Universität Rostock

Traditio et Innovatio

Bachelorarbeit

Konzipierung und Implementierung eines Entwicklungstools zur automatischen RESTful Discovery in CoAP-Netzwerken.

eingereicht am 21. Oktober 2013 von:

Simeon Wiedenmann
Martikel-Nr.: 209 207 232

Gutachter:
Vlado Altmann &
Prof. Dirk Timmermann

Universität Rostock Traditio et Innovatio

Fakultät für Informatik und Elektrotechnik
Institut für Angewandte Mikroelektronik und Datentechnik

Thema für eine Bachelorarbeit

für Herrn Simeon Wiedenmann

Konzipierung und Implementierung eines Entwicklungstools zur automatischen RESTful Discovery in CoAP-Netzwerken.

Neue intelligente Geräte, die in der Gebäudeautomatisierung zum Einsatz kommen, verbessern nicht nur den Komfort in einer Immobilie, sondern ermöglichen auch ein Energiemanagement. Durch Einsatz moderner, vernetzter und intelligenter Geräte und Systeme in der Gebäudeautomatisierung lassen sich die Betriebs- und Instandhaltungskosten senken. Für die Kommunikation der Geräte sind geeignete Schnittstellen notwendig. Diese sollen die Gerätekomposition sowie die Interoperabilität zwischen Geräten verschiedenen Herstellern gewährleisten. Ein Problem der Gebäudeautomationssysteme ist die aufwändige Installation der Geräte, bei der meist technisches Fachwissen erforderlich ist. Die Schnittstellen sollen daher über Plug&Play-Funktionalitäten verfügen, um eine einfache Konfiguration und eine vollautomatische Kopplung der Geräte zu ermöglichen. Mit Hilfe des CoAP-Protokolls ist es möglich, die Geräte mittels Plug&Play miteinander zu koppeln. Um die Entwicklung und das Testen der CoAP-Geräte zu vereinfachen, ist ein geeignetes Tool zu entwerfen. Das Tool soll in der Lage sein, automatisiert ein Device Discovery durchzuführen und dabei einem Benutzer die Schnittstellen zu präsentieren.

In dieser Bachelorarbeit sollen zuerst die Prinzipien von Resource Discovery in CoAP-Netzwerken erläutert werden. Darüber hinaus soll die Web Application Description Language (WADL) auf die Eignung für eine detaillierte Beschreibung der CoAP-Schnittstellen analysiert werden. Es soll ein Tool entwickelt werden, welches RESTful Discovery-Mechanismen für CoAP umsetzt. Die gesammelten Discovery-Daten sollen für den Benutzer in einer geeigneten Form dargestellt werden.

Zu den wesentlichen Fragen, die geklärt werden sollen, gehören:

- Einarbeitung in die Themenbereiche RESTful Discovery, CoAP, WADL.
- Ausarbeitung und Implementierung eines Software-Tools für CoAP Discovery.
- Funktionale Validierung der entwickelten Software und Auswertung der ermittelten Ergebnisse.
- Ausführliche Dokumentation aller Arbeitsschritte.

Betreuer: M.Sc. Vlado Altmann
Prof. Dr. Dirk Timmermann
Tag der Ausgabe: 01.07.2013
Tag der Abgabe: 21.10.2013

Prof. Dr. D. Timmermann
Betreuender Hochschullehrer

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACK	Acknowledgement
CoAP	Constrained Application Protocol
CON	Confirmable
CoRE	Constrained RESTful Environments
DNS	Domain-Name-System
GUI	Graphical User Interface
HATEOAS	Hypermedia as the Engine of Application State
HTTP	Hypertext Transfer Protocol
IF	Interface Description
IP	Internet Protokoll
ISO	International Organization for Standartization
JDK	Java Developer Kit
M2M	Machine to Machine
MTU	Maximum Transmission Unit
NON	Non-Confirmable
OSI	Open System Interconnection
RDF	Resource Description Framework
REST	Representational State Transfer
RST	Reset
RT	Resource Typ
UDP	User Datagramm Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
WADL	Web Application Description Language
WS4D	Web Services for Devices
WWW	World Wide Web
XML	Extensible Markup Language

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Im Informationszeitalter des 21. Jahrhunderts ist das Internet nicht mehr aus unserem Alltag wegzudenken. Selbst einfachste Gegenstände wie Lichtschalter und Lampen erhalten eine sogenannte Internetprotokoll-Adresse (IP-Adresse) und werden Teil einer zunehmend vernetzten Welt.
Für das Gebiet der Gebäudeautomation bietet solch eine umfassende Vernetzung von Gegenständen, Geräten und Systemen großes Potential. Durch das Zusammenspiel von Geräten und Gegenständen lässt sich der Komfort in Gebäuden steigern und mittels intelligentem Energiemanagement ist es möglich, Betriebs- und Instandhaltungskosten zu senken. Eine Studie der Siemens Aktiengesellschaft kommt zu dem Ergebnis, dass rund 40 Prozent des weltweiten Energieverbrauches durch Gebäude verursacht werden, [4] und deutet damit die Einsparungsmöglichkeiten in diesem Feld an.
Herkömmliche Gebäudeautomations-Systeme sind häufig proprietär. Dies erzeugt unfreiwillige Abhängigkeiten der Kunden gegenüber den Herstellern, sorgt für überhöhte Kosten und verhindert Interoperabilität zwischen Geräten verschiedener Hersteller. Es wäre wünschenswert, durch Open Source Produkte und offene Standards den Markt freier und flexibler zu gestalten. Nicht selten kommt es vor, dass verschiedene Geräte unterschiedlicher Hersteller, die zu einem intelligenten Netzwerk zusammen geschaltet werden sollen, in Gebäuden bereits vorhanden sind. Auch ändert sich die Gerätekomposition in Gebäudenetzwerken häufig dynamisch durch mobile Geräte. Eine manuelle Installation solcher Systeme ist kompliziert, fehleranfällig und benötigt teures Fachpersonal. Daher ist es ein erstrebenswertes Ziel, offene Kommunikationsstandards zu manifestieren und die Installation zu automatisieren.
Damit sich die einzelnen Geräte vollautomatisch miteinander zu intelligenten Netzwerken zusammenschließen können, müssen sie einander erkennen und miteinander kommunizieren können. Das Auffinden vorhandener Ressourcen im lokalen Umfeld sowie die Dokumentation ihrer Handhabung wird als Resource-Discovery bezeichnet.
Wird das Konzept des „Internet of Things“ in die Gebäudeautomation mit einbezogen, so werden viele einfache Geräte mit geringem Speicher und Energievorrat an solche intelligenten Netzwerke angeschlossen. Für solche Geräte ermöglicht das Constrained Application Protocol (CoAP) Machine-to-Machine-Kommunikation (M2M) über das Internet. Durch standardisierte, offene Kommunikationsschnittstellen wird diese Kommunikation auch für Gerätekompositionen verschiedener Hersteller ermöglicht.
Um die Entwicklungen auf diesem Anwendungsgebiet der Gebäudeautomationstechnik voranzutreiben, entsteht im Rahmen dieser Arbeit ein Open Source Entwickler-Tool, welches automatische Geräteerkundung durchführt und erkundete Schnittstellen grafisch

präsentiert. Das Ziel dieser Software ist die Unterstützung von Entwicklern beim Testen und Entwickeln zukünftiger CoAP-Geräte.
Die Entwicklung, Validierung und Auswertung eines solchen Tools für RESTful-Discovery-Mechanismen im CoAP-Protokoll stellen den Kern dieser Arbeit dar. Zudem werden die theoretischen Grundlagen des RESTful-Resource-Discovery in CoAP-Netzwerken erläutert und analysiert, inwiefern sich die Web Application Description Language (WADL) für detaillierte Beschreibung der CoAP-Schnittstellen eignet. Am Ende der Entwicklung steht das Ziel, vollautomatische Ressourcenerkundung per Plug&Play-Prinzip zu realisieren, welche im Umfeld der Gebäudeautomation maschinenverständliche, einheitliche Schnittstellen vorhandener Ressourcen zusammenträgt.

2. RESTful Discovery

Im Folgenden werden wichtige theoretische Grundkenntnisse behandelt und wesentliche Begrifflichkeiten eingeführt, auf welche sich der praktische Teil der hier vorliegenden Arbeit bezieht. Dazu wird die Formulierung „RESTful Discovery“ betrachtet. RESTful Discovery bezeichnet die Erkundung von Ressourcen[1] in einem REST-konformen Umfeld. Anschließend folgen eine Einführung in ein Protokoll zur Implementierung des RESTful-Discovery sowie ein Theorieteil zu einer Beschreibungssprache für Webservices. Zunächst jedoch sei eine Einführung in den REST-Architekturstil geboten, welcher eine wichtige Basis für die weitere Arbeit darstellt.

2.1. Der REST-Architekturstil

Das Akronym REST steht für „REpresentational State Transfer“. Es bezeichnet einen stark skalierenden Architekturstil für verteilte Hypermedia-Informationssysteme [2], [5]. Die Bezeichnung stammt aus der Dissertation Thomas R. Fieldings, „Architectural Styles and the Design of Network-based Software Architectures“, in der Fielding den Erfolg des Web formal zu erklären versucht. Als Satz von Designkriterien ist REST an kein konkretes Protokoll gebunden [2, vgl. S. 86], jedoch ist der Ausdruck REST häufig in einem Atemzug mit den Begriffen Hypertext Transfer Protokoll (HTTP) und Uniform Resource Identifier (URI) zu hören, da der wichtigste Vertreter des REST-Architektur-Stils, das Web, HTTP als Protokoll und URIs als Identifikatoren für Ressourcen nutzt. Der REST-Architekturstil ist optimiert für den Hauptanwendungsfall im Web: den Transport großer Hypermedia-Daten [2, vgl. S. 82]. Er eignet sich zudem sehr gut für einfache Ad-hoc Integrations-Szenarien [5, S. 1], wie sie z. B. im Feld der Gebäudeautomation zu finden sind. Durch die einfache Art, mit der sich RESTful-Webservices bedienen und entwickeln lassen, erfreuen sich diese zunehmend an Beliebtheit. Die für REST nötige Infrastruktur existiert bereits und ist weit verbreitet [5, vgl. S. 1]. Ein handelsüblicher Browser auf der Client-Seite reicht bereits aus, um REST-Services in Anspruch zu nehmen.

2.1.1. Die REST-Designkriterien

Fielding leitet in seiner Dissertation REST von verschiedenen Architekturstilen ab und führt sechs wesentliche Designkriterien als Auflagen für RESTful Design an. Zunächst

[1] Resource-Discovery siehe Kapitel 2.2

wird die Trennung von Angelegenheiten der Datenspeicherung und der Nutzerschnittstelle angeführt. Eine solche **Client-Server**-Struktur erlaubt Plattform-übergreifende Portabilität der Nutzerschnittstellen, vereinfachte Serverkomponenten, unabhängige Entwicklungen auf beiden Seiten sowie eine höhere Skalierbarkeit der Dienste [2, vgl. S. 78].
Darüber hinaus sind alle RESTful Interaktionen **zustandslos** [5, S. 3]. Das bedeutet, dass jede Anfrage alle Informationen beinhaltet, die zu ihrer Bearbeitung nötig sind. Jede Anfrage kann daher vollständig unabhängig von vorangegangenen Anfragen bearbeitet werden [2, vgl. S. 78 f.]. Dies erhöht Wartbarkeit, Verlässlichkeit und Skalierbarkeit, da Anfragen individuell nachvollziehbar sind, partielle Fehler leicht behandelt werden können [6, vgl. S. 3] und Service-Provider ihre Ressourcen flexibel verwalten können. Anfragen-übergreifendes Ressourcenmanagement ist nicht nötig, jedoch wirkt sich zusätzlicher Overhead in den Anfragen negativ auf die Netzwerk-Performance aus.
Eine Möglichkeit, die Netzwerkeffizienz zu erhöhen bietet **Caching**. Server können bestimmte Antworten auf Anfragen als zwischenspeicherbar markieren. So kann ein Client die bereits gespeicherte Antwort auf vorhergehende, identische Anfragen erneut verwenden, anstatt gleiche Anfragen mehrfach zu stellen. Caching entlastet das Netzwerk und steigert die User-Performance und Skalierbarkeit des Dienstes. Der Preis hierfür ist das Risiko, nicht immer die aktuellsten Daten zu verwenden, falls sich der zwischengespeicherte Datensatz auf der Serverseite bereits verändert hat [2, vgl. S. 79 f.].
Ein sehr zentrales Designkriterium für REST stellt die einheitliche Schnittstelle, das **Uniforme Interface**, dar. Jedes Objekt in einer RESTful-Anwendung reagiert auf ein und dieselbe bekannte Schnittstelle [7, vgl. S. 14]. So werden konkrete Implementierungen von den durch sie angebotenen Services entkoppelt und eine unabhängige Entwicklung dieser Komponenten gestattet. Das ermöglicht einen einheitlichen Zugriff auf diese Services von unterschiedlichsten Endgeräten aus, steigert Benutzerfreundlichkeit durch Wiedererkennung und vereinfacht die Systemarchitektur zugunsten genereller Komponenten [8, vgl. S. 29 f.]. Die Allgemeingültigkeit einer solchen Schnittstelle kommt Hand in Hand mit dem Trade-off eines weniger effizienten Informationstransportes, welcher in diesem Fall standardisiert und nicht für einzelne Spezialfälle optimiert stattfindet [2, vgl. S. 82].
Um den Skalierungsanforderungen des modernen Internets gerecht zu werden bietet die REST-Architektur ein **Layered System** an. Einzelne Verarbeitungsebenen können spezielle Aufgaben optimieren und so dazu beitragen, die Skalierbarkeit zu erhöhen und die Systemkomplexität gering zu halten. Zwar führen hierarchische Zwischenschichten zu mehr Overhead und Latenz in der Informationsverarbeitung, jedoch kann durch Lastenverteilung, sogenanntes Load Balancing, und intelligente Caching-Ebenen dennoch eine bedeutende Performanceverbesserung erzielt werden [2, vgl. S. 82 ff.].
Die REST-Architektur basiert zunächst auf sehr einfachen Client Komponenten. Mittels **Code on Demand** besteht darüber hinaus die Möglichkeit, Clients durch downloadbare Applets und Scripte um spezielle Funktionalitäten zu erweitern [2, vgl. S. 84 f.].

2.1.2. Die REST-Daten-Elemente

Ressourcen, deren Identifikatoren und Repräsentationen von Ressourcen sind zentrale Elemente der REST-Architektur. Alles, was wichtig genug ist, um als Sache an sich referenziert zu werden, ist eine Ressource [7, vgl. S. 83]. Fielding bezeichnet eine Ressource als Abstraktion einer Information und führt an, dass jedes Konzept, das Ziel einer Hypertext-Referenz sein könnte, der Definition einer Ressource gerecht werden müsse [2, vgl. S. 88]. Im Feld der Gebäudeautomation könnte z. B. eine Liste mit allen verfügbaren Sensoren eines Raumes eine Ressource darstellen. Jeder Sensor auf der Liste könnte zudem selbst eine Ressource darstellen.
Um Ressourcen eindeutig identifizieren zu können, werden Resource-Identifier benötigt. Das sind Zeichensequenzen, welche eine Bezeichnung und Adresse für abstrakte oder physikalische Ressourcen darstellen und diese dadurch identifizieren [9, vgl. S. 3]. Auf diese Art werden Ressourcen adressierbar, können zwischen Anwendungen ausgetauscht und durch Vernetzung einem breiten Feld zugänglich gemacht werden.
Zur Darstellung der Information einer REST-Ressource werden Repräsentationen verwendet. Fielding bezeichnet Repräsentationen als Sequenz von Bytes, erweitert um Repräsentations-Metadaten, welche diese Bytes beschreiben [2, vgl. S. 90]. Typischerweise besitzt eine Ressource mehrere Repräsentationen. Das ermöglicht es, viele Informationsquellen zusammenzufassen, unabhängig von deren Implementierung. Die im obigen Beispiel erwähnte Liste verfügbarer Sensoren eines Raumes könnte z. B. als menschenlesbare Textdatei und als maschinenverarbeitbare Extensible Markup Language (XML) Datei vorliegen. Je nach Möglichkeiten und Vorlieben des Empfängers sowie der Art der Ressource ist durch sogenanntes „Content Negotiation“ vom Client eine bestimmte Repräsentation einer Ressource auswählbar [2, vgl. S. 89 ff.]. Das Datenformat einer Repräsentation wird als Media Type bezeichnet und ein gemeinsames Verständnis der verwendeten Datentypen zwischen Serviceprovider und Client wird durch Metadaten erzielt. Das sind Name-Wert-Paare, bei denen sich der Name auf einen Standard bezieht, welcher Struktur und Semantik des Wertes beschreibt [2, vgl. S. 91].
Ressourcen stehen in Beziehung zueinander und verweisen durch ihre Repräsentationen aufeinander. Die Beispielliste aller verfügbarer Ressourcen in einem Raum könnte daher auf einzelne Sensoren direkt verweisen, indem sie die jeweiligen Resource-Identifier der einzelnen Sensoren mit aufführt. Ressourcen, die über verschiedene Repräsentationen verfügen und durch eindeutige Identifikatoren aufeinander verweisen, bilden somit das für die REST-Architektur zentrale Hypermedia-Netz.
Die Kommunikation zwischen REST-Komponenten in diesem Netz erfolgt durch die Übertragung ausgewählter Repräsentationen konkreter Ressourcen [2, vgl. S. 87].
Die Bedeutung einer Nachricht zwischen Komponenten wird durch Kontrolldaten bestimmt. Hiermit lässt sich z. B. ausdrücken, welche Aktion angefragt wurde oder wie die Antwort auf eine Anfrage zu verstehen ist. Auch parametrisierte Anfragen lassen sich so realisieren [2, vgl. S. 91].
Die Tabelle 2.1 fasst die REST Datenelemente zusammen und fügt konkrete Beispiele

aus dem Modernen Web für die einzelnen Elemente an.

