird von der ifmapj-Bibliothek aktualisiert und zur Kontrolle auf der Konsole erneut ausgegeben.

Listing 21: Nutzung der Bibliothek zur automatischen Zeitsynchronisation

```
1 SSRCImpl ssrc = IfmapJ.createSSRC(Config.BASIC_AUTH_SERVER_URL, Config
      .BASIC_AUTH_USER, Config.BASIC_AUTH_PASSWORD,
2       IfmapJHelper.getTrustManagers(getClass().getResourceAsStream(
            Config.TRUST_STORE_PATH), Config.TRUST_STORE_PASSWORD),
            Identifiers.createDev());
3 Calendar serverTime = ssrc.getClockFromServer();
4 Integer clockSkew = ssrc.getClockSkewSeconds();
```

8.3.2 Manueller Zeitabgleich

Wenn der Zeitabgleich nicht bei Erstellung der Verbindung durchgeführt werden soll, da beispielsweise noch kein Gerätebezeichner für den Client selbst bekannt ist, kann die Funktion `performClockSkewDetection()` unter Verwendung eines Gerätebezeichners für den MAP Client im späteren Verlauf auch noch manuell aufgerufen werden.

Die Funktion `getClockSynchronized()` zur Überprüfung der erfolgreichen Synchronisation findet beim manuellen Abgleich keine Anwendung. Daher ist die Rückgabe der Funktion hier immer `false`. Statt einem `Calendar`-Objekt steht jetzt nur der Zeitversatz in Millisekunden zur Verfügung. Dieser kann beliebig weiter verarbeitet werden. Zuletzt wird in dem Beispiel eine fiktive externe Zeitquelle genutzt um die Synchronisation durchzuführen. Hierbei wird die externe Zeit in Form eines `Calendar`-Objektes an die Funktion übergeben.

Listing 22: Manuelle Erkennung einer Zeitabweichung zum MAP Server

```
1 SSRCImpl ssrc = IfmapJ.createSSRC(Config.BASIC_AUTH_SERVER_URL, Config
      .BASIC_AUTH_USER, Config.BASIC_AUTH_PASSWORD,
2       IfmapJHelper.getTrustManagers(getClass().getResourceAsStream(
            Config.TRUST_STORE_PATH), Config.TRUST_STORE_PASSWORD));
3 long clockSkew = ssrc.performClockSkewDetection(Identifiers.createDev
      ());
```

9 Bewertung

9.1 Erreichte Ziele

Die notwendigen Umsetzungen zur Erreichung der vollständigen Kompatibilität zur IF-MAP Spezifikation in der aktuellen Version 2.1 sind vollzogen. Hierzu wurden in der Analyse die neuen Spezifikationen untersucht und die Änderungen, die sich daraus ergeben, herausgearbeitet. In der Konzepterstellung wurde geprüft, wie sich die bestehenden Softwarekomponenten an die geänderte Spezifikation anpassen lassen und gleichzeitig abwärtskompatibel bleiben, um die Nutzung der neuen Bibliothek so einfach wie möglich zu gestalten.

Letztlich wurden in der Implementierung neue Programmfunktionen angelegt und bestehende sinnvoll erweitert. Schwerpunkt der Umsetzungen waren zum einen die erweiterten Bezeichner und zum anderen die Zeitsynchronisation. Jedoch wurden auch etliche bestehende Funktionen erweitert und die Nutzung der ifmapj-Bibliothek insgesamt vereinfacht. Hierzu wurden neue Funktionsaufrufe in der ifmapj-Bibliothek erstellt sowie Teile des MAP Server irond erweitert. Auch das Visualisierungswerkzeug irongui wurde um die Funktionalität zur Nutzung der erweiterten Bezeichner erweitert.

9.2 Erfüllte Anforderungen

Die Anforderungen an die Implementierung wurden vollständig erfüllt. Dabei wurde auf die Stabilität (abfangen von Ausnahmen) geachtet, die gute Wartbarkeit erhalten (klare Klassenstruktur), die Erweiterbarkeit sichergestellt (Kapselung von Implementationen) und die Dokumentation umfassend (beinhaltet dieses Dokument sowie die Javadoc-Kommentare) verfasst.

Die Abwärtskompatibilität wurde insbesondere in der ifmapj-Bibliothek durch das Überladen von Funktionen und Konstruktoren sichergestellt. Funktionsaufrufe, die nach der aktuellen Spezifikation nicht mehr zulässig sind, wurden nicht entfernt, sondern als veraltet markiert. Um nicht zu sehr ins Detail zu gehen, wurden diese Umstände bei der Implementierung nur teilweise erwähnt.

An einigen Stellen wurden im Quellcode noch veraltete Kommentare entdeckt. Diese sind von mir sinnvoll angepasst oder ggf. sogar entfernt worden, wenn ihr Inhalt nicht mehr nötig war.

Der Quelltext der jeweiligen Programme ist unter der gängigen Open-Source Lizenz `Apache 2.0` verfügbar.

10 Ausblick

Ein Punkt der aktuellen Spezifikation wurde in dieser Arbeit nicht berücksichtigt. Dabei geht es um die Anforderung Suchanfragen unter Umständen um einige Ergebnisse zu zensieren, da der Client nicht die notwendigen Rechte zum Abrufen der Informationen besitzt. Grund hierfür ist die aufwendige Implementation dieses Punktes, da umfangreiche Vorarbeiten geleistet werden müssen. Im Detail bedeutet das, dass ein komplexes, idealerweise rollen-basiertes, Berechtigungssystem erschaffen wird. Allein der Aufwand hierfür ist groß genug, um daraus eine eigene Bachelor- oder sogar Masterarbeit zu erstellen.

Sofern ein sinnvoller Anwendungsfall gefunden wird und der Aufwand gerechtfertigt ist, kann zukünftig auch die Implementation zur Schema-Validierung in die ifmapj-Bibliothek aufgenommen werden.

Die IF-MAP Spezifikation ist weiterhin $work\ in\ progress$, also nach wie vor Änderungen und Korrekturen unterworfen. Einen Hinweis hierzu findet man bereits in der Spezifikation bei den erweiterten Bezeichnern. Dort wird erwähnt, dass es sich bei der aktuellen Umsetzung nur um eine Übergangslösung zur Wahrung der Kompatibilität handelt.

Neue Anwendungsfälle können jederzeit Bestandteil der Spezifikation werden und eine erneute Überarbeitung der hier untersuchten Softwarekomponenten notwendig machen.

Der durchgeführte Nachweistest lag leider bis zum Abgabetermin dieser Arbeit nur für die bisherige IF-MAP Version 2.0 vor. Es wäre durchaus sinnvoll, sobald der überarbeitete Test für IF-MAP 2.1 erscheint, diesen gleich durchzuführen und nötigenfalls weitere Anpassungen zur Einhaltung der Spezifikationen vorzunehmen. Interessant wäre auch zu sehen, wie sich die durchgeführten Änderungen in der Interoperabilität bei einem der nächsten IF-MAP Plugfests auswirken.

A Anhang

A.1 Glossar

Kürzel	Beschreibung
ARC	Asynchronous receive channel
FHH / HS	Hochschule Hannover
IFMAP	Interface Metadata Access Point Protocol
JAXB - RI	Java Architecture for XML Binding Reference Implementation
JRE	Java Runtime Environment
JSSE	Java Secure Socket Extension
MAP	Metadata Access Point
MAPC	Metadata Access Point Client (MAP Client)
MAPS	Metadata Access Point Server (MAP Server)
PDP	Policy Decision Point
SSRC	Synchronous send-receive channel
TNC	Trusted Network Connect
TNG	Trusted Network Group
URI	Uniform Resource Identifier
WSDL	Webservice Description Language
XML	Extensible Markup Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformation

A.2 Verwendete Software

Bei der Erstellung dieser Bachelorarbeit wurden die folgenden Programme verwendet:

- Java SE 7u5 - `http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1637583.html`
- Java SE 6u33 - `http://www.oracle.com/technetwork/java/javase/downloads/jdk6-downloads-1637591.html`
- Netbeans 7.1.2 - `http://netbeans.org/`
- MikTEX Basic 2.9 - `http://miktex.org/`
- IF-MAP 2.0 Compliance Test Suite (r3) - TCG CONFIDENTIAL `http://www.trustedcomputinggroup.org/`

A.3 Verwendeter Quellcode

- irond 0.3.4 - http://trust.inform.fh-hannover.de/joomla/index.php/downloads
- ifmapj 0.1.4 - http://trust.inform.fh-hannover.de/joomla/index.php/downloads
- ifmapj-examples 0.1.4 - http://trust.inform.fh-hannover.de/joomla/index.php/downloads
- irongui 0.4.0 - http://trust.inform.fh-hannover.de/joomla/index.php/downloads

Der Quellcode wurde nicht in der aktuell angebotenen Version von der Webseite heruntergeladen sondern in der jeweils letzten (Trunk-)Fassung über den SVN-Server trust.inform.fh-hannover.de abgerufen.

A.4 Inhalt des Datenträgers

Bestandteil dieser Arbeit ist eine CD-ROM, die den Quellcode enthält, welcher ebenfalls über den SVN Server trust.inform.fh-hannover.de abrufbar ist. Auf der CD-ROM sind neben dem Quellcode auch die zugehörige JavaDoc Dokumentation, kompilierte Binärdateien für die Java JRE und die benötigten XML Schemata enthalten.

A.5 Übersicht neuer Funktionen und Einstellungen

In dieser Übersicht befinden sich alle Funktionsaufrufe, die im Rahmen dieser Bachelorarbeit modifiziert oder neu angelegt worden sind.

A.5.1 ifmapj 0.1.5

Die folgenden Funktionsaufrufe sind in der ifmapj-Bibliothek Version 0.1.5 neu hinzugekommen oder wurden weiter überladen:

- **IfmapJ** *public SSRC* createSSRC()
  Erstellen eines sicheren Kommunikationskanals zum MAP Server.
- **IfmapStrings** *public class* IfmapStrings()
  Erweiterung der Konstanten um benötigte Werte.
- **SsrcImpl** *public class* SsrcImpl()
  Implementierung der Serverkommunikation.

- **SsrcImpl** *public void* performClockSynchronization()
  Ausführen eines Zeitabgleichs durch Veröffentlichung und Abruf der eigenen Zeit.
- **SsrcImpl** *public long* performClockSkewDetection()
  Einen zeitlichen Versatz zum Server feststellen.
- **SsrcImpl** *public boolean* getClockSynchronized()
  Wurde die Uhrzeit mit dem Server synchronisiert?
- **SsrcImpl** *public long* getClockSkewMilliseconds()
  Ausgabe des Zeitversatzes zum Server in Millisekunden.
- **SsrcImpl** *public Integer* getClockSkewSeconds()
  Ausgabe des Zeitversatzes zum Server in Sekunden.
- **SsrcImpl** *public Calendar* getClockFromServer()
  Zeit und Datum des Servers.
- **SsrcImpl** *public Calendar* getClockLastSynchronization()
  Zeitpunkt der letzten Zeitsynchronisation.
- **SsrcImpl** *private void* publishClientTime()
  Veröffentlicht Metadaten mit dem eigenen Zeitstempel.
- **SsrcImpl** *private Element* searchClientTime()
  Sucht nach selbst veröffentlichten Zeitstempel-Metadaten.
- **AccessRequest** *public AccessRequest* AccessRequest()
  Konstruktor zur Erstellung eines AccessRequest-Bezeichners.
- **IdentifierFactoryImpl** *public Identity* createIdentity()
  Erstellung eines Identitätsbezeichners.
- **Identifiers** *public Identity* createExtendedIdentity()
  Erstellung eines erweiterten Bezeichners gekapselt in einem Identitätsbezeichner.
- **Identifiers** *public Identity* createOtherIdentity()
  Erstellung eines Identitätsbezeichners vom Typ other.
- **Identity** *public Identity* equals()
  Vergleichsfunktion für beliebige Identitäten.
- **Identity** *private boolean* extendedIdentifierEquals()
  Prüffunktion zum Vergleich eines erweiterten Bezeichners mit einem Anderen.

- **Identity** *private boolean* distinguishedNameEquals()
  Prüffunktion zum Vergleich von eindeutigen X.500 Namensidentitäten.
- **StandardIfmapMetadataFactoryImpl** *public Document* create()
  Erstellung von herstellerspezifischen Metadaten mit den angegebenen Attributen.
- **DomHelpers** *public String* prepareExtendedIdentifier()
  Vorbereitung des XML eines erweiterten Bezeichners zur Übertragung an den Server.
- **DomHelpers** *protected String* removeNamespaceAndHeader()
  Entfernen des Namensraums eines XML Dokumentes.
- **DomHelpers** *public Document* toDocument()
  Konvertierung einer Zeichenkette in ein XML Dokument.
- **DomHelpers** *public boolean* compare()
  Funktion zum Vergleich von zwei XML Dokumenten.
- **DomHelpers** *private String* convertStreamToString()
  Umwandlung der Daten eines `StreamSource`-Objektes in eine Zeichenkette.
- **DateHelpers** *public String* getUtcTimeAsIso8601()
  Konvertierung eines `Calendar`-Objektes in eine ISO-8601 konforme Zeichenkette.
- **DateHelpers** *public Calendar* getTimeFromIso8601String()
  Konvertierung einer ISO-8601 konformen Zeichenkette in ein Objekt vom Typ `Calendar`.
- **MD5Provider** *public String* getMD5()
  Berechnung der MD5-Prüfsumme einer Zeichenkette.
- **StringHelpers** *public int* getStringCharCount()
  Berechnung der Länge einer Zeichenkette.
- **StringHelpers** *public int* fromByte()
  Umwandlung eines `byte[]`-Array in eine Zeichenkette.
- **StringHelpers** *public int* getRandomString()
  Ausgabe einer zufälligen Zeichenkette mit bestimmter Länge.

A.5.2 irond 0.3.5

Die folgenden Funktionsaufrufe sind im MAP Server irond 0.3.5 neu hinzugekommen oder wurden weiter überladen:

- **JaxbResultMarshaller** *private void* setResponseValidationAttribute()
  Hinzufügen des Validierungsattributs für Metadaten und Basisdaten.
- **JaxbResultMarshaller** *private void* createSearchResponse()
  Erweiterung um Angabe des Validierungsattributs.
- **JaxbResultMarshaller** *private void* createPollResponse()
  Erweiterung um Angabe des Validierungsattributs.
- **ClientIdentifier** *public String* getIdentificationString()
  Erzeugen einer eindeutigen Identifikationszeichenkette für einen Client.
- **ClientIdentifier** *private int* calculateHashCode()
  Berechnung des Hashcodes für eine Instanz von `ClientIdentifier`.
- **ClientIdentifier** *public boolean* equals()
  Vergleichsfunktion zweier Instanzen von `ClientIdentifier`.
- **CertificateChannelAuth** *public void* authenticate()
  Übergabe alle notwendigen Zertifikatsinformationen an die Klasse `ClientIdentifier`.
- **ChannelAcceptor** *public void* run()
  DEBUG-Ausgabe erweitert um Informationen zu TCP Keep-Alive und Socket Timeout.
- **AccessRequest** *protected int* getHashCode()
  Berechnung des Hashcodes für eine Instanz von `AccessRequest`.
- **Device** *protected int* getHashCode()
  Berechnung des Hashcodes für eine Instanz von `Device`.
- **IdentifierImpl** *protected int* getHashCode()
  Berechnung des Hashcodes für eine Instanz von `IdentifierImpl`.
- **IdentifierImpl** *public int* hashCode()
  Ausgabe des Hashcodes für eine Instanz von `IdentifierImpl`.
- **IdentifierWithAdministrativeDomain** *protected int* getHashCode()
  Berechnung des Hashcodes für eine Instanz von `IdentifierWithAdministrativeDomain`.
- **Identity** *public constructor* Identity()
  Um Validierungen und Normalisierungen bestimmter Identitätstypen erweitert.
- **Identity** *public boolean* equals()
  Anpassungen der Vergleichsfunktion an die speziellen Bedingungen für erweiterte Bezeichner und eindeutige X.500 Namen.