Daten Element	Beispiele des modernen Web
Ressource	beabsichtigtes konzeptionelles Ziel einer Hypertext-Referenz
Resource-Identifier	URL[1], URN[2]
Repräsentation	HTML Dokument, JPEG Bild
Repräsentations-Metadaten	Media Type, Zeitpunkt letzter Änderung
Ressourcen-Metadaten	Quellverweise, Ersatz, Veränderung
Kontrolldaten	ob geändert seit, Cache-Kontrolle

[1] Uniform Resource Locator
[2] Uniform Resource Name

Tabelle 2.1.: REST-Daten-Elemente [2, S. 88]

2.1.3. Das REST-Prinzip in Kürze

RESTful Serviceprovider bieten in verteilten Netzen Ressourcen an. Diese Ressourcen sind durch eindeutige Identifikatoren, die Resource-Identifier, adressierbar und verfügen über multimediale Repräsentationen, welche aufeinander verweisen. Über eine einheitliche Schnittstelle, das Uniforme Interface, lassen sich die Ressourcen durch festgelegte Operationen manipulieren. Die wesentlichen Grundoperationen sind das Erstellen neuer Ressourcen, das Lesen, das Aktualisieren sowie das Löschen von Ressourcen. Diese wesentlichen Operationen sind auf alle Ressourcen anwendbar. Sehr bekannt sind hier die Verben „PUT“, „GET“, „POST“ und „DELETE“ aus der HTTP-Implementierung [5, vgl. S. 3]. Serviceprovider und Client kommunizieren zustandslos mittels selbstbeschreibender Nachrichten durch den Austausch von Repräsentationen konkreter Ressourcen. Die Hypermedialität der Repräsentationen stellt dabei den Motor der Anwendung dar, was gerne als „Hypermedia as the Engine of Application State“ (HATEOAS) bezeichnet wird.

2.2. Resource-Discovery

Unter Resource-Discovery wird das Erkunden vorhandener Ressourcen in einem bestimmten Umfeld verstanden. Im Gebiet der Gebäudeautomation ist ein denkbares Anwendungsbeispiel hierfür die Erkundung alle Sensoren und Aktorenr eines unbekannten Raumes. Handelt es sich bei den zu erkundenden Ressourcen um solche, die durch Webserver angeboten werden, so wird auch von „Web Discovery“ gesprochen [10, vgl. S. 2].
Mobile Sensorknoten sorgen für zunehmend dynamische Netzwerke, auch im Gebiet der Gebäudeautomation. Durch Machine-to-Machine Kommunikation ist es möglich, dass

die einzelnen Komponenten in Automatisierungsnetzen auch ohne menschliche Interaktion miteinander kommunizieren können. Damit Clients in solchen Netzwerken die Ressourcen erkunden können, welche durch Service-Provider angeboten werden, sind Resource-Discovery-Mechanismen nötig [10, vgl. S. 3]. Deren Ziel ist das maschinenverständliche Dokumentieren der Handhabung vorhandener Ressourcen für den Client. Im wesentlichen beantworten solche Discovery-Mechanismen folgende Fragen:

- Welche Ressourcen werden in einem konkreten Umfeld angeboten?
- Wie ist eine konkrete Ressource adressierbar?
- Wie kann eine konkrete Ressource genutzt werden?

Kennt der Client die IP-Adresse eines Service-Providers, so kann eine Anfrage zur Resource-Discovery via Unicast direkt an die bekannte Adresse gestellt werden. Ist unbekannt welche Netzwerkteilnehmer Ressourcen anbieten, so bietet sich Multicast-Resource-Discovery an. Hierbei wird eine Anfrage zur Erkundung von Ressourcen an mehrere Netzwerkteilnehmer versendet. Service-Provider können daraufhin reagieren und die durch sich angebotenen Ressourcen beim anfragenden Client bekannt machen. Discovery-Mechanismen liefern dem Client Resource-Identifier, mit deren Hilfe die erkundeten Ressourcen adressierbar sind. Darüber hinaus werden verfügbare Attribute von Ressourcen und ggf. Verlinkungen auf weitere Ressourcen ermittelt. Der REST-Architekturansatz bietet wesentliche Elemente für die Erkundung von Ressourcen. In RESTful Services können Ressourcen durch Resource-Identifier, wie z. B. URIs, identifiziert werden. Die Resource-Identifier sind global eindeutig und spannen einen Adressraum auf, der für Resource-Discovery genutzt wird [5, vgl. S. 3]. Eine Liste aller von einem konkreten Service-Provider angebotener Ressourcen kann im REST-Architekturansatz selbst als Ressource aufgefasst werden und vom entsprechenden Service-Provider zur Ressourcenerkundung angeboten werden. Durch die hypermediale Vernetzung der Ressourcen ist es möglich, die Beziehung einzelner Ressourcen zueinander zu erkunden. Dies wird als „Web Linking“ bezeichnet [10, vgl. S. 2]. Durch Resource-Identifier und hypermediale Vernetzung ist es möglich, Resource-Discovery ohne eine zentrale Registrierungsstelle zu implementieren [5, vgl. S. 3]. Über welche Attribute eine Ressource verfügt und wie diese genutzt werden kann, wird durch selbsterklärende Nachrichten und Metadaten sowie zusätzliche Dokumentationen, z. B. durch Web Application Description Language Dateien, dokumentiert.
Zu den bekanntesten Resource-Discovery-Mechanismen in Automatisierungsnetzen gehören neben WADL auch das Resource Description Framework (RDF) sowie das Constrained RESTful Environments- (CoRE) Link-Format [11]. WADL gestattet eine ausführliche, maschinenverständliche Dokumentation angebotener Ressourcen und deren Handhabung. Dazu kann ein Service-Provider anfragenden Clients eine entsprechende WADL-Datei zur Verfügung stellen [12]. RDF stellt Informationen über Ressourcen in Form eines gerichteten Graphen dar. Ressourcen sowie konkrete Werte von Attributen

werden dabei als Knoten im RDF-Graph dargestellt und Beziehungen zwischen Ressourcen als Kanten modelliert [13]. In Sensornetzwerken mit leistungsbeschränkten Netzwerkknoten bietet das CoRE-Link-Format die Möglichkeit, Resource-Discovery durchzuführen. Dazu werden die hypermedialen Verknüpfungen der REST-Architektur ausgenutzt, indem von einem bekannten Einstiegspunkt aus weitere Ressourcen durch Verfolgen der Verlinkungen erkundet werden [10].

2.3. Das CoAP-Protokoll

Das Constrained Application Protocol ist ein Web Transfer Protokoll, welches für ressourcenlimitierte Netzwerke und Geräte mit sehr geringem Speicher spezialisiert ist. Es stellt die Schnittstelle zwischen Transport-Layer und Application-Layer im ISO / OSI Schitenmodell[2] dar. Die aktuelle CoAP-Spezifikation ist ein Internetdraft der CoRE Working Group mit dem Status „Work in Progress“ [1].
Das Protokoll ist speziell für M2M-Anwendungen designed und eignet sich daher sehr gut für das Feld der Gebäudeautomation. Es setzt den REST-Architekturstiel für limitierte Netzwerke um und bedient sich wichtiger Webtechnologien wie z. B. URIs und Internet Media Types [1, vgl S. 1].
CoAP basiert auf einem Request- / Response-Prinzip und bietet u.a. Service- und Resource-Discovery mittels des CoRE-Link-Formates an. Sogenannte „cross-protocol Proxys“ ermöglichen Interaktionen zwischen CoAP und HTTP [1, vgl. S. 14]. Im Gegensatz zu HTTP setzt CoAP auf verbindungslose Transportprotokolle, wie z. B. das User Datagramm Protocol (UDP).
Wesentliche Anforderungen an das Protokoll sind ein einfacher, asynchroner Nachrichtenaustausch mit sehr geringem Overhead und geringer Parsingkomplexität. Darüber hinaus unterstützt es Multicast-Anfragen, optionale Zuverlässigkeit für Datenaustausch sowie Caching und Proxy-Server [1, vgl. S. 5 f.].

2.3.1. CoAP – Aufbau und Funktionsweise

Das Request / Response Prinzip von CoAP basiert auf dem asynchronen, verbindungslosen Austausch von binär-codierten CoAP-Nachrichten zwischen Clients und Servern. CoAP-Client und CoAP-Server tauschen dabei je nach Anwendungsfall (siehe Tabelle 2.2) eine von vier Nachrichtentypen aus. Es existieren Confirmable- (CON), Non-Confirmable- (NON), Acknowlegement- (ACK) sowie Reset- (RST) Nachrichten. Confirmable-Nachrichten werden erneut gesendet, wenn Sie nicht innerhalb einer gewissen Zeit durch den Kommunikationspartner mit einer Acknowledgement-Nachricht bestätigt werden. Auf Non-Confirmable-Nachrichten muss die Gegenstelle nicht reagieren. Reset-

[2] International Organization for Standartization / Open System Interconnection Referenzmodell für Netzwerkprotokolle

	CON	NON	ACK	RST
Request	X	X	-	-
Response	X	X	X	-
Empty	X[1]	-	X	X

[1] Spezialfall CoAP ping

Tabelle 2.2.: Verwendung von CoAP-Nachrichten-Typen [1, vgl S. 23]

Nachrichten dienen dazu, auf erhaltene Nachrichten zu reagieren, die nicht behandelt werden können [1, vgl. S. 10 ff.].
Um eine Aktion auf einer Ressource durchzuführen, sendet ein CoAP-Client einen CoAP-Request an einen CoAP-Server. Dieser antwortet darauf z. B. mit der Repräsentation einer angefragten Ressource. CoAP kennt die aus HTTP bekannten Methoden GET zum Anfragen von Ressourcen, POST zum Bearbeiten von Ressourcen, PUT zum Anlegen neuer bzw. zur Aktualisierung vorhandener Ressourcen und DELETE zum Entfernen von Ressourcen. Die Methoden GET, PUT und DELETE sind idempotent, verursachen also äquivalentes Verhalten bei mehrfacher identischer Anwendung [1, vgl. S. 46 f.].
Mittels Confirmable-Nachrichten und Message-IDs schafft CoAP trotz Nutzung eines verbindungslosen Protokolles wie UDP Möglichkeiten zum zuverlässigen Datenaustausch. Erhält der Absender einer CON-Nachricht innerhalb einer gewissen, exponentiell ansteigenden Timeout-Zeit keine ACK-Nachricht mit derselben Message-ID wie in der originalen CON-Nachricht, so wird diese Nachricht automatisch erneut gesendet. Darüber hinaus können unbeabsichtigt dublizierte Pakete anhand identischer Message-IDs erkannt und ignoriert werden [1, vgl. S. 10 ff.].
CoAP unterscheidet zwei Möglichkeiten bei der Reaktion auf CON-Nachrichten. Steht einem Server das Ergebnis der von einer eingehenden CON-Nachricht geforderten Aktion sofort zur Verfügung, so kann er dieses mittels einer ACK-Nachricht an den anfragenden Clienten senden und die Bearbeitung der CON-Nachricht damit sofort abschließen. Dieses Antwortverhalten wird als „piggy-backed response“ bezeichnet. Dem gegenüber steht die „separate response“, bei der ein Server den Eingang einer gültigen CON-Nachricht zunächst mit einem leeren ACK-Paket bestätigt. Sobald die Antwort auf die Anfrage verfügbar ist, wird sie per CON-Nachricht an den Client gesendet, der in diesem Fall den Erhalt des Ergebnisses mit einem ACK-Paket gleicher Message-ID bestätigen muss [1, vgl. S. 12 f.].
Um die Last in ressourcenlimitierten Netzwerken zu reduzieren, ermöglicht CoAP Caching durch „Aktualitäts-“ und Gültigkeitsinfos, die in den Optionen und Antwort-Codes enthalten sein können. Mittels der „MAX-Age“-Option kann z. B. eine ablaufende Gültigkeitsdauer für eine Antwort festgelegt werden, innerhalb derer die Aktualität der Antwort garantiert ist und diese somit zwischengespeichert werden kann [1, vgl. S. 14, S. 41, S. 47 ff.].
Auch der Einsatz von Proxys bietet Möglichkeiten zur Reduktion von Netzwerkverkehr,

Antwortzeiten und Energieverbrauch der limitierten Ressourcen. Neben Performancesteigerung und besserer Erreichbarkeit für schlafende Knoten lassen sich auch Sicherheitsaspekte durch Proxying behandeln [1, vgl. S. 14].
Der Aufbau einer CoAP-Nachricht lässt sich am Besten an Hand des Paketaufbaus eines CoAP Datenpaketes nachvollziehen. Daher folgt nun die Erläuterung des CoAP-Paketaufbaus.

Der CoAP-Paketaufbau

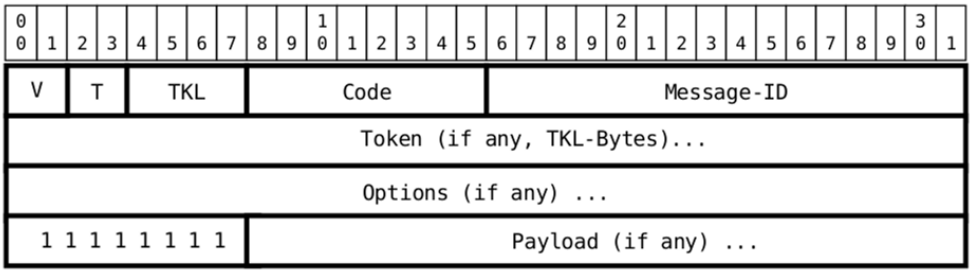

Abbildung 2.1.: Das CoAP-Datagramm [1, vgl. S. 15]

Der Header eines CoAP-Paketes ist vier Byte groß. Er enthält die verwendete CoAP-Versionsnummer (V), den genutzten Nachrichten-Typ (T), eine Längenangabe „Token-Length“ (TKL) für ein ggf. im Paket enthaltenes Token, einen acht Bit breiten Code, der je nach Anwendungsfall verwendete Anfrage-Methoden codiert oder Antwort-Codes angibt sowie eine 16 Bit breite, unsigned Integer Message-ID. Diese Message-ID wird dazu verwendet, Duplikate von Paketen zu erkennen. Es folgt ein null bis acht Byte langes Token, das zur Zuordnung von Anfragen und Antworten genutzt wird sowie eine variable Anzahl von Optionen, die im Type-Length-Value-Format angegeben werden. Hierüber lässt sich z. B. der in den Nutzdaten verwendete Media Type beschreiben. Für den Fall, dass es sich um ein nicht leeres CoAP-Paket handelt, schließt sich ein acht Bit Payload-Marker mit dem festgesetzten Wert 0xFF an die Optionen an, gefolgt von Nutzdaten, auch Nutzlast oder Payload genannt. Üblicherweise enthält die Payload eine Repräsentation einer Ressource oder das Resultat einer angefragten Aktion und wird durch Internet-Media-Type- bzw. Content-Format-Optionen näher spezifiziert. Die Payload-Größe wird anhand der Paketgröße bestimmt [1, vgl. S. 15 ff.]. Ob es sich bei einem konkreten Paket um eine Anfrage oder Antwort handelt, wird durch den Inhalt des Code-Feldes festgelegt. Anfragen enthalten hier einen Method-Code, der die verwendete Anfragemethode codiert während Antworten hierin über den Status der Bearbeitung mittels Fehler-Codes informieren.

Der CoAP-Request

Ein CoAP-Request enthält neben der anzuwendenden Methode einen Resource-Identifier, der auf die zu behandelnde Ressource verweist. Zudem können optionale Metadaten über die Anfrage, wie z. B. der Internet Media Type der Payload, enthalten sein. Mittels der „Accept“-Option wird der Client in die Lage versetzt, „content negotiation“ zu betreiben und bevorzugte Formate für die Antwort anzugeben [1, vgl. S. 40 f.].

Die CoAP-Response

Die Antwort auf eine Anfrage wird auch als Response bezeichnet. Um eine Antwort der entsprechenden Anfrage zuzordnen, enthält sie eine Kopie des vom Client generierten und in der Anfrage mitgesendeten Tokens. Darüber hinaus teilt ein Server per Response-Code den Status der Bearbeitung von Anfragen an den Client mit. Aktuell verwendet CoAP-folgende drei Klassen von Response-Codes: erfolgreich erhaltene, verarbeitbare und akzeptierte Anfragen führen zu Response-Codes der Klasse zwei. Ungültige Anfragen, z. B. durch Syntaxfehler, führen zu Client Errors der Klasse vier und Server Errors der Klasse fünf entstehen, wenn ein Server eine scheinbar gültige Anfrage nicht erfüllen kann. Innerhalb dieser Klassen sind detailreiche Unterteilungen vorgesehen (siehe Tabelle A.1). Dafür wird das Code-Feld im CoAP-Header aufgeteilt in einen Klassen- und einen Detail-Part [1, vgl. S. 30 f.].

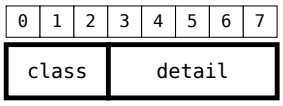

Abbildung 2.2.: Aufteilung des „Code“ Header-Feldes zur Response-Code Darstellung [1, vgl. S. 31]

Eine Erfolgreiche Anfrage auf eine Ressource mittels GET würde beispielsweise den Response-Code „2.05 Content“ zusammen mit einer Repräsentation der betreffenden Ressource zurückliefern.

2.3.2. Resource-Discovery via. CoRE-Link-Format

Ein CoAP-Server sollte im CoRE-Umfeld die durch sich angebotenen Ressourcen mittels des CoRE-Link-Formates erkundbar machen, damit M2M-Kommunikation möglich ist. Ob und welche Ressourcen ein Server für andere Netzwerkteilnehmer erkundbar macht, ist seine eigene Entscheidung [1, vgl. S. 63].
Zur Erkundung eines Servers kann ein Client eine URI-Referenz nutzen, die im Namespace des entsprechenden Servers liegt. Dazu muss der Client diese URI bereits kennen,

oder sie wird ihm bekannt gemacht. Alternativ kann der Client eine CoAP-Multicast-Anfrage an alle CoAP-Knoten im Netzwerk senden. Server, die erkundbare Ressourcen anbieten, müssen auf solche CoAP-Discovery-Anfragen auf dem Standard UDP Port 5683 lauschen [1, vgl. S. 62 ff.].
Ein CoAP-Client kann alle ihm durch Server angebotenen Ressourcen durch eine spezielle Anfrage der Form:

```
GET coap://host[:port]/.well-known/core
```

erhalten. Diese Anfrage kann an Singelcast- oder auch Multicast-Adressen versendet und bei Bedarf durch Angabe von Optionen gefiltert werden [1, vgl. S. 58, 105].
CoAP erweitert das CoRE-Link-Format zur Erkundung von Ressourcen um ein Conten-Format-Attribut „ct“, mit dessen Hilfe das von einer Ressource erwartete Inhaltsformat, z. B. XML, angegeben werden kann [1, vgl. S. 63].
Zur Identifizierung und Lokalisierung der hierarisch organisierten Ressourcen nutzt CoAP URIs. Der Aufbau einer CoAP-URI ist wie folgt:

```
"coap:" "//" host [ ":" port ] path-abempty [ "?" query ]
```

Zunächst wird das verwendete Protokoll genannt, gefolgt von einem durch einen Doppelslash getrennten Host. Der Host-Teil kann eine bekannte IP-Adresse oder ein durch ein Domain-Name-System (DNS) auflösbarer Name sein. Optional folgt durch ein „:“ getrennt eine Portnummer und anschließend eine Sequenz von Path-Segmenten, die mittels eines Slash getrennt werden. Dabei wird ein besonderer Path Prefix $/.well-known/core$ zur Host-Erkundung im CoRE-Link-Format verwendet. Über ein Fragezeichen getrennt sind anschließend Parameter anführbar. Diese erlauben eine weitere Parametrisierung der Ressourcen und werden voneinander mittels eines „&“ getrennt [1, vgl. S. 57 ff.].

2.4. Die Web Application Description Language

Die Web Application Description Language, kurz WADL, bezeichnet ein XML-basiertes Dateiformat zur maschinell verarbeitbaren Beschreibung von HTTP-basierten Web-Anwendungen [12, vgl. Kap. 1]. Mit WADL wird Service-Providern von RESTful Web-Anwendungen eine Möglichkeit an die Hand gegeben, die durch sich angebotenen Ressourcen maschinenverständlich zu beschreiben. Neben Repräsentationsformaten für Ressourcen und Beziehungen unter Ressourcen sind vor allem die auf Ressourcen anwendbaren Methoden detailliert beschreibbar. Dadurch kann ein Client alle ordnungsgemäß durchführbaren Anfragen an eine Ressource erlernen und somit sämtliche Funktionalitäten der Web-Anwendung selbständig ermitteln und nutzen [12, vgl. Kap. 1.1]. Er benötigt hierfür lediglich Zugriff zur WADL-Beschreibung der Ressource. Eine zentrale Registrierung einzelner Ressourcen ist nicht nötig. WADL stellt damit das für RESTful-Webservices geltende Gegenstück zur Web Service Description Language (WSDL) dar, welche SOAP-basierte Webservices beschreibt [7, vgl. S. 20].

2.4.1. Aufbau einer WADL-Datei

Der Aufbau einer WADL-Datei basiert auf hierarchisch verschachtelten Elementen. Die einzelnen WADL-Elemente können zudem über Attribute verfügen, welche diese näher beschreiben. Das Wurzelelement „application“ kann die folgenden weiteren Elemente, zum Teil auch mehrfach und verschachtelt, enthalten [12, vlg. Kap. 2.2 Kap. 2.12]:

- Ein „grammars“ Element zur Definition auszutauschender Datenformate,
- ein „resources“ Element als Container für beschriebene Ressourcen,
- „doc“ Elemente zur Dokumentation des entsprechenden WADL-Elementes,
- „resource“ Elemente zur Beschreibung konkreter Ressourcen,
- „method“ Elemente, die Ein- und Ausgabe für die jeweilige Methode auf der zugehörigen Ressource beschreiben,
- „representation“ Elemente zur Beschreibung der Repräsentation des Zustandes einer Ressource,
- „resource_type“ Elemente zur Gruppierung von Ressourcen, die die gleichen Methoden anbieten und
- „param“ Elemente zur Beschreibung parametrisierbarer Komponenten des jeweiligen übergeordneten Elternelementes.

2.4.2. Ressourcenbeschreibung mittels WADL

Zur besseren Veranschaulichung sei an dieser Stelle eine beispielhafte WADL-Beschreibung einer Ressource angeführt. Listing 2.4.2 führt in den Zeilen zwei bis sechs Namensräume als Attribute des „application“ Elementes an, die in der weiteren Beschreibung genutzt werden. Unter anderem auch den von allen WADL-Beschreibungen genutzten XML Namensraum „http://wadl.dev.java.net/2009/02“ [12, vgl. Kap. 2]. Zeile acht definiert per XML Schema ein Datenformat, welches die Anwendung nutzen kann [12, vgl. Kap. 1.2]. Das „resources“ Element (Zeilen 10-28) liefert durch das Attribut „base“ einen relativen Pfad, der für alle enthaltenen Ressourcen gilt. Das Beispiel zeigt eine Temperatursensor-Ressource (Zeilen 12-27), welche unter dem Pfad „coap://127.0.0.1/test/tempSens“ die Methoden GET (Zeilen 13-25) und DELETE (Zeile 26) anbietet. Während für die Methode DELETE keine weiteren Details beschrieben werden, wird die Methode GET durch Attribute näher beschrieben. Eine GET-Anfrage (Zeilen 14-20) an die beschriebene Ressource besitzt einen String-Parameter mit dem Namen „unit“. Dieser ist für eine gültige Anfrage erforderlich, jedoch existiert der Standartwert „Celsius“, falls keine anderen Angaben gemacht wurden. Die gültigen Werte

für den Parameter werden in den Zeilen 17 und 18 angeführt. Das „style“ Attribut definiert, wie der Parameter übergeben wird. Ein Query Parameter, wie im Beispiel zu sehen, kann bei Anfragen an die entsprechende URL durch ein „?“ getrennt als Zeichenkette nach dem Muster „Parametername=Parameterwert“ angehängt werden. Zur Auflistung mehrerer solcher Query-Parameter wird das Et-Zeichen als Trennungszeichen genutzt. Weitere Attribute für das Parameter-Element sind „id“, um auf eine Parameterdefinition anhand eines Identifiers verweisen zu können, „repeating“ zur Angabe, ob ein Parameter ein oder mehrere Werte anführt, „fixed“ zur Angabe eines festen Parameterwertes und „path“ zur Angabe eines auf das Elternelement bezogenen Pfades zum Wert des Parameters [12, vgl. Kap. 2.12]. Als Antwort auf eine Anfrage – z. B. „GET coap://127.0.0.1/test/tempSens?unit=Celsius“ – wird eine XML-Repräsentation des im „grammars“ Element bekannt gemachten Datenformats geliefert (Zeilen 21-24).

```
1  <?xml version="1.0"?>
   <application  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
5    xmlns:eg="http://example.com/EGMeteringResponse"
     xmlns="http://wadl.dev.java.net/2009/02">
7    <grammars>
       <include href="EGMeteringResponse.xsd" />
9    </grammars>
     <resources base="coap://127.0.0.1/test/">
11     <doc xml:lang="EN" title="A CoAP resource wadl descriptionexample"/>
       <resource path="tempSens">
13       <method name="GET">
           <request>
15           <param name="unit" type="xsd:string"
                  style="query" required="true" default="Celsius">
17               <option value="Celsius"/>
                 <option value="Fahrenheit"/>
19         </param>
           </request>
21         <response>
             <representation mediaType="text/xml"
23             element="eg:TemperatureResponse"/>
           </response>
25       </method>
         <method name="DELETE"/>
27     </resource>
     </resources>
29 </application>
```

Listing 2.1: WADL-Beschreibung einer Coap-Ressource

3. Ausarbeitung eines Entwickler-Tools für CoAP-Discovery

3.1. Anforderungen

Das Ziel der hier vorliegenden Arbeit ist die Erstellung eines Entwickler-Tools für CoAP-Discovery. Ein wesentlicher Aspekt bei der Entwicklung von Softwareprodukten ist die Spezifikation. Noch vor der Programmierarbeit werden hier wesentliche Anforderungen und Kerneigenschaften des zu erstellenden Softwareproduktes ausgearbeitet. Die Entwicklung des Produktes kann somit entlang wesentlicher Anforderungen und zielgerichtet erfolgen. Auch wird hierdurch eine Bewertung des Ergebnisses anhand der vorab erstellten Anforderungen ermöglicht. Eine gute Spezifikation ist Teil des Qualitätsmanagements für hochwertige Softwareprodukte. Im Folgenden werden nun wichtige Anforderungen an das zu erstellende Entwickler-Tool für CoAP-Discovery herausgearbeitet.
Die Hauptfunktionalität des Entwickler-Tools ist die vollautomatische Geräteerkundung vorhandener Ressourcen in leistungsbeschränkten Netzwerken. Ohne manuelle Zwischenschritte soll das Entwickler-Tool durch M2M-Kommunikation alle vorhandenen CoAP-Ressourcen eines Netzwerkes auflisten.
Bietet ein Service-Provider in einem REST-konformen Netzwerk erkundbare Ressourcen an, so kann er die Schnittstellen dieser Ressourcen veröffentlichen. Eine maschinenlesbare Möglichkeit hierfür bietet die Beschreibung der Ressourcen durch WADL-Dateien. Die Schnittstellen-Analyse und Darstellung erkundeter Ressourcen anhand solcher WADL-Dateien stellt eine weitere Kernfunktionalität des Entwickler-Tools dar. Hierdurch werden dem Nutzer des Entwickler-Tools vorhandene Funktionen der erkundeten Ressourcen aufgezeigt und deren Handhabung veranschaulicht.
Vorhandene Funktionen der erkundeten Ressourcen sollen durch den Entwickler zudem getestet werden können. Dazu soll es ermöglicht werden, einzelne Parameter aus dem Entwicklungs-Tool heraus mit konkreten Werten zu belegt, Funktionen mit entsprechender Parameterbelegung aufzurufen und das Ergebnis der Interaktion darzustellen.
Die Bedienung des Entwickler-Tools soll über eine grafische Benutzeroberfläche, eine sogenannte GUI (Graphical User Interface) erfolgen. Dies gestattet eine intuitive Handhabung für den Nutzer und die übersichtliche Darstellung wesentlicher Informationen. Eine Verwendung des Entwickler-Tools über Sprachgrenzen hinweg soll durch die Bedienung in englischer Fachsprache ermöglicht werden.
Um das Einsatzgebiet des Entwickler-Tools möglichst groß zu halten, bietet es sich an,

das Produkt in der Plattform übergreifenden Programmiersprache Java zu erstellen. Durch die zugrunde liegende „Java Virtual Machine“ ist somit ein einfacher und flexibler Einsatz auf verschiedenen Betriebssystemen möglich.
Um die Arbeit bei der Entwicklung und dem Testen neuer CoAP-Geräte zu erleichtern, ist es hilfreich, das Produkt flexibel erweiterbar zu gestalten. Somit kann ein Entwickler das Produkt nutzen und selbständig um individuelle Aspekte erweitern. Daher soll das entstehende Entwickler-Tool unter einer Open-Source Lizenz stehen. Bei der Erstellung des Produktes ist folglich auch darauf zu achten, dass keine proprietären Komponenten verwendet werden, die einer quelloffenen Lizenz des Endproduktes im Wege stehen würden.
Die wesentlichen Anforderungen an das Entwickler-Tool lassen sich im Kern wie folgt zusammenfassen:

- Realisierung maschineller Erkundung vorhandener CoAP-Ressourcen innerhalb eines REST-basierten Netzwerkes.
- Analyse und Darstellung der Schnittstellen erkundeter Ressourcen.
- Intuitive Bedienbarkeit des Entwickler-Tools durch eine grafische Benutzeroberfläche in englischer Sprache.
- Portabler Einsatz des Produktes durch Verwendung der Programmiersprache Java.
- Sicherstellung einer Open-Source-Lizenz für das Endprodukt.

3.2. Integration bestehender Softwaremodule

Bei der Entwicklung von umfangreichen Softwareprodukten kommt es nicht selten vor, dass Produkte einen hohen Komplexitätsgrad erreichen. Häufig tauchen in den Anforderungen an ein Softwareprodukt Funktionalitäten auf, die schon in einem anderen Umfeld softwaretechnisch umgesetzt wurden. Durch die modulare Integration bereits bestehender, quelloffener Projekte können so wesentliche Funktionalitäten schnell und kostengünstig umgesetzt werden. Dies ist jedoch nur dann möglich, wenn die entsprechenden Bibliotheken zur Verfügung stehen und unter Lizenzen veröffentlicht wurden, die eine solche Nutzung gestatten.
Bei der Integration bestehender Module stehen den Vorteilen der reduzierten Entwicklungsdauer und -kosten jedoch auch Nachteile gegenüber: genutzte Bibliotheken bringen ggf. zusätzliche Lizenzbedingungen für das entstehende Produkt mit sich. Hier muss im Einzelfall geprüft werden, ob die dadurch vorgeschriebenen Bedingungen eine Nutzung ausschließen oder ermöglichen. Auch werden mögliche Sicherheitslücken und Fehlfunktionen einer Bibliothek in das eigene Projekt mit integriert. Daher ist es ratsam, solche Bibliotheken und deren Herkunft kritisch zu betrachten und die Verwendung individuell

abzuwägen. Gegebenenfalls sind Änderungen an den zu integrierenden Quellen nötig, bevor eine sinnvolle und verlässliche Nutzung im eigenen Produkt möglich ist.
Um die unterschiedlichsten Komponenten eines komplexen Softwareprodukts funktionstüchtig und effektiv miteinander zu verbinden, ist ein umfassendes Verständnis für alle Teilkomponenten erforderlich. An dieser Stelle soll daher ein einführender Überblick über die Teilkomponenten des im Rahmen dieser Arbeit entwickelten CoAP-Discovery Tools geboten werden.
Bei der Entwicklung dieses Tools ist insbesondere auf zwei wesentliche Quellen zurückgegriffen worden:

- JCoAP [14]
- ShallowWadlParser [15]

JCoAP ist eine Java Bibliothek, welche das „Constrained Application Protocol“ (CoAP) implementiert. Sie stammt von der „Web Services for Devices“- (WS4D) Initiative der Universität Rostock und steht unter der “Apache License 2.0„ [16] für Software. Für das zu entwickelnde Softwareprodukt liefert JCoAP somit wesentliche Grundsteine für CoAP-Server und -Clientfunktionalitäten, sowie für den Austausch von CoAP-Paketen. Die zum Zeitpunkt der Bearbeitung verfügbare Version von JCoAP bezieht sich auf die CoAP-Spezifikation Version 11 [3]. Dies bringt einige wesentliche Unterschiede im Paketaufbau und der Verwaltung von Optionen eines CoAP-Datenpaketes im Vergleich zur gegenwärtig neuesten Version 18 mit sich, welche es zu beachten gilt.
Das zu entwickelnde Software-Tool soll WADL-Dateien zur Beschreibung von Ressourcen und deren Schnittstellen nutzen. Um solche WADL-Dateien zu parsen und die einzelnen beschriebenen Komponenten in entsprechende Datenstrukturen umzusetzen, wird ein WADL-Parser namens „ShallowWadlParser“ verwendet. Dieser analysiert eine WADL-Datei und liefert Datenstrukturen zu den enthaltenen Ressourcen, deren Methoden und zugehörigen Parametern [17]. Die entsprechende Javabibliothek [15] wurde an der Universität Alberta von Frau Reihaneh Rabbany entwickelt. Sie steht unter der „BSD-open source Lizenz“ [18] und wurde online inklusive Dokumentation verfügbar gemacht. Frau Rabbany hat der Nutzung ihrer Software im Rahmen dieser Bachelorarbeit schriftlich zugestimmt.