- **Identity** *protected int* getHashCode()
  Berechnung das Hashcodes für eine Instanz von `Identity`.
- **Identity** *private Document* prepareExtendedIdentifier()
  Vorbereiten und normalisieren eines erweiterten Bezeichners zum Vergleich mit anderen Instanzen.
- **Identity** *private X500Principal* prepareDistinguishedName()
  Vorbereiten und normalisieren eines eindeutigen X.500 Namens zum Vergleich mit anderen Instanzen.
- **IpAddress** *public constructor* IpAddress()
  Um Validierungsfunktionen erweitertet zur Prüfung auf das korrekte Format einer Adresse.
- **IpAddress** *protected int* getHashCode()
  Berechnung des Hashcodes für eine Instanz von `IpAddress`.
- **MacAddress** *protected int* getHashCode()
  Berechnung des Hashcodes für eine Instanz von `MacAddress`.
- **Metadata** *public constructor* Metadata()
  Erweitert um die Angabe des Status der XML-Validierung von Metadaten.
- **MetadataFactoryImpl** *public constructor* MetadataFactoryImpl()
  Umfassende Ergänzungen zur Schema-Validierung (inkl. abgeschottetem Modus).
- **PollResultImpl** *public boolean* hasMetadataAndOnlyValidatedMetadata()
  Prüft ob im Ergebnis Metadaten enthalten sind ob diese alle schema-validiert wurden.
- **SearchResultImpl** *public boolean* hasMetadataAndOnlyValidatedMetadata()
  Prüft ob im Ergebnis Metadaten enthalten sind ob diese alle schema-validiert wurden.
- **PublisherIdGeneratorImpl** *public String* generatePublisherIdFor()
  Erzeugen einer Publisher ID für einen mit dem MAP Server verbundenen Client.
- **ServerConfigurationProviderPropImpl** *public boolean* getXmlValidationMetadata()
  Konfigurationseinstellung zur Schema-Validierung von Metadaten.
- **ServerConfigurationProviderPropImpl** *public boolean* getXmlValidationExtendedIdentity()
  Konfigurationseinstellung zur Schema-Validierung von erweiterten Bezeichnern.

- **ServerConfigurationProviderPropImpl** *public boolean* getXmlValidationMetadataLockDownMode()
  Konfigurationseinstellung zum abgeschotteten Modus für die Validierung von Metadaten.

- **ServerConfigurationProviderPropImpl** *public boolean* getXmlValidationExtendedIdentityLockDownMode()
  Konfigurationseinstellung zum abgeschotteten Modus für die Validierung von erweiterten Bezeichnern.

- **ServerConfigurationProviderPropImpl** *private Map<String, String>* getMetadataSchemaFileNames()
  Bereitstellen aller konfigurierten XML Schemadateien für Metadaten.

- **ServerConfigurationProviderPropImpl** *private Map<String, String>* getExtendedIdentifierSchemaFileNames()
  Bereitstellen aller konfigurierten XML Schemadateien für erweiterte Bezeichner.

- **ServerConfigurationProviderPropImpl** *public StreamSource* getExtendedIdentitySchema()
  Ausgabe einer `StreamSoruce` zu einem bestimmten XML Schema für erweiterte Bezeichner.

- **ServerConfigurationProviderPropImpl** *public StreamSource* getMetadataSchema()
  Ausgabe einer `StreamSource` zu einem bestimmten XML Schema für Metadaten.

- **ServerConfigurationProviderPropImpl** *public boolean* getSocketKeepAlive()
  Ausgabe des Konfigurationsparameters zu TCP Keep-Alive.

- **ServerConfigurationProviderPropImpl** *public boolean* getSocketTimeout()
  Ausgabe des Konfigurationsparameters zu TCP Socket Timeout.

- **ServerConfigurationProviderPropImpl** *public boolean* getStrictDistinguishedName()
  Konfigurationseintrag zur Verwendung von strikten eindeutigen X.500 Namen.

- **ServerConfigurationProviderPropImpl** *public boolean* getStrictExtendedIdentity()
  Konfigurationseintrag zur Verwendung von strikten erweiterten Bezeichnern.

- **PropertiesReaderWriter** *public constructor*
  Nutzung der neuen `SortedProperties` Klasse zur Speicherung der Konfiguration in alphabetischer Reihenfolge.

- **DomHelpers** *public Class* DomHelpers()
  Klasse mit Hilfsfunktionen zur Bearbeitung von XML Dokumenten.

- **DomHelpers** *public DocumentBuilder* newDocumentBuilder()
  Rückgabe eines `DocumentBuilder`-Objektes.

- **DomHelpers** *public Document* toDocument()
  Erzeugt ein `org.w3c.dom.Document` aus einem `String`-Objekt.

- **DomHelpers** *public String* fromDocument()
  Erzeugt ein `String`-Objekt aus einem `org.w3c.dom.Document`.

- **DomHelpers** *public boolean* compare()
  Vergleicht zwei XML Dokumente.

- **DomHelpers** *public String* removeNamespace()
  Entfernt einen evtl. vorhandenen Namensraum von einem XML Dokument.

- **DomHelpers** *public String* escapeXml()
  Erzeugt XML Entitäten aus den Steuerzeichen für XML.

- **DomHelpers** *public String* unescapeXml()
  Erzeugt XML Steuersequenzen aus den vordefinierten XML Entitäten.

- **DomHelpers** *public int* calculateHashCode()
  Berechnung eines Hashcodes von XML Dokumenten.

- **DomHelpers** *private void* logResult()
  Funktion zur Aufzeichnung von Ereignissen von Informationen zur aufrufenden Klasse zur Unterstützung beim Debugging.

- **MD5Provider** *public MessageDigest* getInstance()
  Erstellt bzw. übergibt die Instanz eines `MessageDigest` Objekts.

- **MD5Provider** *public String* getMD5()
  Erzeugt eine MD5-Prüfsumme zur übergebenen Zeichenkette.

- **SortedProperties** *public Enumeration* keys()
  Erweiterung der intern Klasse `Properties` um die Sortierung der Konfigurationseinstellungen beim Speichern.

Funktionen, die nur einen Variablenwert zurückgeben, sog. get-Methoden, werden der Übersicht halber hier nicht mit aufgeführt.