3.3. Entwicklungstools

Bei der Erstellung der Software, die im Rahmen dieser Bachelorarbeit angefertigt wurde, kamen einige wichtige Werkzeuge und Programme zum Einsatz, welche an dieser Stelle kurz dokumentiert werden.
Als integrierte Entwicklungsumgebung wurde “Eclipse für Java Entwickler„[1] mit dem Java Developer Kit (JDK) Version 1.7 der Firma Oracle eingesetzt. Die Dokumentation

[1] Version: 2.0.0.20130613-0530

der Software wurde durch das Tool „JavaDoc“, ebenfalls aus dem Hause Oracle und Teil des JDK [19], erstellt. Zudem wurden Call-Graphen mittels des bekannten Dokumentationstools Doxygen generiert und Klassendiagramme mithilfe des Eclipse Plugins ObjectAid UML Explorer angefertigt.
Bei der Entwicklung der grafischen Benutzeroberfläche kam das Eclipse-Plugin „Window Builder“[2] zum Einsatz.
Um Datenpakete zwischen dem zu entwickelnden Client und Testservern zu analysieren, fand bei der Entwicklung und Validierung das Tool „Wireshark“[3] Anwendung und zur Versionsverwaltung aller anfallenden Dateien wurde das verteilte Versionsverwaltungssystem „git“ genutzt.

[2]Version: 1.6.0.r43x201305211944
[3]Version 1.10.2

4. Implementierung des Entwickler-Tools „Corn Editor 1.0“

Die Implementierung des „Constrained Restful Network Discovery Tools - Corn Editor 1.0“ erfolgte entlang der Anforderungen an das Entwickler-Tool, welche in der Spezifikationsphase (Siehe Kapitel 3.1) zusammengetragen wurden. Das entstandene Software-Projekt unterteilt sich in sechs Java-Klassen, die unterschiedliche Aufgaben übernehmen. Im Folgenden werden nun zunächst Implementierungsansätze der wesentlichen Anforderungen dargestellt und anschließend wird eine genauere Analyse der wesentlichen Java-Klassen des Corn Editor-Projektes vorgenommen.

4.1. Implementierungsansätze für Anforderungen an den Corn Editor

4.1.1. Automatische Erkundung von CoAP-Ressourcen in REST-basierten Netzwerken

Das Ressource-Discovery-Verfahren des CoRE Link-Formates [10] findet im CoAP-Protokoll Anwendung. Dabei ist es einem CoAP-Client möglich, eine spezielle Anfrage der Form „GET /.well-known/core“ zu senden, welche lauschende CoAP-Server mit einer Liste der durch sie angebotenen Ressourcen beantworten.
Der Corn Editor stützt sich bei der Realisierung dieses Discovery-Mechanismus auf die Java-Bibliothek JCoAP[14]. Sie stellt wesentliche Client- und Server-Funktionalitäten bereit. Nach dem Start des Discovery-Prozess im Corn Editors durch Klicken des entsprechenden Buttons „Start Discovery“ wird zunächst die Resource-Discovery-Anfrage „GET“ mit dem Pfad-Attribut „/.well-known/core“ an einen CoAP-Server gesendet. Die gegenwärtige Implementierung des Corn Editors realisiert vorläufig lediglich sogenanntes „Unicast Resource-Discovery“. Das bedeutet, dass der Client diese Anfrage an eine konkrete IP-Adresse und einen konkreten Port sendet. Dem gegenüber steht das „Multicast Resource-Discovery“, bei welchem der Client solch eine Anfrage an die IP-Adresse einer Multicast-Gruppe sendet. In solch einer Multicast-Gruppe können beliebig viele CoAP-Server registriert sein, die alle auf eine solche Discovery-Anfrage mit einer Liste der durch sie angebotenen Ressourcen an den Client antworten.

```
REQ: GET /.well-known/core
RES: 2.05 Content
</test/temp>;rt="temperature";if="coap://127.0.0.1:5683/test/temp.wadl",
</test/light>;rt="light";if="coap://127.0.0.1:5683/test/light.wadl"
```

Listing 4.1: Beispiel einer erfolgreichen Corn Editor Resource-Discovery-Anfrage

Beim Absenden einer Discovery-Anfrage speichert der Corn Editor die Message-ID der zuletzt ausgesendeten Anfrage in einem Field „discovReqMsgID“. Erhält der Client in Zukunft eine Antwort mit identischer Message-ID, so handelt es sich dabei definitiv um eine Antwort auf diese Ressourcenerkundung. Sollte die Antwort auf eine Discovery-Anfrage erst nach dem Senden einer neueren Anfrage beim Client eintreffen, so wird diese nicht behandelt, da die Klassenvariable nun den Wert der neuen Message-ID enthält. Die Antwort auf die letzte Erkundungsanfrage enthält jedoch automatisch alle noch aktuellen Resultate vorangehender Anfragen und macht somit alte Antworten überflüssig.
Trifft also eine solche Antwort beim Client ein, so wird der Payload-String der Antwort durch den Editor aufbereitet und die einzelnen erkundeten Ressourcen werden in Objekten des Typs „Device“ mit den zugehörigen Attributen gespeichert. Ein solches Objekt enthält wesentliche Eigenschaften der erkundeten Ressource für die weitere Behandlung und zur Darstellung der Ressourcen für den Nutzer.

4.1.2. Darstellung erkundeter Ressourcen

Das obige Beispiel aus Listing 4.1 erkundet zwei Ressourcen, eine vom Resource-Typ (RT) „light“ und eine vom RT „temperature“. Beide liefern eine Adresse, an der die Beschreibung ihrer Schnittstelle (engl. Interface Description (IF)) zu finden ist. Es werden hierfür nun zwei Device-Objekte erstellt, die jeweils den Wert des RT-Attributs als Namen erhalten. Weitere ermittelte Parameter der Ressourcen, in diesem Fall die Adresse der jeweiligen Schnittstellenbeschreibung, werden ebenfalls im Device-Objekt gespeichert.
In einem Vektor werden alle aktuell erkundeten Device-Objekte genau einmal gespeichert. Dieser Vektor wird zur grafischen Darstellung der erkundeten Ressourcen auf eine Tabelle abgebildet, welche für jede erkundete Ressource den RT und die IP-Adresse ihres Servers auf der GUI anführt.
Sollten keine Ressourcen erkundet worden sein, so wird der Nutzer durch eine Meldung „No resources discovered!“ über diesen Zustand informiert. Anstelle dieses Hinweises wird der Nutzer andernfalls zur Auswahl einer Ressource in der Tabelle aufgefordert oder eine Übersicht über die entsprechend ausgewählte Ressource eingeblendet (siehe Abbildung 4.1). Für diese Darstellung wurde ein „CardLayout“ genutzt, welches verschiedene GUI Komponenten übereinander stapeln kann, jedoch nur die oberste sichtbar macht.
Die Tabelle der erkundeten Ressourcen ist nicht durch den Nutzer editierbar und gestattet lediglich die Auswahl einer Ressource, um über diese Ressource detailliertere Informationen anzeigen zu können.

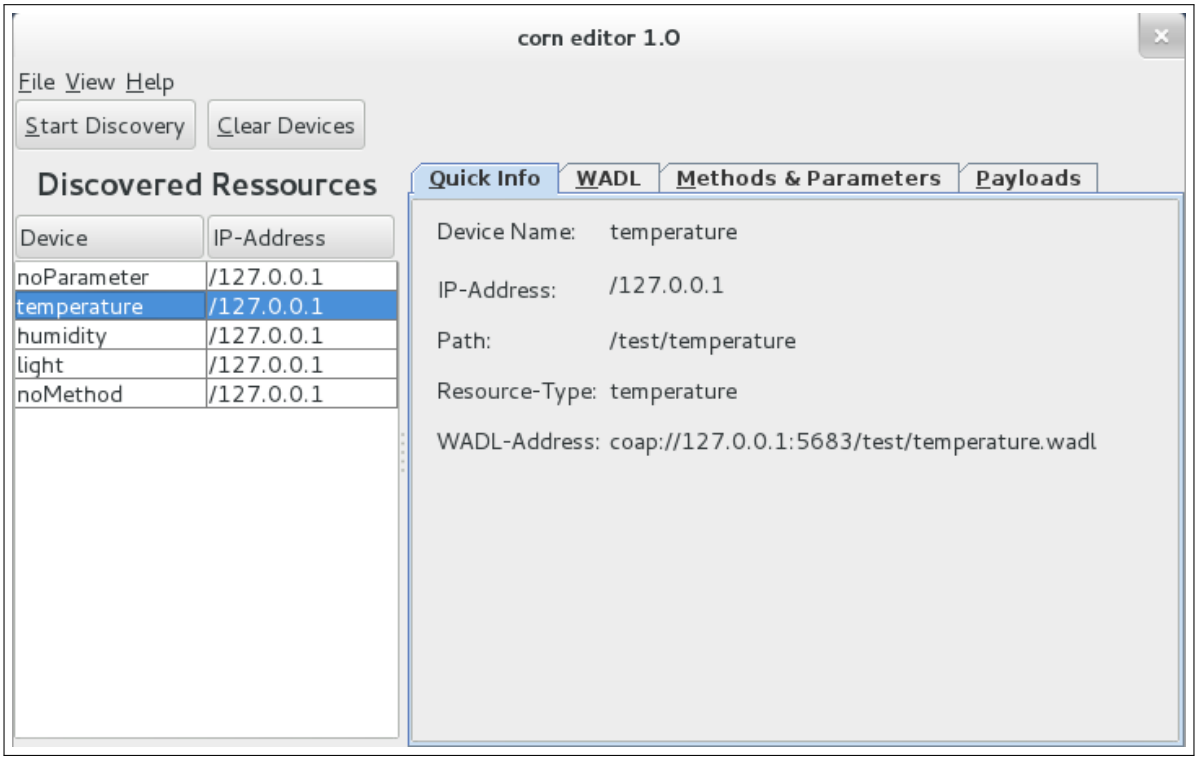

Abbildung 4.1.: Corn Editor – Detaildarstellung einer erkundeten Ressource

Eine tiefergehende Darstellung ausgewählter Ressourcen bieten die Reiterkarten „WADL“ sowie „Methods & Parameters“, auf welche in Kapitel 4.1.3 näher eingegangen wird.

4.1.3. Schnittstellenanalyse erkundeter Ressourcen

Server, die CoAP-Geräte anbieten, können eine maschinenverständliche Beschreibung der Schnittstellen für ihre Geräte mit anbieten. Der Corn Editor nutzt dazu WADL-Dateien. Ein Server kann bei der Ressourcenerkundung einer Ressource das Interface-Description-Attribut IF nutzen, um eine URI zu solch einer WADL-Beschreibung anzugeben. Das Device-Objekt, welches für jede Ressource erstellt wird, speichert diesen Verweis auf die entsprechende Beschreibung ab und der Corn Editor zeigt diese Datei nach Auswahl der erkundeten Ressource im WADL-Panel an. Dazu wird die Datei Zeile für Zeile mittels eines gepufferten Readers in das Textfeld des WADL-Panels geladen. So kann ein Entwickler die Interface-beschreibende WADL-Datei eines Gerätes direkt betrachten und analysieren. In der ersten Version des Corn Editors ist eine Übertragung der WADL-Dateien vom Server zum Client noch nicht implementiert. Stattdessen liefert der Server einen scheinbaren Pfad zur WADL-Datei, die bereits auf der Client Seite hinterlegt ist. Der Client nutzt lediglich den letzten Teil der vom Server erhaltenen URI,

um von einem ihm bekannten, festen Verzeichnis aus, relativ nach der entsprechenden WADL-Datei zu suchen. Diese Vereinfachung wurde vorgenommen, da der tatsächliche Speicherort der Datei für die Analyse und Darstellung der wesentlichen Informationen im Corn Editor keine entscheidende Rolle spielt. Eine zweckmäßige Entwicklung der grafischen Oberfläche wurde der Implementation der Datenübertragung zunächst vorgezogen.

Neben der Darstellung der WADL-Datei einer Ressource im Corn Editor wird diese auch mittels des „ShallowWadlParsers“ [15] in entsprechende Datenstrukturen umgesetzt. Der Parser bietet je einen Vektor für in der Beschreibung vorhandene Ressourcen-Elemente, die Methoden-Elemente der entsprechenden Ressource sowie Parameter-Elemente zu den zugehörigen Methoden. Dazu stehen die Klassen „ResourceNode“, „ParamNode“ und „MethodNode“ zur Verfügung [17]. Die entsprechenden Vektoren bieten eine Funktion an, um ihre Größe zu ermitteln. Eine WADL-Beschreibung, die keine Ressource bzw. Methode oder keinen Parameter enthält, führt jedoch zu NullPointerExceptions in der Parsing-Bibliothek. Zwar ist es wenig sinnvoll, dass eine WADL-Beschreibung keine Ressource enthält und auch die Beschreibung einer Ressource ohne angebotene Methoden ist fraglich. Sehr wohl ist es aber denkbar, dass eine Methode auf eine Ressource keine Parameter enthält. Für eine bessere Integration des Parsers in den Anwendungsfall des Corn Editors wäre eine ausgereifte Fehlerbehandlung von Vorteil, die gegenwärtig jedoch nur grob vorgenommen wird.
Die Reiterkarte „Methods & Parameters“ gestattet die Auswahl genau einer der durch die Ressource angebotenen Methode mittels einer Radio-Button-Gruppe und stellt die Parameter für die ausgewählte Methode grafisch dar. Parameter für Anfragen an die ausgewählte Ressource können durch ein grün gefärbtes Eingabefeld angegeben werden und Antwortparameter werden in nicht editierbaren, orangenen Feldern angegeben. Die Bezeichnung für Eingabeparameter beginnt zudem mit einem Pfeil von links nach rechts, während Ausgabeparameter einen entgegengesetzten Pfeil anführen. Sollte die ausgewählte Ressource keine Methoden anbieten oder eine angebotene Ressource über keine Parameter verfügen, so wird mittels eines CardLayout ein entsprechender Hinweis anstelle der Parameter angezeigt.