Zusätzlich wurden, entsprechend der konzeptionellen Überlegungen, bei der Implementierung neue Konfigurationseinstellungen in der Datei `ifmap.properties` etabliert. Diese Liste bietet eine kurze Übersicht über die Funktionsweise der neuen Parameter:

- **irond.comm.socket.keep-alive** (Standard: false)
  Gibt an, ob die TCP Option SO_KEEPALIVE gesetzt werden soll.
- **irond.comm.socket.timeout** (Standard: 0 - unendlich)
  Angabe eines Timeout Wertes in Millisekunden für den Abbruch blockierender Übertragungen.
- **irond.ifmap.strict.distinguished-name.rewrite** (Standard: false)
  Speichert den Namen einer eindeutigen X.500 Namensidentität in normalisierter Form und gibt diese auch an Clients in Suchanfragen aus.
- **irond.ifmap.strict.identity.extended.lockdown** (Standard: false)
  Verweiget einen erweiterten Identifier sofern kein passendes XML Schema (XSD) dafür vorliegt bzw. konfiguriert wurde.
- **irond.ifmap.strict.identity.extended.rewrite** (Standard: false)
  Speichert eine erweiterte Identität in normalisierter Form und liefert diese bei Suchanfragen zurück.
- **irond.ifmap.strict.identity.extended.schema.file.<nr>** (kein Standard)
  Angabe einer XML Schemadatei zur Validierung erweiterter Identitäten. Der Parameter muss am Ende mit einer fortlaufenden Nummer angegeben werden. So können mehrere Schemata genutzt werden.
- **irond.ifmap.strict.identity.extended.schema.uri.<nr>** (kein Standard)
  Die Angabe erfolgt ebenfalls nummeriert. Hier wird der Namespace URI des jeweiligen XML Schemas angegeben. Dieser wird zum Abgleich im abgeschotteten Modus benötigt.
- **irond.ifmap.strict.identity.extended.schema.validate** (Standard: false)
  Gibt an, ob erweiterte Identitäten einer XML Validierung unterzogen werden sollen.
- **irond.ifmap.strict.metadata.lockdown** (Standard: false)
  Betreibt den Server in einem abgeschotteten Modus, bei dem nur Metadaten zugelassen werden, für die ein Schema vorliegt. Hierzu muss die Option irond.ifmap.strict.metadata.schema.validate ebenfalls auf `true` gesetzt werden.
- **irond.ifmap.strict.metadata.schema.file.<nr>** (kein Standard)
  Pfadangabe zu einer Schema-Datei zur Validierung von Metadaten. Auch hier muss die Angabe des Parameters nummeriert erfolgen.

- **irond.ifmap.strict.metadata.schema.uri.<nr>** (kein Standard)
  Korrespondierender Namespace URI zu den Schema-Dateien zur Validierung von Metadaten. Der Parameterschlüssel muss auch hier nummeriert angegeben werden.

- **irond.ifmap.strict.metadata.schema.validate** (Standard: false)
  Festlegung ob Metadaten einer XML Validierung unterzogen werden sollen.

Zur besseren Übersicht werden die Einträge in der Konfigurationsdatei jetzt aufsteigend alphabetisch sortiert. Dies erleichtert das Wiederfinden von Einträgen und verschafft einen besseren Überblick über die verfügbaren Parameter. Hierzu wurde die Klasse `SortedProperties` geschaffen, welche die bisher genutzte `Properties` Klasse erweitert.

Es wurden XML Schema Dateien hinzugefügt um einen Betrieb mit Schema-Validierung der spezifizierten Elemente zu ermöglichen. Zur Nutzung der Schema-Validierung erweiterter Bezeichner ist ein Beispielschema (siehe A.6.2) enthalten.

A.5.3 irongui 0.4.1

Die folgenden Funktionsaufrufe sind in dem Visualisierungswerkzeug irongui 0.4.1 neu hinzugekommen oder wurden weiter überladen:

- **IfmapUtils** *public String* escapeXml()
  Funktion zur Ersetzung von XML Steuerzeichen durch ihre Entitäten zur Darstellung innerhalb von HTML.

- **IfmapUtils** *public String* unescapeXml()
  Funktion zur Umwandlung von XML Entitäten in XML Steuerzeichen.

- **ViewController** *public void* showSubscriptionDialog()
  Zeigt das Dialogfenster zum Abonnement von Metadaten an.

- **SubscriptionDialog** *public Class* SubscriptionDialog()
  Beinhaltet die Elemente des Dialogfensters zum Abonnement von Metadaten.

- **SubscriptionDialog** *public void* show()
  Zeigt das Dialogfenster zum Abonnement von Metadaten an.

- **SubscriptionDialog** *public void* hide()
  Versteckt das Dialogfenster zum Abonnement von Metadaten.

- **XmlLoader** *public Class* XmlLoader()
  Klasse zur Auswahl von XML Dokumenten per Dateiauswahldialog.

- **XmlLoader** *public String* loadXml()
  Funktion zum Laden eines XML Dokumentes aus einer Datei und Rückgabe als Zeichenkette.

A.6 Listings

A.6.1 Beispiel für erweiterte Bezeichner

Listing 23: Beispiel-XML eines erweiterten Bezeichners der ein Netzwerk darstellt

```
<?xml version="1.0" encoding="UTF-8"?>
<ns:network administrative-domain=""
  address="192.168.1.1"
  type="IPv4"
  netmask="255.255.255.0"
  xmlns:ns="http://trust.inform.fh-hannover.de/2012/NETWORK-IDENTITY-EXAMPLE/1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://trust.inform.fh-hannover.de/2012/NETWORK-IDENTITY-EXAMPLE/1
   network-identifier-2.1v1.xsd" />
```

A.6.2 Vorgabe XSD für erweiterte Bezeichner

Listing 24: Grundgerüst zur Erstellung eines erweiterten Bezeichners

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.
    trustedcomputinggroup.org/2012/IFMAP-IDENTIFIER/1" targetNamespace="http://www.
    trustedcomputinggroup.org/2012/IFMAP-IDENTIFIER/1">
  <xsd:complexType name="IdentifierType">
    <!-- All extended identifier types MUST extend the IdentifierType complexType as
        defined by this specification.
      If the administrative-domain is not used, it MUST be set to an empty string in
          an extended identifier instance.-->
    <xsd:attribute name="administrative-domain" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:schema>
```

A.6.3 Beispiel XSD Netzwerkbezeichner

Listing 25: Beispielhaftes XSD für einen erweiterten Bezeichner zur Angabe eines Netzwerkes

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:base-id="http://www.
    trustedcomputinggroup.org/2012/IFMAP-IDENTIFIER/1" xmlns="http://trust.inform.fh-
    hannover.de/NETWORK-IDENTITY" targetNamespace="http://trust.inform.fh-hannover.de
    /2012/NETWORK-IDENTITY-EXAMPLE/1">
  <xsd:import namespace="http://www.trustedcomputinggroup.org/2012/IFMAP-IDENTIFIER/1"
      schemaLocation="ifmap-identifier-2.1v1.xsd"/>
  <xsd:element name="network">
    <xsd:complexType>
      <xsd:complexContent>
        <xsd:extension base="base-id:IdentifierType">
          <xsd:attribute name="address" type="xsd:string" use="required"/>
```

```
9                <xsd:attribute name="netmask" type="xsd:string" use="required"/>
10               <xsd:attribute name="type" use="required">
11                  <xsd:simpleType>
12                     <xsd:restriction base="xsd:string">
13                        <xsd:enumeration value="IPv4"/>
14                        <xsd:enumeration value="IPv6"/>
15                     </xsd:restriction>
16                  </xsd:simpleType>
17               </xsd:attribute>
18            </xsd:extension>
19         </xsd:complexContent>
20      </xsd:complexType>
21   </xsd:element>
22 </xsd:schema>
```

A.6.4 Vorgabe XSD für operative Metadaten

Listing 26: XSD für die operativen Metadaten zur Bestimmung der Zeitabweichung zwischen Client und Server