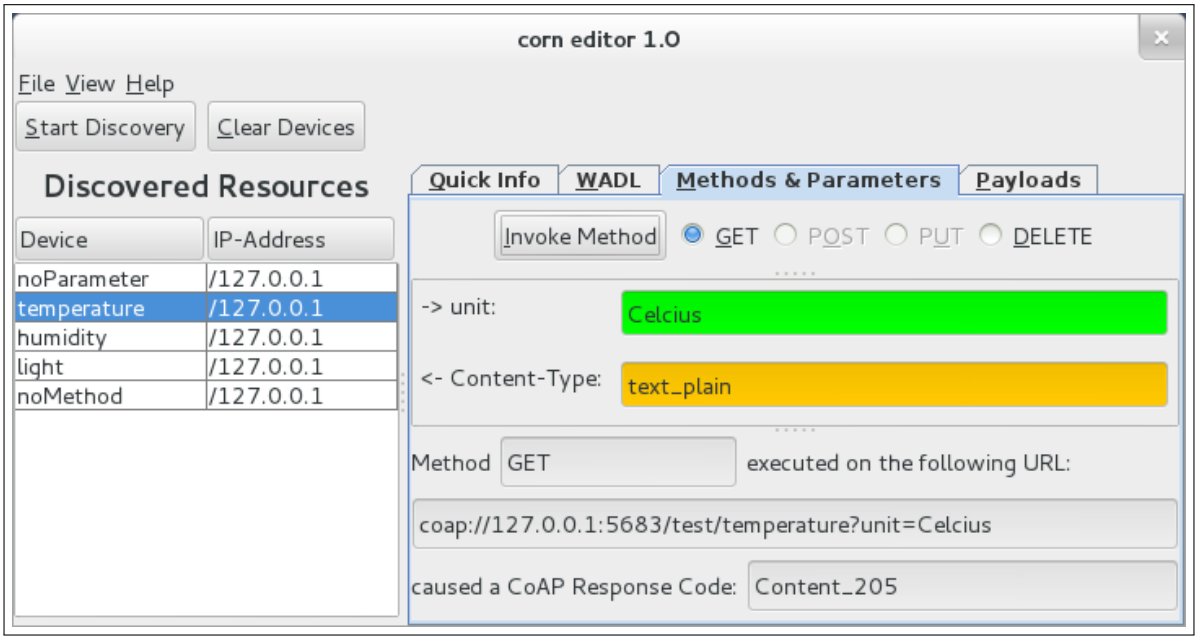

Abbildung 4.2.: Corn Editor – Senden einer „GET“-Anfrage

Die einzelnen grafischen Komponenten zur Parameterdarstellung einer ausgewählten Methode werden dynamisch zur Laufzeit bei Auswahl der erkundeten Ressource erzeugt. Für jede verfügbare Methode werden alle Parameter-Komponenten generiert und auf der entsprechenden Card-Komponente hinterlegt. Die Auswahl der Methode durch den Radio-Button führt dazu, dass die entsprechende Parameter-Card des GridCard Layouts angezeigt wird. Damit die dynamisch erzeugten Komponenten im weiteren Programmverlauf ansprechbar sind, um z. B. deren Inhalte zu lesen oder zu schreiben, wird für jeden Parameter ein Objekt der Klasse „ParamComponent“ angelegt. Dieses beinhaltet das den Parameter bezeichnende Label sowie das zugehörige Wertefeld und weitere Eigenschaften des entsprechenden Parameters. Die einzelnen ParamComponent-Objekte werden in einer Liste zusammengetragen, welche im Device-Objekt jeder Ressource hinterlegt ist.
Durch die Taste „Invoke Method“ kann die ausgewählte Methode angefragt werden. Neben Parametern kann hierzu im Panel „Payloads“ eine mögliche Payload angegeben werden, die in der zu sendenden CoAP-Anfrage enthalten sein wird. Im unteren Bereich des „Methods & Parameters“-Panels wird die ausgeführte Methode zusammen mit der Zieladresse der Anfrage angezeigt. Sobald eine Antwort auf die gesendete Anfrage beim Client eintrifft, zeigt dieser den Antwort-Code an, welcher Auskunft über Erfolg oder Misserfolg der Anfrage gibt. Im Panel „Payloads“ kann zudem die Payload der Antwort betrachtet werden, sofern die Antwort eine solche enthielt. So kann ein Entwickler eines CoAP-Gerätes überprüfen, ob die von ihm untersuchte Ressource sich verhält wie

erwartet und unterschiedlichste Anfragen an die Ressource testen.

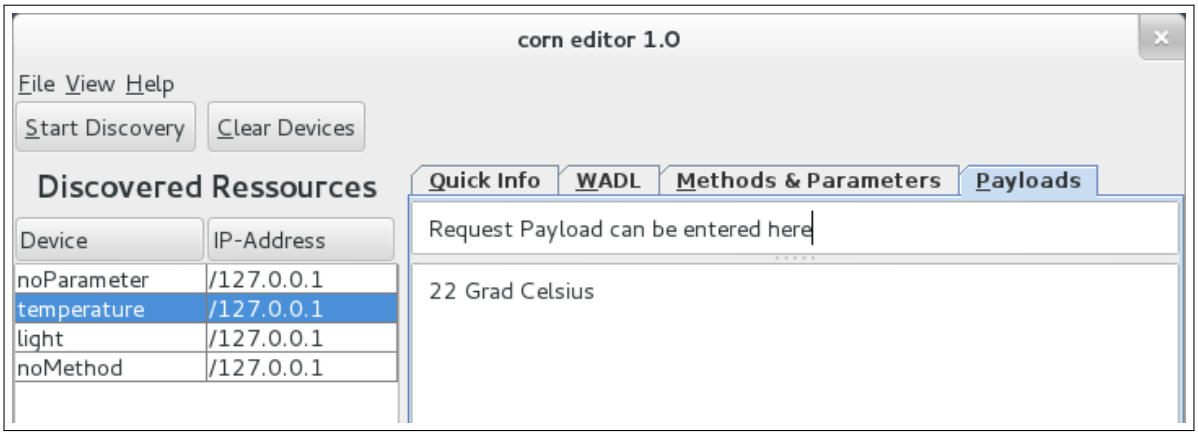

Abbildung 4.3.: Corn Editor – Darstellung der erhaltenen Payload einer Anfrage

Die Speicherung von Informationen zur Darstellung einer erkundeten Ressource in der GUI erfolgt direkt nach der Erkundung dieser Ressource und ist daher nach einem „InvokeMethod()“-Aufruf auf diese ggf. nicht länger aktuell. In der vorliegenden Version des Corn Editors ist somit ein erneutes Klicken der Taste „Start Discovery“ nötig, um die Darstellung auf den aktuellen Stand zu bringen.

4.1.4. Intuitive Bedienbarkeit und portable Einsetzbarkeit des Entwickler-Tools

Die grafische Nutzerschnittstelle des Corn Editors wurde mit dem Ziel Implementiert, eine intuitive Bedienung zu ermöglichen. Dabei wird die GUI im nativen „Look and Feel“ des jeweiligen Betriebssystems ausgeführt, was für eine gewisse Vertrautheit beim Nutzer sorgen soll.
Die Benutzeroberfläche ist komplett in englischer Sprache gehalten, was den Einsatz über den deutschsprachigen Raum hinaus ermöglicht. Das platzsparende Popup-Menü enthält alle wesentlichen Funktionen und Informationen zur Bedienung der Software. Wichtige Funktionen, wie der Start der Ressourcenerkundung, sind zusätzlich direkt und schnell auf der Programmoberfläche erreichbar. Direkt nach Starten des Corn Editors wird ein Info-Text angezeigt. Dieser beschreibt die vorliegende Software, informiert über Lizenzbedingungen des Editors und bietet eine Kontaktmöglichkeit zum Entwickler an. Per E-Mail kann der Nutzer so Fragen stellen, Anmerkungen geben oder Fehler melden. Intuitive Tastenkürzel erlauben eine schnelle Bedienung des Tools per Tastatur und erleichtern somit insbesondere erfahrenen Nutzern die Arbeit mit dem Editor. Verfügbare

Tastenkürzel sind hierbei grafisch hervorgehoben und können daher ohne Lernphase direkt angewendet werden. Ein kombinierter Tastendruck des mauslosen „Look and Feel“ Modifikators[1] und des jeweiligen unterstrichenen Buchstaben führt zur gewünschten Aktion. Tabelle A.3 auf Anhang-Seite XIII dokumentiert alle verfügbaren Tastaturkürzel sowie deren Gültigkeitsbereich.
Das Layout-Management stützt sich auf das dynamische Gridbag-Layout. Dieses erlaubt flexible Veränderungen der Größe und Position einzelner Komponenten. Durch entsprechende Parametrisierung einzelner Komponenten gestattet die Implementierung eine dynamische Vergrößerung bzw. Verkleinerung des Editor-Fensters. Dabei bleiben wesentliche Bedienelemente und Relationen einzelner Komponenten zueinander erhalten. Auch kann die Platzaufteilung einzelner Komponenten des Corn Editors somit zur Laufzeit durch den Nutzer individuell angepasst werden. Zum Beispiel lässt sich das Verhältnis der Payload Felder für Anfragen bzw. Antworten nach belieben anpassen. Komponenten, deren Inhalt stark in der Größe variieren kann, befinden sich in sogenannten ScrollPanes. Diese bieten bei Bedarf automatisch Scrollbalken an, falls die gegenwärtige Größe der Komponenten den aktuellen Inhalt nicht fassen kann. Dadurch lässt sich der benötigte Bildschirmausschnitt individuell auswählen.
Im Bereich „Methods & Parameter“ werden nicht editierbare Textfelder eingesetzt, um Informationen über gesendete Anfragen und erhaltene Antworten auszugeben. Obwohl dem Nutzer zu keiner Zeit gestattet wird, diese Textfelder zu manipulieren, wurden hier die nicht editierbaren Textfelder bewusst den sonst üblichen Textausgabe-Komponenten vorgezogen. Die Motivation dahinter ist es, auf die Dynamik der wichtigen, sich mit jeder neuen Anfrage ändernden Inhalte dieser Felder aufmerksam zu machen.
Durch Java als zugrunde liegende Programmiersprache ist die Software sehr portabel auf unterschiedlichsten Betriebssystemen einsetzbar. Die Integration der bestehenden CoAP-Implementierung JCoAP ist dadurch ebenfalls sichergestellt. Die gegenwärtige JCoAP Implementierung hinkt der aktuellsten Version der Spezifikation zwar gegenwärtig hinterher, sollte sich jedoch die JCoAP Implementierung weiterentwickeln, so kann auch der Corn Editor von dieser Entwicklung profitieren.

4.1.5. Open-Source Lizenzierung

Alle bei der Implementierung des Corn Editors eingesetzten Bibliotheken sind quelloffen und online zu finden. Es wurde darauf geachtet, dass die Lizenzen dieser Komponenten einer Open-Source-Lizenzierung des Corn Editors nicht im Wege stehen. Die zum Parsen von WADL-Dateien eingesetzte Java-Bibliothek [15] ist unter der „BSD-open source Lizenz“ [18] zur Verwendung freigegeben worden und die Implementierung JCoAP [14] wurde unter der „Apache License 2.0“ [16] veröffentlicht. Hieran angelehnt ist nun auch das entwickelte Softwareprodukt unter diese Lizenz gestellt worden, welche eine Verwendung, Verteilung und Veränderung der Quellen unter ähnlicher Lizenz gestattet [16].

[1] Standartmäßig die Taste „Alt“

4.2. Analyse der Java-Klassen des Entwickler-Tools „Corn Editor“

Neben den Klassen, welche aus den gut dokumentierten Bibliotheken zum Einsatz kommen, enthält die Software des Corn Editors folgende sechs Klassen:

- CornEditor
- RowListener
- CornCoAPClient
- Device
- ParamComponent
- CoapServerExampleRessources

Diese werden nun näher betrachtet und deren Funktion wird im Detail dargestellt. Alle Komponenten innerhalb der Klassen wurden, soweit möglich und sinnvoll, lokal definiert, um einen sauberen Programmierstiel zu pflegen und Gültigkeitsbereiche der Variablen kleinstmöglich zu halten. Einige Komponenten werden dennoch als Klassenvariablen definiert, da diese von mehreren Funktionen der Klasse heraus angesprochen werden und daher funktionsübergreifend zur Verfügung stehen müssen. Konstante Komponenten und Zeichenketten werden als final und somit unveränderbar angelegt und Attribute möglichst privat gehalten, was deren Nutzung auf die aktuelle Klasse eingrenzt. Im Programm genutzte Zeichenketten wurden als Konstanten angelegt, um eine einfache, zentrale Änderung der Inhalte in späteren Versionen zu ermöglichen.

4.2.1. Die Klassen „CornEditor“ und „RowListener“

Die Klasse CornEditor enthält die Main-Routine des Corn Editors. Ihre wesentliche Aufgabe ist die Darstellung der grafischen Oberfläche des Editors. Hierfür wird in der Main-Funktion das Betriebssystem-spezifische „Look and Feel“ für die Software aktiviert und ein separater Thread gestartet, welcher den Konstruktor der CornEditor Klasse aufruft. Darin werden die darzustellenden GUI-Komponenten durch die Funktion „initialize()“ erstellt und mit dem Funktionsaufruf „initializeMenu()“ ein entsprechendes Menü entwickelt. Ein Channel-Manager zur Behandlung des Netzwerkverkehrs wird instantiiert und das begrüßende Info-Panel angezeigt. Für die Verwaltung und Darstellung der GUI ist ein separater Thread nötig, damit die Verarbeitung der sichtbaren Komponenten parallel zur Ausführung der Programmlogik möglich ist. Änderungen an der Darstellung sollten daher auch stets und ausschließlich aus diesem Thread heraus durchgeführt werden. Ein durch das Dokumentationstool Doxygen erstellter Call-Graph für die Hauptroutine des Corn Editors ist im Anhang Seite IX in Abbildung A.7 zu sehen.

Nach Aufbau der GUI bietet diese Klasse wesentliche Funktionen zum Betrieb des Corn Editors an. So zum Beispiel die Funktion „startDiscover()“. Ihr Aufruf stellt zunächst einen Reset-Zustand der Oberfläche her und sendet parallel eine Resource-Discovery-Anfrage aus. Dazu werden die Funktionen „clearDevices()“ und „clearDiscoveredDevices()“ aufgerufen, welche ggf. veraltete Textausgaben entfernen, die Einträge der Tabelle erkundeter Ressourcen löschen und den Vektor mit gespeicherten, erkundeten Ressourcen leeren. Da nun keine Ressource ausgewählt sein kann, werden die Reiterkarten, welche eine ausgewählte Ressource näher beschreiben, vorläufig gesperrt, bis wieder eine gültige Auswahl vorliegt. Das Quick Info-Panel bleibt aktiv. Hier wird mittels „setQuickInfoDisplay()“ die Information ausgegeben, dass keine erkundeten Ressourcen vorliegen. Parallel dazu wird auf der als Field definierten Instanz der Klasse CornCoapClient ein CoAP-Client gestartet, welcher die erkundende „Get /.well-known/core“-Nachricht aussendet und auf Antworten lauscht. In Abbildung 4.4 ist der Call Graph der „startDiscover()“ Funktion grafisch dargestellt.

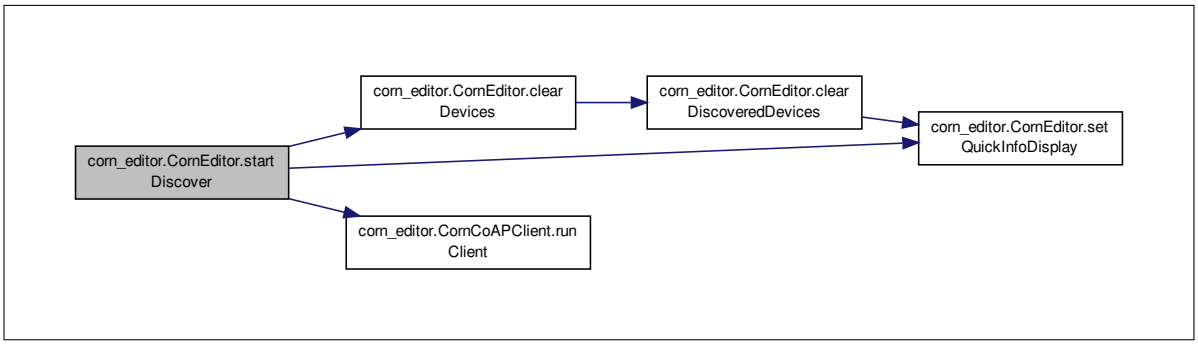

Abbildung 4.4.: Corn Editor – Call Graph der Funktion „startDiscover()“

Um erkundete Ressourcen auf die Tabelle abzubilden, wird für jedes Device-Objekt aus dem Vektor erkundeter Geräte mittels „addDevice“ ein Eintrag in der Tabelle erstellt. Die Position des Objektes im Vektor spiegelt dabei die Anzahl der Zeile des Eintrages in der Tabelle wieder.
Die Klasse „RowListener“ implementiert einen „ListSelectionListener“. Sollte sich die Auswahl in der Tabelle erkundeter Ressourcen ändern, so wird der Listener diese Veränderung registrieren und durch den Funktionsaufruf „valueChanged()“ das entsprechend gewünschte Programmverhalten realisieren. Dabei wird die Auswahl der bisher markierten Ressource auf die nun angewählte übertragen, alte Rückgabeinformationen gelöscht und die Eigenschaften der neu ausgewählten Ressource in die entsprechenden GUI Komponenten eingelesen. Das Interface-Description Attribut der ausgewählten Ressource wird genutzt, um die zugehörige WADL-Beschreibung einzulesen und im WADL-Panel zur Verfügung zu stellen. Alle Reiterkarten, die ggf. deaktiviert waren, werden nun wieder