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3    xmlns:ifmap="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
4    xmlns="http://www.trustedcomputinggroup.org/2012/IFMAP-OPERATIONAL-METADATA/1"
5    targetNamespace=
6     "http://www.trustedcomputinggroup.org/2012/IFMAP-OPERATIONAL-METADATA/1">
7    <xsd:import namespace="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
8      schemaLocation="ifmap-base-2.0v17.xsd"/>
9    <!-- a singleValue metadata published on a device identifier that is unique
10     and represents the MAP Client. MAP Clients SHOULD update the client-time
11     metadata with a lifetime of session. -->
12   <xsd:element name="client-time">
13     <xsd:complexType>
14       <xsd:attribute name="current-timestamp" type="xsd:dateTime" use="required"/>
15       <xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
16     </xsd:complexType>
17   </xsd:element>
18 </xsd:schema>
```

A.6.5 Beispiel XSD Netzwerkzugehörigkeit

Listing 27: Beispielhaftes XSD für Metadaten zur Verknüpfung von Endgeräten mit einem Netzwerk

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ifmap="http://www.
       trustedcomputinggroup.org/2010/IFMAP/2" xmlns="http://trust.inform.fh-hannover.de
       /2012/NETWORK-METADATA-EXAMPLE/1" targetNamespace="http://trust.inform.fh-hannover.
       de/2012/NETWORK-METADATA-EXAMPLE/1">
3    <xsd:import namespace="http://www.trustedcomputinggroup.org/2010/IFMAP/2"
         schemaLocation="ifmap-base-2.0v17.xsd"/>
4    <!-- Example XSD for an extended identifier which represents a network and allows to
         create
5         links between devices located on the same subnet for example as shown in ifmapj-
              examples -->
6    <xsd:element name="part-of">
```

```
      <xsd:complexType>
         <xsd:attribute name="dhcp-lease-time" type="xsd:string" />
         <xsd:attribute name="link-status" type="xsd:string" use="required" />
         <xsd:attributeGroup ref="ifmap:singleValueMetadataAttributes"/>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

A.6.6 XSLT zur Entfernung eines Namensraums

Listing 28: XSL Transformation zur Entfernung des Namensraums eines erweiterten Bezeichners

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="*">
    <xsl:element name="{local-name(.)}">
      <xsl:apply-templates select="@* | node()"/>
    </xsl:element>
  </xsl:template>
  <xsl:template match="@*">
    <xsl:attribute name="{local-name(.)}">
      <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:template>
</xsl:stylesheet>
```

A.7 Ergebnis des IF-MAP 2.0 Nachweistests

Listing 29: Ergebnis des IF-MAP 2.0 Nachweistests mit irond 0.3.5

```
09/15/12 18:17:20: All Tests Finished: 73454
09/15/12 18:17:20: SUCCESS (IFMSTC01_Sec->testSecAuthTLS2): 1004ms
09/15/12 18:17:20: SUCCESS (IFMSTC01_Sec->testSecNoAuth3): 27ms
09/15/12 18:17:20: SUCCESS (IFMSTC01_Sec->testSecNoTLS5): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC02_MetaData->testMetadata3): 103ms
09/15/12 18:17:20: SUCCESS (IFMSTC02_MetaData->testMetadata4): 27ms
09/15/12 18:17:20: SUCCESS (IFMSTC02_MetaData->testMetadata5): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC02_MetaData->testMetadata6): 14ms
09/15/12 18:17:20: SUCCESS (IFMSTC02_MetaData->testMetadata7): 18ms
09/15/12 18:17:20: SUCCESS (IFMSTC02_MetaData->testMetadata8): 13ms
09/15/12 18:17:20: SUCCESS (IFMSTC02_MetaData->testMetadata9): 23ms
09/15/12 18:17:20: SUCCESS (IFMSTC02_MetaData->testMetadata10): 13ms
09/15/12 18:17:20: SUCCESS (IFMSTC03_UTF8->testUnicode3): 14ms
09/15/12 18:17:20: SUCCESS (IFMSTC03_UTF8->testUnicode4): 290ms
09/15/12 18:17:20: SUCCESS (IFMSTC04_Ident->testIdent3): 41ms
09/15/12 18:17:20: SUCCESS (IFMSTC04_Ident->testIdent4): 96ms
09/15/12 18:17:20: SUCCESS (IFMSTC05_IdentSize->testIdentOptionsAccess3_Device): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC05_IdentSize->testIdentOptionsAccess4_Device): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC05_IdentSize->testIdentOptionsAccess5_Device): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC05_IdentSize->testIdentOptionsAccess6_Device): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC05_IdentSize->testIdentOptionsAccess7_Device): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC05_IdentSize->testIdentOptionsSearch8): 13ms
09/15/12 18:17:20: SUCCESS (IFMSTC05_IdentSize->testIdentOptionsSearch9): 14ms
09/15/12 18:17:20: SUCCESS (IFMSTC05_IdentSize->testIdentOptionsSearch10): 19ms
```

```
09/15/12 18:17:20: SUCCESS (IFMSTC05_IdentSize->testIdentOptionsSearch11): 13ms
09/15/12 18:17:20: SUCCESS (IFMSTC05_IdentSize->testIdentOptionsSearch12): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC06_AccessReq->testAccessReqEquiv3): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC06_AccessReq->testAccessReqEquiv4): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC06_AccessReq->testAccessReqEquiv6): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC06_AccessReq->testAccessReqEquiv7): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC06_AccessReq->testAccessReqEquiv8): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom3): 14ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom4): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom5): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom6): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom7): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom8): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom9): 13ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom10): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom11): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom12): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom13): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom14): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom15): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom16): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom17): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom18): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom19): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom20): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom21): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom22): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom23): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom24): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom25): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC07_AdminDom->testDom26): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC08_DeviceIdent->testDeviceEquiv3): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC08_DeviceIdent->testIdentEquiv4): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC08_DeviceIdent->testDeviceEquiv5): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC08_DeviceIdent->testDeviceEquiv6): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC08_DeviceIdent->testDeviceEquiv7): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC08_DeviceIdent->testDeviceEquiv8): 0ms
09/15/12 18:17:20: SUCCESS (IFMSTC09_IdentIdent->testIdentReqEquiv3): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC09_IdentIdent->testIdentReqEquiv4): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC09_IdentIdent->testIdentReqEquiv5): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC09_IdentIdent->testIdentReqEquiv6): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC09_IdentIdent->testIdentReqEquiv7): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC09_IdentIdent->testIdentReqEquiv8): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC10_IdentName->testIdentEnums): 115ms
09/15/12 18:17:20: SUCCESS (IFMSTC11_HIPHIT->testValidHit3): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC11_HIPHIT->testValidHit4): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC11_HIPHIT->testValidHit5): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC11_HIPHIT->testInvalidHit6): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC11_HIPHIT->testInvalidHit7): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC11_HIPHIT->testInvalidHit8): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC11_HIPHIT->testInvalidHit9): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testValidIP3): 19ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP4a): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP4b): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP4c): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP4d): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP4e): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP4f): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP4g): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testValidIP5): 20ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP6a): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP6b): 8ms
```

```
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP6c): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP6d): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP6e): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP6f): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP6g): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP6h): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC12_IdentIP->testInvalidIP6i): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC13_AdminDomCase->testIPDomain3): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC13_AdminDomCase->testIPDomain4): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC13_AdminDomCase->testIPDomain5): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC13_AdminDomCase->testIPDomain6): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC13_AdminDomCase->testMacDomain7): 13ms
09/15/12 18:17:20: SUCCESS (IFMSTC13_AdminDomCase->testMacDomain8): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC13_AdminDomCase->testMacDomain9): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC13_AdminDomCase->testMacDomain10): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC14_Mac->testValidMAC3): 18ms
09/15/12 18:17:20: SUCCESS (IFMSTC14_Mac->testInvalidMAC4a): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC14_Mac->testInvalidMAC4b): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC14_Mac->testInvalidMAC4c): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC14_Mac->testInvalidMAC4d): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC14_Mac->testInvalidMAC4e): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC14_Mac->testInvalidMAC4f): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC14_Mac->testInvalidMAC4g): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC14_Mac->testInvalidMAC4h): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC14_Mac->testInvalidMAC4i): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC15_Search->testSearch3): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC15_Search->testSearch4): 19ms
09/15/12 18:17:20: SUCCESS (IFMSTC16_AttrPrefix->testAttrPrefix3): 97ms
09/15/12 18:17:20: SUCCESS (IFMSTC16_AttrPrefix->testAttrPrefix4): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC17_PubIdUniq->testPubId4): 19ms
09/15/12 18:17:20: SUCCESS (IFMSTC17_PubIdUniq->testPubId6): 17ms
09/15/12 18:17:20: SUCCESS (IFMSTC17_PubIdUniq->testPubId8): 23ms
09/15/12 18:17:20: SUCCESS (IFMSTC17_PubIdUniq->testPubId10): 0ms
09/15/12 18:17:20: SUCCESS (IFMSTC19_InvalidCard->testInvalidCardinality3): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC19_InvalidCard->testInvalidCardinality4): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC19_InvalidCard->testInvalidCardinality5): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC19_InvalidCard->testInvalidCardinality6): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC19_InvalidCard->testInvalidCardinality7): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC19_InvalidCard->testInvalidCardinality8): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC20_Lifetime->testLifetime3): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC20_Lifetime->testLifetime5): 25ms
09/15/12 18:17:20: SUCCESS (IFMSTC20_Lifetime->testLifetime6): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC20_Lifetime->testLifetime7): 7ms
09/15/12 18:17:20: SUCCESS (IFMSTC20_Lifetime->testLifetime8): 19ms
09/15/12 18:17:20: SUCCESS (IFMSTC20_Lifetime->testLifetime9): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC20_Lifetime->testLifetime10): 7ms
09/15/12 18:17:20: SUCCESS (IFMSTC20_Lifetime->testLifetime11): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC21_NotifyLifetime->testNotifyLifetime3): 411ms
09/15/12 18:17:20: SUCCESS (IFMSTC21_NotifyLifetime->testNotifyLifetime4): 38ms
09/15/12 18:17:20: SUCCESS (IFMSTC21_NotifyLifetime->testNotifyLifetime5): 19ms
09/15/12 18:17:20: SUCCESS (IFMSTC21_NotifyLifetime->testNotifyLifetime6): 201ms
09/15/12 18:17:20: SUCCESS (IFMSTC21_NotifyLifetime->testNotifyLifetime7): 17ms
09/15/12 18:17:20: SUCCESS (IFMSTC21_NotifyLifetime->testNotifyLifetime8): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC22_DeleteFilter->testDeleteFilter3): 17ms
09/15/12 18:17:20: SUCCESS (IFMSTC22_DeleteFilter->testDeleteFilter4): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC22_DeleteFilter->testDeleteFilter5): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC22_DeleteFilter->testDeleteFilter6): 13ms
09/15/12 18:17:20: SUCCESS (IFMSTC22_DeleteFilter->testDeleteFilter7): 7ms
09/15/12 18:17:20: SUCCESS (IFMSTC23_Purge->testPurgeStart3): 19ms
09/15/12 18:17:20: SUCCESS (IFMSTC23_Purge->testPurgeSub4): 209ms
09/15/12 18:17:20: SUCCESS (IFMSTC23_Purge->testPurge5): 238ms
09/15/12 18:17:20: SUCCESS (IFMSTC24_MetaSize->testMetaSize3): 231ms
```

```
09/15/12 18:17:20: SUCCESS (IFMSTC24_MetaSize->testMetaSize4): 273ms
09/15/12 18:17:20: SUCCESS (IFMSTC24_MetaSize->testMetaSize5): 575ms
09/15/12 18:17:20: SUCCESS (IFMSTC25_SearchFilter->testSearchFilter3): 48ms