aktiv und die WADL-Beschreibung des ausgewählten Objektes wird geparst. Abbildung A.8 im Anhang der Arbeit zeigt einen Call-Grapf für die Funktion „valueChanged“ der Klasse RowListener.
Beim Einlesen der WADL-Datei kommt die Funkton „corn_editor.importString()“ zum Einsatz, der ein Pfad zur gewünschten Datei übergeben wird. Durch einen gepufferten Reader wird der Inhalt der Datei nun Zeile für Zeile in einen String geladen, welche später im WADL-Panel angezeigt wird.
Damit die in der WADL-Datei dokumentierte Schnittstelle einer Ressource im Corn Editor grafisch aufgearbeitet werden kann, muss der Inhalt der Datei in entsprechenden Datenstrukturen gespeichert werden. Für jede erkundete Ressource wird ein Vektor angebotener Methoden aus MethodNode-Objekten angelegt. Dieser wird nach den vier Methoden „GET“, „POST“, „PUT“ und „DELETE“ durchsucht und bei Vorhandensein der jeweiligen Methode der entsprechende Radio-Button zur Auswahl dieser Methode aktiviert. Zudem werden durch den Funktionsaufruf „printParams“ alle für diese Methode angebotenen Parameter ermittelt und je ein Paar aus Namensfeld und Wertefeld angelegt. Diese Komponenten werden in einem ParamComponent-Objekt gespeichert und auf einer CardKomponenten grafisch dargestellt. Eingabeparamter werden grün, Ausgabeparamter orange hinterlegt. Welche CardKomponente einer Ressource im „Methods & Parameter“-Panel zu sehen ist, wird mittels CardLayout und der Auswahl der aktivierten Radio-Buttons geregelt. Standardmäßig wird der erste der verfügbaren Radio-Buttons entsprechend der oben genannten Reihenfolge der Methoden markiert, sofern mindestens eine Methode durch die aktuelle Ressource angeboten wird.
Um die Umsetzung der WADL-Schnittstellenbeschreibung in grafische Komponenten besser nachvollziehen zu können, folgt ein Auszug aus der im Anhang beschriebenen Test-Ressource „temperature“ zusammen mit den zugehörigen Bildausschnitten der GUI. Die Schnittstellenbeschreibung in Listing 4.2.1 der Test-Ressource verfügt über zwei „method“-Elemente mit den Namen „GET“ und „DELETE“. Daher werden die beiden Methoden in der Radio-Button-Gruppe aktiviert und können ausgewählt werden, wie in Abbildung 4.5 zu sehen ist. Da die Methode „GET“ in der WADL-Beschreibung das „param“-Element „unit“ als Kind eines „request“-Elements anfügt, wird ein Text mit dem ensprechenden Namen des „param“-Elementes und ein dazugehöriges Eingabefenster für den Wert des Parameters erzeugt. Weil es sich um einen Anfrageparameter handelt, wird das Eingabefenster grün eingefärbt. Das zweite „param“-Element der Methode ist ein Kind eines „response“-Elementes. Es handelt sich also um einen Antwortparameter, dessen Anzeige orange hinterlegt wird. Auch für diesen Parameter wird ein Text mit dem Namen des Parameters sowie ein Bereich zur Ausgabe des Parameter-Wertes erstellt und angezeigt.
Die Methode „DELETE“ besitzt laut WADL-Beschreibung keine Parameter. Wird diese Methode ausgewählt, so zeigt der zugehörige „Methods & Parameters“-Bereich einen entsprechenden Hinweis anstelle der Parameter

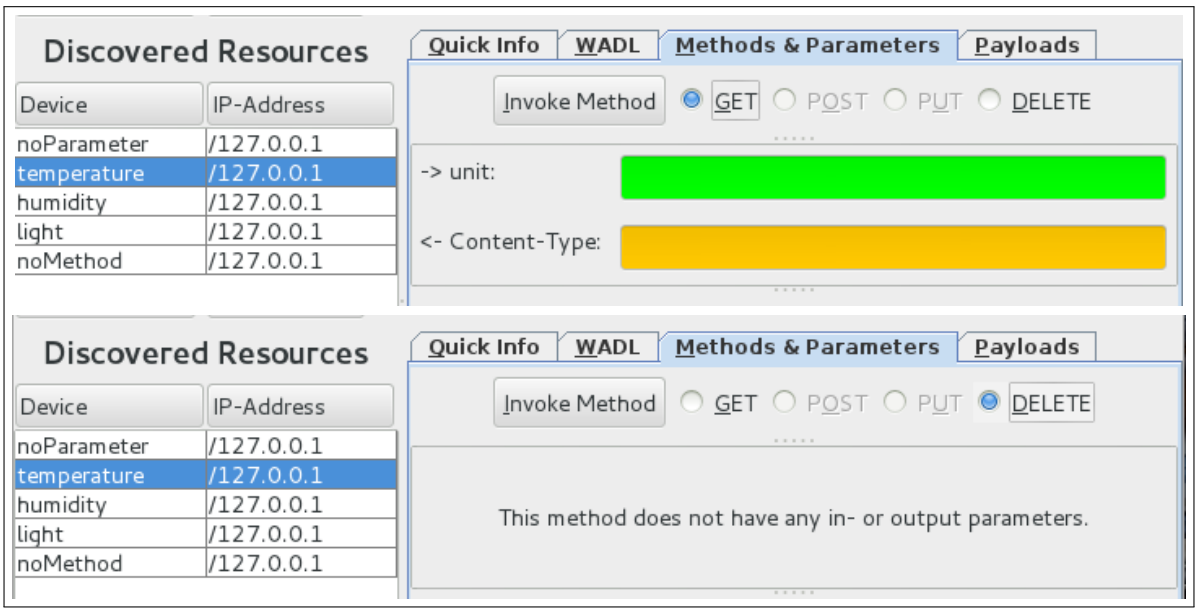

Abbildung 4.5.: Corn Editor – Grafische Umsetzung der WADL Parameter

```
<application  ...
  <resources base="coap://127.0.0.1/">
    <resource path="test/temperature">
      <!-- method offered by the resource -->
      <method name="GET">
        <request>
          <!-- specifying request paramter -->
          <param name="unit" type="xsd:string"
               style="query" required = "true" default="Celsius">
                 <option value="Celsius"/>
                 <option value="Fahrenheit"/>
          </param>
        </request>

        <response>
          <!-- specifying response paramter -->
          <param name="Content-Type" type="xsd:unsignedInt">
          </param>
        </response>
      </method>

      <!-- method offered by the resource -->
      <method name="DELETE">
      </method>
    </resource>
  </resources>
</application>
```

Listing 4.2: Auszug aus einer WADL-Schnittstellenbeschreibung

Nach erfolgreicher Erkundung von Ressourcen können diese entsprechend ihrer Schnittstellenbeschreibung angesprochen werden. Durch den Aufruf der Funktion „InvokeMethod()“ wird eine Anfrage an eine ausgewählte Ressource gesendet. Dazu werden zunächst wichtige Informationen für die Anfrage und Anzeige zusammengestellt, bevor der Aufruf der Funktion „cornCoapClient.performRequest()“ die logische Abhandlung der Anfrage an die Klasse CornCoAPClient übergibt.
Um Änderungen der Inhalte von Payload, Response-Codes und anderer Informationen auf der GUI zu realisieren, bietet die Klasse entsprechende Funktionen öffentlich an. Andere Klassen können über diese Schnittstellen die Inhalte der Oberfläche entsprechend anpassen. Um das Device-Objekt der aktuell markierten Ressource zu erhalten, wird die Hilfsfunktion „getSelectedDevice()“ und zur Aufhebung einer Auswahl die zugehörige „clearSelectedDevice()“-Funktion angeboten. Beide Funktionen machen von dem Boolean-Parametern „isSelected“ eines Device-Objektes Gebrauch. Sollte keine Ressource als ausgewählt markiert sein, so wird ein neues Device-Objekt erstellt, dessen „isSelected“-Wert jedoch auf „false“ gesetzt ist. So kann die Gültigkeit der Antwort auf eine „getSelectedDevice()“-Anfrage überprüft werden.

4.2.2. Die Klasse „CornCoAPClient“

Die logische Programmfunktionalität wird in der Klasse „CornCoAPClient“ realisiert. In einem von der Oberflächendarstellung getrennten Thread wird hier hauptsächlich die Kommunikation zwischen CoAP-Client und -Servern aus Client-Perspektive realisiert. Dazu werden IP-Adresse und Port des genutzten Servers als konstante Zeichenketten angegeben. So sind diese leicht an veränderte Netzwerkkonfigurationen anpassbar.
Ein als Field angelegter Vektor von Device-Objekten speichert alle durch den Client erkundeten Ressourcen. In einer Field-Liste werden durch den Client ausgesendeten CoAP-Anfragen an Ressourcen gespeichert. Die neueste Anfrage wird an das Ende der Liste angefügt. Dadurch kann bei eintreffenden Antworten durch den Server die zugehörige Anfrage ermittelt werden, indem die Liste von hinten nach vorne nach dem Element durchsucht wird, welches eine zur eingetroffenen Antwort identische Message-ID enthält.
Aus dieser Klasse heraus kann ein CoAP-Client gestartet werden. Die Funktion „runClient()“ übernimmt diese Aufgabe. Sie startet einen CoAP-Client, der eine Discovery-Anfrage über einen Client-Channel an die als Field definierte Server-IP-Adresse aussendet und speichert die Message-ID der Discovery-Anfrage zur späteren Auswertung.
Die Funktion „initializeCoapRequest()“ erstellt ebenfalls einen Client-Channel mit den als Klassenvariablen definierten Informationen zu Port und IP-Adresse des Servers und erzeugt ein CoapRequest-Objekt. Dieses kann nun mit Parametern bestückt werden, um die gewünschte Anfrage zu realisieren. So lassen sich auf dem CoapRequest-Objekt neben Attributen wie IP-Adresse und Port auch Pfad-Komponenten und Parameterwerte setzten und die Methode der Anfrage festlegen. Die so erstellte und individualisierte Anfrage kann anschließend mittels der Funktion „performRequest()“ vom CoAP-Client versendet werden.

Zur Bearbeitung eintreffender Antworten von CoAP-Servern bietet der CoAP-Client die Funktionen „onConnectionFailed()“ und „onResponse()“ an. Erstere informiert über die Standard-Konsolenausgabe des Clients darüber, wenn eine Verbindung gescheitert ist und Letztere beinhaltet die Behandlung eingetroffener CoAP-Antworten. Hier wird zunächst der Antwortcode der Nachricht verarbeitet und auf der GUI ausgegeben sowie die IP-Adresse des Servers ermittelt. Anschließend wird überprüft, um was für eine Antwort es sich bei der erhaltenen Nachricht handelt. Falls eine Antwort auf eine erfolgreiche „Delete“-Anfrage den Client erreicht hat, so wird sichergestellt, dass die entsprechende Ressource aus dem Vektor erkundeter Ressourcen entfernt wird, da diese Ressource nicht länger zur Verfügung steht. Ist die Antwort eine Reaktion auf die letzte ausgesendete Discovery-Anfrage, so wird die erhaltene Payload geparst und in entsprechende Device-Objekte transformiert. Erkennbar ist dies daran, dass die Message-ID der Antwort mit dem Wert der Klassenvariable übereinstimmt, die die Message-ID der letzten ausgesendeten Erkundungsanfrage enthält. Wie in Listing 4.1 zu sehen ist, werden mehrere durch einen Server angebotenen Ressourcen in der Antwort auf einer Erkundungsanfrage durch Kommata voneinander getrennt. Die Funktion „parseDiscoveryResponse()“ unterteilt die erhaltene Payload zunächst in einzelne Strings für jede Ressource. Durch die Funktion „parseElement()“ werden anschließend die Informationen über diese Ressource aus dem entsprechenden String herausgefiltert und in ein neu angelegtes Device-Objekt gespeichert. Dieses so erstellte Objekt repräsentiert die erkundete Ressource und wird dem Vektor aller erkundeten Ressourcen hinzugefügt, sofern es nicht bereits darin enthalten ist. Falls eine Antwort die Option „Content-Type“ enthält, so wird hierfür ein Label mit gleichem Namen und ein nicht editierbares Textfeld auf der GUI erstellt, in welchem der Inhalt dieser Option dargestellt wird.
Um Multicast-Resource-Discovery für den Corn Editor zu realisieren, muss der CoAP Client des Editors dahingehend verändert werden, dass die Netzwerkverbindung durch Multicast-Sockets aufgebaut wird. Anschließend bietet die „CornCoAPClient“-Klasse die beschriebenen Funktionen für alle Server an, die auf Anfragen dieser Klasse reagieren.

4.2.3. Die Klasse „Device“

Instanzen der Klasse „Device“ werden im Corn Editor genutzt, um erkundete Ressourcen zu repräsentieren. Eigenschaften einer Ressource, wie z. B. die IP-Adresse des Servers auf dem sie gespeichert ist, ein Pfad zur Lokalisation der Ressource auf dem Server, eine Adresse zu einer Schnittstellenbeschreibung der Ressource oder gar der Typ der Ressource werden in solchen Device-Objekten gespeichert. Neben IP-Adresse, Port des Servers der Ressource, dem relativen Pfad der Ressource auf dem Server sowie Resource-Typ und Interface-Description besitzt ein solches Objekt noch einen Boolean-Parameter, der erkennen lässt, ob die durch das Objekt repräsentierte Ressource aktuell vom User in der GUI ausgewählt wurde. Zudem enthält es eine Liste aller Methoden, welche die Ressource anbietet und eine Liste mit ParamComponent-Objekten (siehe Kapitel 4.2.4). Der Zugriff auf diese Parameter erfolgt dem objektorientierten Programmierstiel ent-

sprechend über Getter- und Setterfunktionen. Zusätzlich zu den Schnittstellen für die gerade beschriebenen Parameter bietet die Klasse noch eine Funktion, um ein Element der Methoden-Liste anhand des mitgelieferten Methodennamens auszuwählen, sowie eine Funktion zum Testen zweier Device-Objekte auf Identität und eine Funktion zur Ermittlung der Methoden, die eine Ressource anbietet.

4.2.4. Die Klasse „ParamComponent“

Die durch Ressourcen angebotenen Methoden können über Parameter verfügen, welche bestimmte Eigenschaften besitzen. Um diese Parameter und deren Eigenschaften repräsentieren und grafisch darstellen zu können, nutzt der Corn Editor Instanzen der „ParamComponent“ Klasse. Alle Parameter einer durch eine Ressource angebotenen Methode werden entsprechend der ausgewählten Ressource und Methode im „Methods & Parameters“-Panel dargestellt. Die dazu nötigen Komponenten werden zur Laufzeit erstellt und der Oberfläche hinzugefügt. Um zu späteren Zeitpunkten auf diese Komponenten und deren Inhalt noch zugreifen zu können, wird für jeden Parameter ein solches ParamComponent-Objekt erstellt. Hier wird das Label mit dem Parameternamen sowie das Textfeld für den Wert des Parameters gespeichert. Zudem erfolgt eine Zuordnung zur entsprechenden Methode, welche diesen Parameter anbietet, eine Charakterisierung des Styles dieses Parameters sowie eine Charakterisierung der Parameter nach Anfrage- bzw. Antwortparameter. Für die enthaltenen Parameter stellt die Klasse alle entsprechenden Getter- und Setter-Funktionen zur Verfügung. Durch Aufnahme weiterer Variablen in diese Klasse ist eine Implementierung weiterer Eigenschaften von Parametern in der Zukunft leicht umsetzbar.
Alle ParamComponent-Objekte einer Ressource werden in einer Liste zusammengefasst und diese Liste wird in entsprechenden Device-Objekt gespeichert.

4.2.5. Die Klasse „CoapServerExampleRessources“

Die Klasse „CoapServerExampleRessources“ implementiert einen CoAP-Server. Sie ist eine modifizierte Version des „CoapSampleResourceServer“ aus dem ws4d-jcoap-applications-Paket der Bibliothek JCoAP [14]. Sie dient als Testserver für den entwickelten Corn Editor und wird zur Validierung und Auswertung der Software eingesetzt.
Die Hauptroutine des Servers startet einen CoAP-Server. Dieser muss die in der Klasse „CornCoAPClient“ festgelegten Werte für seine IP-Adresse und den Port nutzen, damit der Client die Ressourcen auf diesem Server erkunden kann. Ein detailliertes Logging der Ereignisse des Servers wird zudem aktiviert und einige simulierte Beispiel-Ressourcen werden erstellt und dem Server hinzugefügt. Für die Ressourcen werden an dieser Stelle wesentliche Parameter wie Resource-Typ und eine Adresse für Interface-Beschreibung festgelegt. Anschließend wird der CoAP-Server gestartet, was die Netzwerkports öffnet und die Ressourcen im Netzwerk erreichbar macht. Um einige Testgeräte mit einer simplen Dynamik zu verstehen, wird ein Timer erstellt. Dieser wird alle fünf Sekunden

inkrementiert und der jeweils aktualisierte Zählerstand von manchen Ressourcen als neuer Datenwert übernommen. So kann in der Simulation durch Anfragen mit zeitlichem Abstand ein verändertes Ergebnis erzielt werden.

5. Validierung der entwickelten Software & Auswertung der Ergebnisse

5.1. Validierung der Corn Editor Software

Der Corn Editor stellt das Resultat der beschriebenen Software-Entwicklungsarbeit dar. Wie bei jeder komplexeren Software bietet es sich an, das entstandene Produkt genauer unter die Lupe zu nehmen, um Abweichungen zum gewünschten Verhalten und nicht behandelte Situationen ausfindig zu machen. Im Idealfall bieten sich regelmäßige Unit-Tests an, welche das gewünschte Verhalten nachhaltig überprüfen können. Um Denkfehler des Softwareentwicklers aufzudecken ist es zudem von Vorteil, wenn eine andere Person als der Produktentwickler Testfälle für das Produkt implementiert. Die Bearbeitungszeit ließ jedoch eine zusätzliche, zeitaufwendige Entwicklung von Unit-Tests nicht zu und eine Einbindung weiterer Person zur Entwicklung von Test-Szenarien war im Rahmen dieser Abschlussarbeit ebenfalls nicht möglich. Daher wurde zur Validierung der Software ein sehr primitiver CoAP-Server erstellt, welcher fünf simulierte Test-Ressourcen anbietet. Für die genutzten Test-Ressourcen wurden entsprechende Schnittstellenbeschreibungen in WADL erstellt und bereitgestellt. Diese finden sich im Anhang der Arbeit (siehe Anhang A.5).
Um das Verhalten des Corn Editors analysieren zu können, wurden die gesendeten und empfangenen CoAP-Pakete betrachtet. Dazu wurde das Netzwerkanalysetool „Wireshark“ eingesetzt. Hierbei fiel zunächst auf, dass das erwartete Verhalten der JCoAP-Bibliothek von dem in der neuesten CoAP-Spezifikation beschrieben Verhalten abweicht. Die aktuellste Spezifikation von CoAP [1] liegt in Version 18 vor. Die Implementation der Bibliothek orientiert sich jedoch an Version elf [3]. Das bringt einige Änderungen im CoAP-Paketaufbau mit sich und in der Behandlung von CoAP-Optionen. Der in Kapitel 2.3.1 beschriebene Paketaufbau ist in weiten Teilen auch für CoAP-Version elf gültig. Jedoch kennt diese Spezifikation noch kein separates Token-Feld zur Zuordnung von Anfragen und Antworten im CoAP Header. Das Token wird in CoAP V.11-Paketen noch als Option Nr. elf übertragen, wie Tabelle A.2 im Anhang zeigt. Folglich existiert auch das Token-Lengh-Feld noch nicht. Statt dessen wird daraus ein Option-Count Feld, dessen Inhalt angibt, wie viele Optionen das Paket enthält. Die Optionen werden in beiden Versionen im Type-Length-Value-Format angegeben und müssen entsprechend

ihrer Optionsnummern (siehe Tabelle A.2) geordnet im Paket auftreten. Dabei wird zunächst die Optionsnummer, dann die Länge der Option in Byte und anschließend der Optionswert angegeben. Die neuere CoAP-Spezifikation gestattet jedoch durch größere Felder für Optionsnummer und -Länge größere und vielfältigere Optionen. Für genauere Beschreibungen einzelner Optionen sei auf die jeweilige Spezifikation verwiesen. Da in Version elf die Anzahl der Optionen durch das Option-Count-Feld definiert wird, fällt auch der dann nicht nötige Payloadmarker weg, welcher das Ende der Optionen signalisiert. Für die Implementierung des Corn Editors sind diese Unterschiede jedoch nicht von besonders großer Relevanz, sodass sie an dieser Stelle lediglich der Vollständigkeit halber angeführt wurden.
Damit der Editor in einem realistischen Einsatzszenario getestet werden konnte, kamen die bereits erwähnten Test-Ressourcen zum Einsatz. Der zur Validierung erstellte Server (siehe Kapitel 4.2.5) bietet diese Ressourcen an. Die Erkundung der Ressourcen eines Servers erfolgt durch Klicken des Buttons „Start Discovery“. Die Unicast-Discovery-Anfrage „GET“ auf den Pfad „/.well-known/core“ führt bei laufendem Server dazu, dass eine entsprechende Antwort auf die Ressourcen-Erkundung im Core Link-Format beim Corn Editor eintrifft. Diese ist nach der Erkundung auf der Payloads Reiterkarte zu sehen.

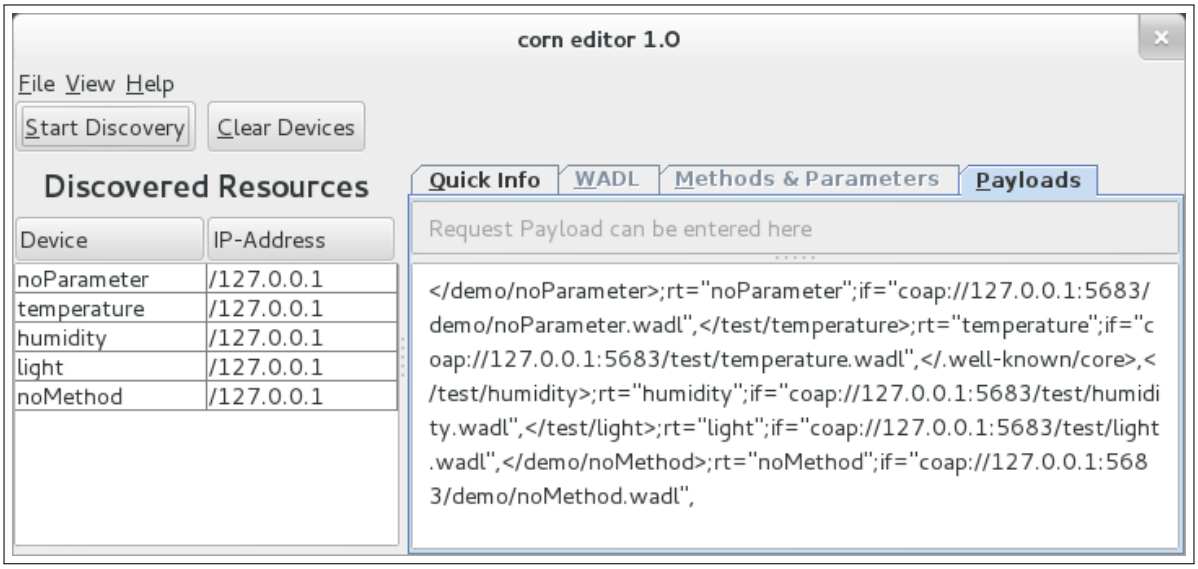

Abbildung 5.1.: Corn Editor – Payload nach Ressourcen-Erkundung

In der „Discovered Resources“-Tabelle der GUI werden die erkundeten Ressourcen mit IP-Adresse ihres Servers aufgelistet und können einzeln durch den Nutzer ausgewählt werden. Eine Markierung mehrerer Tabelleneinträge zur gleichen Zeit ist nicht möglich.

Nach Auswahl einer erkundeten Ressource werden entsprechende Detailinformationen im „Quick Info“-Bereich angezeigt (siehe Abbildung 4.1) und die WADL-Schnittstellenbeschreibung ist in der zugehörigen Reiterkarte einsehbar. Das Ausführen weiterer, ggf. durch Parameter und Payload näher spezifizierter, Anfragen ist im Panel „Methods & Parameters“ möglich.
Bei der Erstellung der Demo- und Test-Ressourcen wurde darauf geachtet, WADL-Beschreibungen mit möglichst vielen verschiedenen Variationen zu nutzen, um ein umfassendes Testergebnis zu erziehlen. So wurden Demo- und Test-Geräte mit jeweiliger Pfadangabe „/demo“ bzw. „/test“ erstellt. Zwei Demo-Geräte sollen Sonderfälle abdecken, in welchen die WADL-Beschreibung gar keine verfügbaren Methoden (siehe Listing A.4) bzw. keine Parameter (siehe Listing A.5) für vorhandene Ressourcen anführt. Wählt ein User im Corn Editor nach erfolgreicher Ressourcen-Erkundung die entsprechende Ressource aus und wechselt zur Ansicht „Method & Parameters“, so wurde dem Nutzer zunächst in beiden Fällen die Meldung „Selected resource does not offer any methods!“ angezeigt. Dies sollte zwar für die Demo-Ressource ohne Methoden so sein, jedoch sollte für die Demo-Ressource ohne Parameter stattdessen der Hinweiß „This method does not have any in- or output parameters.“ erscheinen. In beiden Fällen wurde auch kein Radio-Button zur Methodenauswahl aktiviert, obwohl die Demo-Ressource „noParameter“ laut ihrer Schnittstellenbeschreibung die Methode „DELETE“ anbietet. Die Tests mit diesen Demo-Geräten haben ergeben, dass der genutzte WADL-Parser nicht in der Lage war die WADL-Beschreibungen zu parsen, ohne dass zusätzliche Definitionen einiger häufig genutzer WADL-Namensräume ergänzt wurden. Solche auftretenden Fehler lassen sich auf der Konsolen- und Fehlerausgabe des Programms durch entsprechende Exception-Stacks und Fehlermeldungen beobachten. Der Nutzer der GUI erhält jedoch keinerlei Information über fehlgeschlagene Parsing-Versuche der Schnittstellenbeschreibung. Die WADL-Beschreibung der Demo-Ressource noParameter wurde entsprechend ergänzt, sodass nun der korrekte Hinweis auf Abwesenheit von Parametern für die ausgewählte Ressource „DELETE“ im entsprechenden Fenster angezeigt wird.
Die Test-Ressource „temperature“ (siehe Listing A.1) bietet zwei Methoden an. Die durch diese Ressource angebotene „DELETE“-Methode enthält ebenfalls keine Parameter und führt nach Auswahl zum entsprechenden Hinweis, dass keine Parameter existieren. Eine zweite Methode „GET“ besitzt hingegen einen Request-Parameter „unit“ zur Angabe einer Temperatureinheit und einen Response-Parameter „Content-Type“. Darin wird das Räpresentationsformat der Antwort-Payload beschrieben [3, vgl. S. 43]. Sollte eine Antwort diese Option beinhalten, so wird der Wert nach Erhalt der Antwort hier dargestellt. Der Request-Parameter ist vom Typ Query und wird entsprechend der Spezifikation von CoAP an die URI der Anfrage angefügt. Die zuletzt ausgeführte Methode und die hierfür erstellte URI werden im unteren Bereich des „Methods & Parameters“-Panel angezeigt. Sobald eine Antwort auf die Anfrage eingetroffen ist, wird der Antwort-Code ebenfalls in diesem Bereich angezeigt. Die Abbildung 4.2 aus Kapitel 4.1.3, Seite 23 zeigt die Situation nach Absenden einer „GET“-Anfrage auf diese Ressource. Abbildung 4.3 auf Seite 24 im gleichen Kapitel zeigt die erhaltene Payload

im entsprechend durch die erhaltene „Content-Type“ Option beschriebenen „text/plain“ Format. Der Request-Parameter „unit“ wird in der Schnittstellenbeschreibung als notwendig definiert und zudem werden die möglichen Werte für den Inhalt der Variable als „Celsius“ bzw. „Fahrenheit“ festgesetzt. Die Darstellung der Parameter im Corn-Editor unterscheiden gegenwärtig noch nicht zwischen notwendigen und optionalen Parametern. Auch werden die Inhalte der Eingabefelder für Werte nicht auf gültigen Inhalt überprüft. Daher wurde der Parameter mit dem Standartwert „Celsius“ belegt. Die Validierung des Corn Editors anhand der Test-Ressource „temperature“ deckt somit Potential für zukünftige Verbesserungen der Software auf.
Die Test-Ressource „humidity“ ähnelt der „temperature“ Ressource sehr. Sie besitzt jedoch keinen Input-Parameter für die Methode „GET“ an. Dennoch enthält die zugehörige WADL-Beschreibung (siehe Listing A.3) ein leeres „request“ Element. Nach ersten Tests ohne dieses stellte sich heraus, dass der WADL-Parser Methoden nicht parsen kann, welche lediglich ein "response“-Element besitzen. Die „DELETE“ Methode der selben Ressource zeigt jedoch auch, dass eine Methode ganz ohne „request“- und „response“-Element jedoch vom Corn Editor akzeptiert wird.
Die letzte und umfassendste Test-Ressource „light“ bietet alle von CoAP genutzten Methoden mit unterschiedlichen Anzahlen an Request- / Response-Parametern an. Somit lässt sich die dynamische Erzeugung und Darstellung der grafischen Komponenten für Parameter im Corn Editor testen. Außerdem sind durch unterschiedliche Anfragen an diese Ressource auch verschiedenste Verhalten der Antworten untersuchbar. Die Tests zeigten, dass die Darstellung mehrerer Parameter-Komponenten möglich ist. Sollte der Parameterbereich für die Darstellung aller Elemente zu klein werden, tauchen Scrollbalken zur Auswahl des sichtbaren Bereiches auf. Jedoch können manche Komponenten der GUI kleiner werden, als sinnvoll. So wächst der Parameterbereich im schlimmsten Fall so weit, dass die Methodenauswahl samt Button „Start Discovery“ unsichtbar wird. Durch manuelle Verschiebung der Komponenten lassen sich jedoch alle Komponenten wieder sichtbar machen. Die durch die Test-Ressource „light“ angebotene Methode „POST“ besitzt zum Beispiel drei „request“- und zwei „response“-Parameter und führt bei kleiner Fenstergröße zum Erscheinen eines Scrollbalken. Werden alle drei „request“-Parameter der Methode mit Werten belegt, so ist erkennbar, dass alle diese Query-Parameter nach einem „?“ an die Anfrage-URI angehängt und durch „&“ untereinander getrennt werden. Der Aufbau der Anfrage-URI erfolgt also auch bei mehreren Parametern korrekt (siehe Abbildung 5.2). Der Server sendet auf die „PUT“-Anfrage eine entsprechende Antwort mit dem Code 2.04, welcher auf Veränderung der entsprechenden Ressource hinweist (siehe Tabelle A.1 auf Anhang-Seite XI).

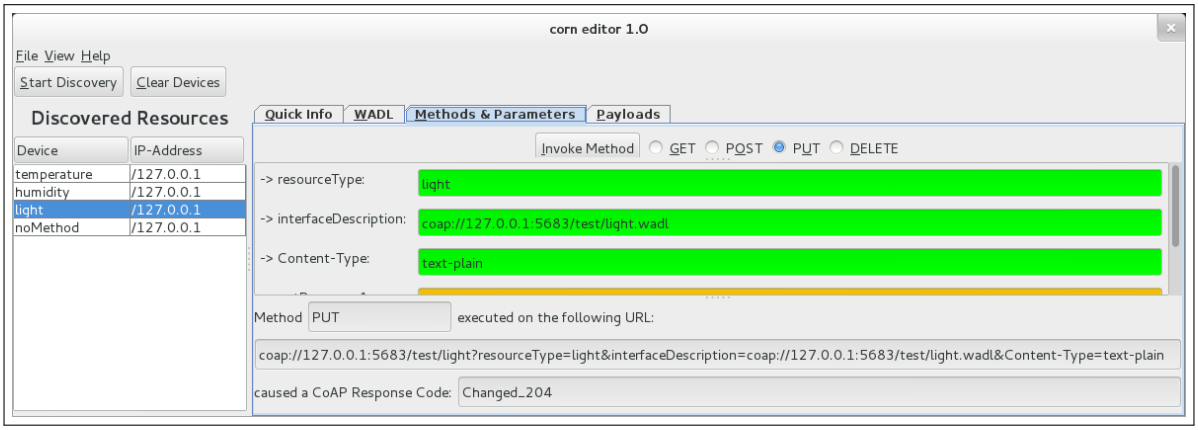

Abbildung 5.2.: Corn Editor – „PUT“-Anfrage an Ressource „light“

Wird auf eine der Ressourcen eine erfolgreiche „DELETE“-Anfrage ausgeführt, so taucht im Ausgabefeld des Antwort-Codes die entsprechende Meldung „Deleted_202“ auf. Daran ist erkennbar, dass die Ressource auf dem Server entfernt wurde, jedoch ist die Ressource noch auf der GUI des Corn Editors vorhanden. Erst durch ein erneutes „Start Discovery“ wird die Darstellung der Ressourcen im Corn Editor aktualisiert. Dabei werden jedoch auch alle eingegeben Parameterwerte und Rückgabe-Informationen zurückgesetzt. Solange die Darstellung der Ressourcen nicht aktualisiert ist, kann eine veränderte oder gelöschte Ressource auf der GUI auch noch auswählt werden. Es können sogar auch noch weitere Anfragen an diese Ressource gesendet werden. Logisch folgerichtig scheitert eine „GET“-Anfrage an eine Ressource im Anschluss an ein zuvor gesendetes „DELETE“. Die Antwort enthält den Antwort-Code „Not_Found_404“. Jedoch sorgen „POST“ und „PUT“ für das neue Erstellen einer Ressource. Wird eine dieser Methoden nach dem Löschen einer Ressource ausgeführt, so wird an der gleichen Adresse eine neue Ressource angelegt. Die Antwort enthält in diesem Fall den Antwort-Code „Created_201“. Auf diese Art neu angelegte Ressourcen haben keinen definierten RT-Wert und keine definierte IF-Adresse mehr. Dies ist in der Quick Info-Ansicht der neu erstellten Ressource zu erkennen (siehe 5.3). Zwar existieren für die Methode „PUT“ der „light“ Ressource Parameter, die den Anschein erwecken, dass sie diese Attribute mit Werten belegen würden, jedoch behandelt der Test-Server die mitgesendeten Parameter noch nicht. Eine erfolgreiche „GET“-Anfrage ist anschließend aber wieder möglich.