09/15/12 18:17:20: SUCCESS (IFMSTC25_SearchFilter->testSearchFilter4): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC25_SearchFilter->testSearchFilter5): 209ms
09/15/12 18:17:20: SUCCESS (IFMSTC26_FilterNS->testFilterNS3): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC26_FilterNS->testFilterNS4): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC26_FilterNS->testFilterNS5): 209ms
09/15/12 18:17:20: SUCCESS (IFMSTC26_FilterNS->testFilterNS6): 24ms
09/15/12 18:17:20: SUCCESS (IFMSTC27_MultiOpFail->testMultiOp3): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC27_MultiOpFail->testMultiOp4): 408ms
09/15/12 18:17:20: SUCCESS (IFMSTC27_MultiOpFail->testMultiOp5): 37ms
09/15/12 18:17:20: SUCCESS (IFMSTC27_MultiOpFail->testMultiOp6): 1042ms
09/15/12 18:17:20: SUCCESS (IFMSTC28_MultiPub->testMultiPub3): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC28_MultiPub->testMultiPub4): 408ms
09/15/12 18:17:20: SUCCESS (IFMSTC28_MultiPub->testMultiPub5): 42ms
09/15/12 18:17:20: SUCCESS (IFMSTC28_MultiPub->testMultiPub6): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC29_SvCard->testSvCard3): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC29_SvCard->testSvCard4): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC29_SvCard->testSvCard5): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC29_SvCard->testSvCard6): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC29_SvCard->testSvCard7): 13ms
09/15/12 18:17:20: SUCCESS (IFMSTC29_SvCard->testSvCard8): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC30_MvCard->testMvCard3): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC30_MvCard->testMvCard4): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC30_MvCard->testMvCard5): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC30_MvCard->testMvCard6): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC30_MvCard->testMvCard7): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC30_MvCard->testMvCard8): 13ms
09/15/12 18:17:20: SUCCESS (IFMSTC31_NotifyPoll->testNotifyPoll3): 409ms
09/15/12 18:17:20: SUCCESS (IFMSTC31_NotifyPoll->testNotifyPoll4): 38ms
09/15/12 18:17:20: SUCCESS (IFMSTC31_NotifyPoll->testNotifyPoll5): 202ms
09/15/12 18:17:20: SUCCESS (IFMSTC31_NotifyPoll->testNotifyPoll6): 30ms
09/15/12 18:17:20: SUCCESS (IFMSTC31_NotifyPoll->testNotifyPoll7a): 1001ms
09/15/12 18:17:20: SUCCESS (IFMSTC31_NotifyPoll->testNotifyPoll7): 24ms
09/15/12 18:17:20: SUCCESS (IFMSTC32_PubDelete->testDelete3): 18ms
09/15/12 18:17:20: SUCCESS (IFMSTC32_PubDelete->testDelete4): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC32_PubDelete->testDelete5): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC32_PubDelete->testDelete6): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC33_PubAtomic->testPubAtomic3): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC33_PubAtomic->testPubAtomic4): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC33_PubAtomic->testPubAtomic5): 88ms
09/15/12 18:17:20: SUCCESS (IFMSTC33_PubAtomic->testPubAtomic6): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC34_MegaSearch->testSearch3): 45ms
09/15/12 18:17:20: SUCCESS (IFMSTC34_MegaSearch->testSearch4): 77ms
09/15/12 18:17:20: SUCCESS (IFMSTC34_MegaSearch->testSearch5): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC35_DefMaxDepth->testDefMaxDepth3): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC35_DefMaxDepth->testDefMaxDepth4): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC36_DefSearchSize->testSearchSize3): 228ms
09/15/12 18:17:20: SUCCESS (IFMSTC36_DefSearchSize->testSearchSize4): 267ms
09/15/12 18:17:20: SUCCESS (IFMSTC36_DefSearchSize->testSearchSize5): 70ms
09/15/12 18:17:20: SUCCESS (IFMSTC36_DefSearchSize->testSearchSize6): 15ms
09/15/12 18:17:20: SUCCESS (IFMSTC37_MaxSearchSize->testMaxSearch3): 229ms
09/15/12 18:17:20: SUCCESS (IFMSTC37_MaxSearchSize->testMaxSearch4): 7ms
09/15/12 18:17:20: SUCCESS (IFMSTC38_SearchLink->testSearchLink3): 18ms
09/15/12 18:17:20: SUCCESS (IFMSTC38_SearchLink->testSearchLink4): 15ms
09/15/12 18:17:20: SUCCESS (IFMSTC39_SearchNoMeta->testSearchNoMeta3): 18ms
09/15/12 18:17:20: SUCCESS (IFMSTC39_SearchNoMeta->testSearchNoMeta4): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC40_SearchIdent->testSearchNoMeta3): 38ms
09/15/12 18:17:20: SUCCESS (IFMSTC41_SingleSub->testSingSub3): 207ms
09/15/12 18:17:20: SUCCESS (IFMSTC41_SingleSub->testSingSub4): 421ms
09/15/12 18:17:20: SUCCESS (IFMSTC41_SingleSub->testSingSub5): 55ms
```

```
09/15/12 18:17:20: SUCCESS (IFMSTC41_SingleSub->testSingSub6): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC41_SingleSub->testSingSub7): 1219ms
09/15/12 18:17:20: SUCCESS (IFMSTC42_InvalidSub->testInvalidSub3): 210ms
09/15/12 18:17:20: SUCCESS (IFMSTC42_InvalidSub->testInvalidSub4): 20ms
09/15/12 18:17:20: SUCCESS (IFMSTC43_InvalidPoll->testInvalidPoll3): 209ms
09/15/12 18:17:20: SUCCESS (IFMSTC43_InvalidPoll->testInvalidPoll4): 20ms
09/15/12 18:17:20: SUCCESS (IFMSTC44_InvalidPollRemove->testInvPollRem3): 408ms
09/15/12 18:17:20: SUCCESS (IFMSTC44_InvalidPollRemove->testInvPollRem4): 37ms
09/15/12 18:17:20: SUCCESS (IFMSTC44_InvalidPollRemove->testInvPollRem5): 0ms
09/15/12 18:17:20: SUCCESS (IFMSTC44_InvalidPollRemove->testInvPollRem6): 1201ms
09/15/12 18:17:20: SUCCESS (IFMSTC45_PollRes->testPollRes3): 18ms
09/15/12 18:17:20: SUCCESS (IFMSTC45_PollRes->testPollRes5_6): 408ms
09/15/12 18:17:20: SUCCESS (IFMSTC45_PollRes->testPollRes6): 26ms
09/15/12 18:17:20: SUCCESS (IFMSTC45_PollRes->testPollRes7): 1ms
09/15/12 18:17:20: SUCCESS (IFMSTC46_SecondPoll->testSecondPoll3): 209ms
09/15/12 18:17:20: SUCCESS (IFMSTC46_SecondPoll->testSecondPoll4): 29ms
09/15/12 18:17:20: SUCCESS (IFMSTC46_SecondPoll->testSecondPoll5): 1038ms
09/15/12 18:17:20: SUCCESS (IFMSTC46_SecondPoll->testSecondPoll6): 201ms
09/15/12 18:17:20: SUCCESS (IFMSTC46_SecondPoll->testSecondPoll7): 21ms
09/15/12 18:17:20: SUCCESS (IFMSTC46_SecondPoll->testSecondPoll8): 1023ms
09/15/12 18:17:20: SUCCESS (IFMSTC46_SecondPoll->testSecondPoll9): 1026ms
09/15/12 18:17:20: SUCCESS (IFMSTC46_SecondPoll->testSecondPoll10): 201ms
09/15/12 18:17:20: SUCCESS (IFMSTC47_MultiSub->testMultiSub3): 29ms
09/15/12 18:17:20: SUCCESS (IFMSTC47_MultiSub->testMultiSub4): 208ms
09/15/12 18:17:20: SUCCESS (IFMSTC47_MultiSub->testMultiSub5): 255ms
09/15/12 18:17:20: SUCCESS (IFMSTC47_MultiSub->testMultiSub5a): 4ms
09/15/12 18:17:20: SUCCESS (IFMSTC47_MultiSub->testMultiSub6): 222ms
09/15/12 18:17:20: SUCCESS (IFMSTC47_MultiSub->testMultiSub6a): 0ms
09/15/12 18:17:20: SUCCESS (IFMSTC48_NotifyQueue->testNotifyQueue3): 209ms
09/15/12 18:17:20: SUCCESS (IFMSTC48_NotifyQueue->testNotifyQueue4): 15ms
09/15/12 18:17:20: SUCCESS (IFMSTC48_NotifyQueue->testNotifyQueue5): 200ms
09/15/12 18:17:20: SUCCESS (IFMSTC49_NotifyOnly->testNotifyOnly3): 410ms
09/15/12 18:17:20: SUCCESS (IFMSTC49_NotifyOnly->testNotifyOnly4): 27ms
09/15/12 18:17:20: SUCCESS (IFMSTC49_NotifyOnly->testNotifyOnly5): 12ms
09/15/12 18:17:20: SUCCESS (IFMSTC50_ClientBuffer->testClientBuffer3): 207ms
09/15/12 18:17:20: SUCCESS (IFMSTC50_ClientBuffer->testClientBuffer4): 9686ms
09/15/12 18:17:20: SUCCESS (IFMSTC50_ClientBuffer->testClientBuffer5): 8455ms
09/15/12 18:17:20: SUCCESS (IFMSTC50_ClientBuffer->testClientBuffer6): 0ms
09/15/12 18:17:20: SUCCESS (IFMSTC50_ClientBuffer->testClientBuffer7): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC50_ClientBuffer->testClientBuffer8): 29059ms
09/15/12 18:17:20: SUCCESS (IFMSTC50_ClientBuffer->testClientBuffer9): 208ms
09/15/12 18:17:20: SUCCESS (IFMSTC51_PurgeResp->testPurgeResp3): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC51_PurgeResp->testPurgeResp4): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC51_PurgeResp->testPurgeResp5): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC51_PurgeResp->testPurgeResp6): 6ms
09/15/12 18:17:20: SUCCESS (IFMSTC51_PurgeResp->testPurgeResp7): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC51_PurgeResp->testPurgeResp8): 6ms
09/15/12 18:17:20: SUCCESS (IFMSTC51_PurgeResp->testPurgeResp9): 7ms
09/15/12 18:17:20: SUCCESS (IFMSTC52_DeleteLifetime->testDeleteLifetime2): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC52_DeleteLifetime->testDeleteLifetime3): 11ms
09/15/12 18:17:20: SUCCESS (IFMSTC52_DeleteLifetime->testDeleteLifetime5): 208ms
09/15/12 18:17:20: SUCCESS (IFMSTC52_DeleteLifetime->testDeleteLifetime6): 233ms
09/15/12 18:17:20: SUCCESS (IFMSTC53_SSRCPoll->testSSRCPoll3): 6ms
09/15/12 18:17:20: SUCCESS (IFMSTC53_SSRCPoll->testSSRCPoll4): 6ms
09/15/12 18:17:20: SUCCESS (IFMSTC54_ARCAllow->testArcAllow2a): 207ms
09/15/12 18:17:20: SUCCESS (IFMSTC54_ARCAllow->testArcAllow3): 22ms
09/15/12 18:17:20: SUCCESS (IFMSTC54_ARCAllow->testArcAllow4): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC54_ARCAllow->testArcAllow5): 9ms
09/15/12 18:17:20: SUCCESS (IFMSTC54_ARCAllow->testArcAllow6): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC54_ARCAllow->testArcAllow7): 5ms
09/15/12 18:17:20: SUCCESS (IFMSTC54_ARCAllow->testArcAllow8): 6ms
09/15/12 18:17:20: SUCCESS (IFMSTC55_HTTPComp->testHttpComp2): 17ms
```

```
09/15/12 18:17:20: SUCCESS (IFMSTC56_SessionMaxPoll->testNewSession2): 15ms
09/15/12 18:17:20: SUCCESS (IFMSTC57_NewSession->testNewSession2): 14ms
09/15/12 18:17:20: SUCCESS (IFMSTC58_MultiSession->testMultiSes3): 408ms
09/15/12 18:17:20: SUCCESS (IFMSTC58_MultiSession->testMultiSes4): 21ms
09/15/12 18:17:20: SUCCESS (IFMSTC58_MultiSession->testMultiSes5): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC58_MultiSession->testMultiSes3a): 0ms
09/15/12 18:17:20: SUCCESS (IFMSTC59_VerifySessionId->testVerifySess3): 7ms
09/15/12 18:17:20: SUCCESS (IFMSTC59_VerifySessionId->testVerifySess4): 7ms
09/15/12 18:17:20: SUCCESS (IFMSTC59_VerifySessionId->testVerifySess5): 6ms
09/15/12 18:17:20: SUCCESS (IFMSTC59_VerifySessionId->testVerifySess6): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC59_VerifySessionId->testVerifySess7): 19ms
09/15/12 18:17:20: SUCCESS (IFMSTC59_VerifySessionId->testVerifySess8): 5ms
09/15/12 18:17:20: SUCCESS (IFMSTC59_VerifySessionId->testVerifySess9): 5ms
09/15/12 18:17:20: SUCCESS (IFMSTC60_InvalidSessionId->testInvalidSess3): 7ms
09/15/12 18:17:20: SUCCESS (IFMSTC60_InvalidSessionId->testInvalidSess4): 14ms
09/15/12 18:17:20: SUCCESS (IFMSTC60_InvalidSessionId->testInvalidSess5): 7ms
09/15/12 18:17:20: SUCCESS (IFMSTC61_SessionEnd->testSessionEnd3): 6ms
09/15/12 18:17:20: SUCCESS (IFMSTC61_SessionEnd->testSessionEnd4): 6ms
09/15/12 18:17:20: SUCCESS (IFMSTC62_DuplicateARC->testDuplicateArc3): 407ms
09/15/12 18:17:20: SUCCESS (IFMSTC62_DuplicateARC->testDuplicateArc4): 1054ms
09/15/12 18:17:20: SUCCESS (IFMSTC62_DuplicateARC->testDuplicateArc4a): 0ms
09/15/12 18:17:20: SUCCESS (IFMSTC62_DuplicateARC->testDuplicateArc5): 8ms
09/15/12 18:17:20: SUCCESS (IFMSTC63_TCPSSLERR_REMOVED->testSSLERR3): 0ms
09/15/12 18:17:20: SUCCESS (IFMSTC64_ReadOnly->testReadOnly3): 7ms
09/15/12 18:17:20: SUCCESS (IFMSTC64_ReadOnly->testReadOnly4): 5ms
09/15/12 18:17:20: SUCCESS (IFMSTC64_ReadOnly->testReadOnly5): 6ms
09/15/12 18:17:20: SUCCESS (IFMSTC65_MultiConn->testMulti3): 10ms
09/15/12 18:17:20: SUCCESS (IFMSTC65_MultiConn->testMulti4): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC65_MultiConn->testMulti5): 19ms
09/15/12 18:17:20: SUCCESS (IFMSTC65_MultiConn->testMulti6): 16ms
09/15/12 18:17:20: SUCCESS (IFMSTC65_MultiConn->testMulti7): 5ms
09/15/12 18:17:20: SUCCESS (IFMSTC65_MultiConn->testMulti8): 6ms
```

A.8 Tabellarische Kapitelübersicht

Thema	IF-MAP	Analyse	Konzept	Implem.	IFMAPJ	IROND
Bedingungen zur Nutzung einer administrativen Domäne	3.2.1	3.1.1	4.1.1	5.1	X	X
„Device“ und „AccessRequest“ Bezeichner	3.2.2	3.1.2	4.1.2	5.2	X	X
Regeln zum Vergleich von Bezeichnern	3.2.2.3	3.1.3	4.1.3	5.3	X	X
Gleichheitsbedingungen von „distinguished-name“ Identitäten	3.2.2.3.1	3.1.4	4.1.4	5.4	X	X
Administrativen Domäne bei „mac-address“ Bezeichnern	3.2.2.5	3.1.1				
Neu eingeführte erweiterte Bezeichner	3.2.3	3.1.5	4.1.5	5.5	X	X
Normalisieren von Metadaten	3.3	3.2.1	4.2.1			
Beziehung eines MAP Client zu einer Publisher ID	3.3.1	3.2.2	4.2.2	5.6	-	X
Kardinalität von Metadaten	3.3.3	3.2.3	4.2.3			
Größe von Metadaten	3.3.6	3.2.4				
Neue operative Metadaten	3.3.7	3.2.5	4.2.5	5.10	X	-
XML Schemavalidierung	3.5	3.3	4.3	5.8	-	X
Fehlermeldungen	3.6.1	3.4				
Nutzung von „match-links“	3.7.2.3	3.5				
Bedingungen für den Suchalgorithmus	3.7.2.8	3.6				
Herstellerspezifische Metadaten	3.9	3.2.6	4.2.6	5.7	X	-
Behandlung von Sitzungen	4.1.1	3.7	4.4	5.9	-	X
Zeitsynchronisation	4.6	3.8	4.5	5.10	X	-

In den Spalten ist jeweils das korrespondierende Kapitel aus der IF-MAP 2.1 Spezifikation und die Abschnitte aus den jeweiligen Kapiteln Analyse, Konzept und Implementierung in dieser Arbeit angegeben. Außerdem ist vermerkt ob die Änderungen den MAP Server irond und/oder die ifmapj-Bibliothek betreffen.

A.9 Literaturverzeichnis

[1] Trusted Computing Group. Overview of Trusted Network Connect IF-MAP. http://www.trustedcomputinggroup.org/files/resource_files/AA9CF85D-1D09-3519-ADC31E1BEA407646/TNC_IF-MAP%20Overview%2004-2009.pdf, April 2009.

[2] Trusted Computing Group. Trusted Network Connect IF-MAP 2.1 Specification FAQ. http://www.trustedcomputinggroup.org/files/resource_files/288A0B24-1A4B-B294-D0B92B9F49507E12/TNC%20IF-MAP%202_1%20FAQ.pdf, Mai 2012.

[3] Trusted Computing Group. Trusted Network Connect IF-MAP binding for SOAP specification version 2.1. http://www.trustedcomputinggroup.org/files/resource_files/2888CAD9-1A4B-B294-D0ED95712B121FEF/TNC_IFMAP_v2_1r15.pdf, Rev. 15 Mai 2012.

[4] Trusted Computing Group. Trusted Network Connect IF-MAP binding for SOAP specification version 2.0. http://www.trustedcomputinggroup.org/files/resource_files/93C23F2A-1A4B-B294-D0A36CCD8A9E7BD3/TNC_IFMAP_v2_0R0_47.pdf, Rev. 47 November 2011.