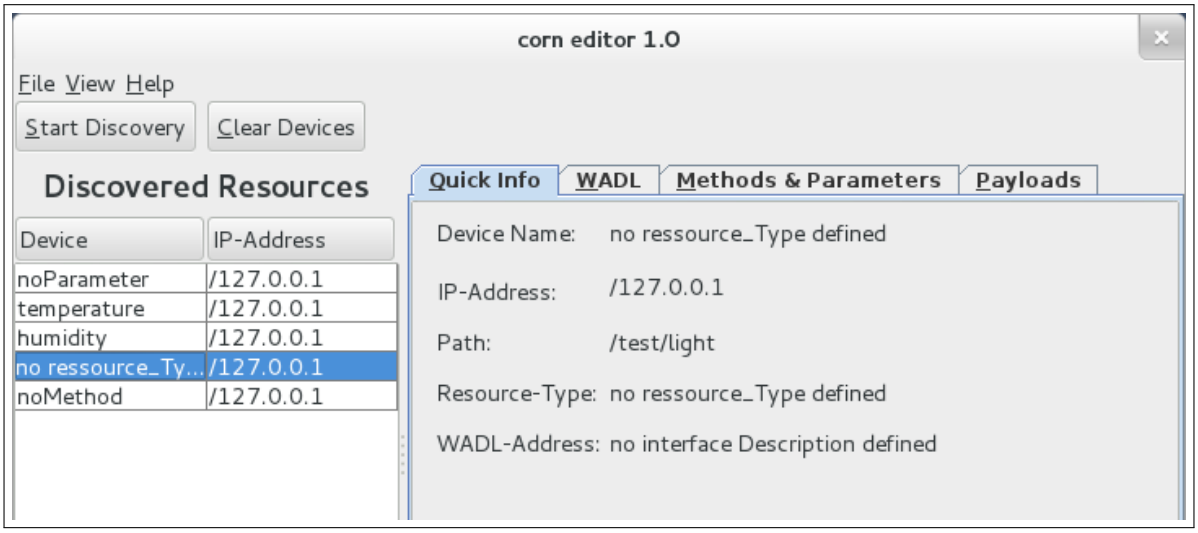

Abbildung 5.3.: Corn Editor – Quick Info einer entfernten und neu angelegten Ressource

In Abbildung 5.3 ist zudem erkennbar, dass der Gerätename einer Ressource im Quick Info-Panel lediglich den Wert des Resource-Types kopiert. Darüber hinaus ist auch zu sehen, dass trotz nicht definierter Adresse einer Schnittstellenbeschreibung der neu angelegten Ressource das WADL-Panel aktiv und auswählbar ist. Nach Auswahl der Reiterkarte findet sich dort auch eine WADL-Beschreibung. Das liegt an zwei Aspekten. Zum einen wird die Reiterkarte aktiviert und versucht die WADL-Datei darzustellen, sobald eine gültige Auswahl einer Ressource in der Tabelle der erkundeten Geräte vorliegt. Es wird an dieser Stelle nicht überprüft, ob die Ressource eine WADL-Datei anbietet. Sollte keine Datei gefunden werden, so wird eine „java.io.FileNotFoundException“ geworfen und das WADL-Panel bleibt leer. Zum anderen implementiert die gegenwärtige Version der Software noch nicht die tatsächliche Übertragung der WADL-Dateien vom Server zum Client. Stattdessen sind die WADL-Dateien schon auf der Client-Seite hinterlegt. Zur Simulation der Übertragung übermittelt der Server lediglich die Adresse der WADL-Datei als URI im IF-Attribut der Ressource. Der Client nutzt den letzten Teil der URI aus der IF – den Pfad der Ressource – als relativen Pfad von einem ihm bekannten Verzeichnis seiner Festplatte aus. So ermittelt er den Speicherort der bereits hinterlegten WADL-Datei und die Beschreibung kann eingelesen und in der GUI dargestellt werden. Da der Pfad der Ressource aber im oben beschriebenen Vorgehen identisch bleibt, genügt das, damit der Client die richtige Speicherstelle der bestehenden WADL-Beschreibung findet und diese kann trotz nicht mehr definierter IF angezeigt werden. Für die grafische Umsetzung der Informationen aus der WADL-Datei ist es nicht wesentlich, an welcher Stelle die Datei gespeichert ist. Daher wurde die Implementierung zurückgestellt und noch nicht vorgenommen. Logisch betrachtet muss natürlich ein Server die Schnittstellenbeschreibungen für unbekannte Ressourcen anbieten, andernfalls

wären die Ressourcen dem Client schließlich nicht unbekannt und er müsste diese nicht mehr erkunden. Dieser simulierte Fernzugriff ist in der weiteren Entwicklung des Editors noch zu ersetzten durch eine reale Übertragung. Hierbei erlangt ein wesentlicher Fakt große Bedeutung. Das ist die Größe der zu übertragenden WADL-Datei. Wird diese zu groß, so müssen die Informationen via Block-Transfer auf mehreren Datenpakete verteilt transportiert werden. Das reduziert die Netzwerkperformance des CoAP-Netzwerkes und ist daher möglichst zu vermeiden.
Da in einem realen Einsatzfall der CoAP-Client und der CoAP-Server auf verschiedenen Netzwerkknoten laufen, wäre eine an diesen Sachverhalt angelehnte Testumgebung von großem Nutzen. Eine solche Netzwerkstruktur lag zum Zeitpunkt der Tests jedoch nicht vor. Bei der Erstellung von virtuellen Maschinen zum Erzeugen eines simulierten Netzwerkes scheiterte die Netzwerkkonfiguration. Daher konnten Test nur auf einem Rechner durchgeführt werden. Sowohl CoAP-Server, als auch CoAP-Client waren dabei über die „localhost“ IP-Adresse 127.0.0.1 erreichbar. Weitere Tests mit separaten Netzwerkknoten für Client und Server stehen noch aus. Auf Grund der mangelnden Netzwerk-Infrastruktur fehlte die Grundlage für Multicast-Discovery, welches folglich noch nicht Implementiert ist. Hierfür müssen Server, statt wie in der vorliegenden Version des Corn Editor Unicast-Netzwerksockets, nun Multicast-Sockets nutzen und sich alle in einer bekannten Multicast-Gruppe registrieren. Für IPv4 Netzwerke müssen alle CoAP-Knoten eine Adresse aus dem Block 224.0.1.x erhalten [3, vgl. S. 78 f.]. Der Client versendet Discovery-Anfragen dann an diese Multicast-Gruppenadresse und kann hierüber alle Ressourcen aller Server der Multicast-Gruppe erkunden.
Zusammenfassend ergaben die Tests des Corn Editor, dass eine Unicast-Erkundung der Test-Ressourcen auf einem Server mit bekannter Adresse funktionieren. Einzelne erkundete Ressourcen lassen sich auf der GUI in einer Tabelle auswählen und verschiedene Reiterkarten liefern Detailansichten zu der ausgewählten Ressource. Sollte keine Ressource ausgewählt oder erkundet worden sein, bzw. eine erkundete Ressource keine Methoden anbieten oder Methoden ohne Parameter anbieten, so wird der Nutzer des Corn Editors darüber informiert. Schlägt das Einlesen einer Schnittstellenbeschreibung für eine Ressource fehl, so wird der Nutzer nicht direkt informiert. Das WADL-Panel bleibt in diesem Fall leer und die „Methods & Parameters“-Darstellung wird nicht aktualisiert. Die Darstellung einer variablen Anzahl an „request“- bzw. „response“- Parametern wird dynamisch zur Laufzeit für die einzelnen Methoden von Ressourcen entsprechend der zugehörigen Schnittstellenbeschreibung realisiert. Parameter für Anfragen und Antworten werden farblich unterschieden und es existieren Eingabefelder zur Eingabe von Werten für die Parameter durch den Nutzer. Weitere Unterscheidungen der Parameter nach Datentyp, Notwendigkeit etc. findet noch nicht statt. Die angebenen Query-Parameter werden beim Erstellen der Anfrage URI korrekt berücksichtigt, jedoch werden Nutzereingaben nicht auf Gültigkeit überprüft und das Verhalten des Testservers reagiert nicht auf die Belegung der Werte. Unterschiedliche Anfragen lassen sich an einen Server senden und die erhaltenen Antworten liefern entsprechende Antwort-Codes, welche im Corn Editor angezeigt werden. Sollte die Antwort eine Content-Type Option enthalten, wird

ihr Wert auf der GUI ausgegeben. Unterstützung für weitere Optionen ist noch in späteren Versionen zu integrieren. Das Setzten von Payload-Inhalt für Anfragen und das Betrachten der Payload einer Antwort ist bereits umgesetzt. Die einzelnen GUI Komponenten lassen sich dynamisch in ihrer Größe ändern und liefern bei Bedarf Scrollbalken. Einzelne Komponenten lassen sich zwar so klein ziehen, dass sie aus dem Sichtbereich verschwinden, können jedoch auch wieder vergrößert werden. Die Darstellung der erkundeten Ressourcen auf der GUI ist nach ausführen einer Methode ggf. nicht aktuell und auf gelöschten Methoden können noch Methoden aufgerufen werden, welche an gleicher Adresse eine neue Ressource erstellen. Zur Aktualisierung der Ansicht ist daher gegenwärtig eine erneute Erkundung nötig.

5.2. Tauglichkeitsanalyse für WADL als Schnittstellenbeschreibungssprache für CoAP-Ressourcen

In diesem Abschnitt soll abschließend der Frage nachgegangen werden, inwiefern sich WADL als Schnittstellenbeschreibung für CoAP-Geräte eignet. WADL wurde laut Spezifikation designt, um HTTP basierte Webanwendungen zu beschreiben [12, vgl. Kap. 1]. CoAP setzt einen Teil von HTTP für M2M-Anwendungen um [3, vgl. S. 5]. Das Projekt Corn Editor zeigt, dass sich das Konzept der Beschreibung von Ressourcen durch WADL in großen Teilen von HTTP auf CoAP übertragen lässt. Einzelne in WADL spezifizierte Aspekte, wie z. B. der Parameter-Style „header“, welcher die Nutzung eines konkreten HTTP-Headers verlangt, sind nicht für CoAP anwendbar, können aber außen vor gelassen werden.
CoAP nutzt zwar zur Erkundung von Ressourcen das Core Link-Format, jedoch kann ergänzend hierzu mittels WADL die Schnittstelle einer Ressource beschrieben werden. Dazu wird das Interface-Description Attribut des Core Link-Formates genutzt, um durch eine URI auf die beschreibende WADL-Datei zu verweisen. Nach Erkundung der Ressource kann ein Client an der im IF beschriebenen Adresse die WADL-Beschreibung eines Gerätes anfragen.
„Verfügt ein Client über Zugriff zu einer WADL-Beschreibung, so genügt dies bereits um die darin beschriebenen Ressourcen nutzen zu können. Weiteres implizites Wissen ist nicht nötig, da Anhand einer ausführlichen WADL-Beschreibung sämtliche beschriebenen Ressourcen sowie deren Handhabung verdeutlicht werden. Konkrete Anfragen an Ressourcen, der Aufbau dieser Anfragen sowie zu erwartende Ergebnisse werden beschrieben. Ein Anwendungsübergreifendes Verständnis der Bezeichnungen in einer WADL-Beschreibung wird durch das Bekanntgeben genutzter Namensräume erzielt. Die Beschreibung eines Webservices durch WADL ermöglicht es, benötigte Datentypen entsprechend der Bedürfnisse zu definieren. So lassen sich auch komplexe Parameter in Anfragen realisieren.“ [11, vgl. S. 26] Für die Definition solcher Datentypen und Re-

präsentationsformate können XML-Schemata genutzt werde, auf welche eine WADL-Beschreibung verweisen kann [12, vgl. Kap. 1.2].
Zu den Vorteilen von Schnittstellenbeschreibungen durch WADL zählt die maschinenverständlichkeit der XML-basierten Sprache WADL, welche auch gut menschenlesbar ist. Eine zentrale Registrierung aller durch CoAP-Clients genutzter Ressourcen ist nicht nötig, wenn die CoAP-Service-Provider maschinenverständliche WADL-Beschreibungen für die angebotenen Ressourcen zur Verfügung stellen. Ein Client kann in einem dynamischen Netzwerk auch bei neu hinzukommenden Geräten automatisch erlernen, welche Schnittstellen die neue Ressource anbietet und wie diese zu nutzen ist. Durch CoAP-Multicast-Resource-Discovery können Schnittstellen von Ressourcen verschiedenster Server erkundet werden. Zudem erlaubt WADL eine flexible Entwicklung von CoAP-Clients, welche sich dynamisch an den in WADL-Beschreibungen enthaltenen Informationen orientieren kann. Sollte sich die Ressource auf einem Server ändern, so genügt die Anpassung der WADL-Schnittstellenbeschreibung, um die Bedienbarkeit der neuen Ressource durch den Client zu ermöglichen. Dadurch wird eine unabhängige Entwicklung von Client und Server entsprechend der REST Designkriterien (siehe Kapitel 2.1.1) unterstützt.
Die Entwicklung eines CoAP-Clients, welcher die Schnittstellen erkundeter Ressourcen lediglich an Hand von WADL-Beschreibungen der Ressourcen erlernen und bedienen kann, ist jedoch nicht zu empfehlen. CoAP garantiert nicht, dass eine Schnittstellenbeschreibung von Ressourcen durch WADL existiert. Das IF-Attribut des CLF kann mehrere Beschreibungen von Ressourcen enthalten und es ist nicht ausgeschlossen, dass keine WADL-Beschreibung angeboten wird. Ein Client, welcher sich ganz auf die Analyse der Schnittstellen durch WADL verlässt, kann solche Ressourcen dann nicht automatisch erkunden und nutzen.
WADL bietet zwar die Möglichkeit zu beschreiben, welche Methoden durch eine Ressource angeboten werden, mit welchen Parametern Anfragen an diese Ressource belegt werden können und welche Datentypen dabei zu nutzen sind, aber es besteht keine Möglichkeit, durch WADL einem Client die Ausführung einer Methode zu untersagen. Möchte ein Server eine Ressource mehreren Clients zur Verfügung stellen, jedoch mit unterschiedlichem Methodenumfang, so muss dieser Server unterschiedliche WADL-Dateien anbieten. Zudem kann ein Client auch Methoden auf eine Ressource anwenden, welche nicht in der WADL-Beschreibung aufgeführt sind. Davor schützt das Auslassen der Beschreibung einer Methode in einer WADL-Datei nicht. Es muss also bei Bedarf vom Server anderweitig sichergestellt werden, dass Anfragen von nicht in der WADL-Datei beschriebenen Methoden entsprechend behandelt werden. Ein Client muss zudem darauf vertrauen, dass die WADL-Beschreibung mit der Implementierung der beschriebenen Ressource überein stimmt. Es besteht nicht die Möglichkeit, die Integrität von beschriebenem und implementiertem Verhalten einer Ressource aus Client-Sicht zu überprüfen. Schlägt die Bedienung einer Ressource entsprechend der WADL-Beschreibung fehl, so ist diese Ressource nicht durch den Client nutzbar.
Mit WADL kann beschrieben werden, welche Optionen eine Antwort enthält und in welchem Datenformat diese Antwort zu erwarten ist. Sollte eine Ressource z. B. meh-

rere Response-Parameter anbieten, so ist in einer WADL-Beschreibung nicht festgelegt, in welcher Reihenfolge die Parameter in der Antwort enthalten sind. Handelt es sich bei dem Parameterwert um eine Angabe, die der Server per CoAP-Option in die Antwort packt, ist die Reihenfolge durch die aufsteigende Nummer der CoAP-Optionen eingegrenzt. Handelt es sich um Informationen, die in der Payload des Antwortpaketes übermittelt werden, so muss der Client andere Möglichkeiten finden, die Reihenfolge der erhaltenen Informationen zu ermitteln. Eine Möglichkeit dieses Problem anzugehen ist es, die Payload als XML Datei zu übertragen, deren Schema die Inhalte und Reihenfolgen der Inhalte festlegt. Solche Schema-Definitionen lassen sich in eine WADL-Datei integrieren.
Die Größe der WADL-Beschreibung spielt für die Eignung in CoAP-Netzwerken eine besonders wichtige Rolle. CoAP als Protokoll für ressourcelimitierte Netzwerkknoten legt einen besonderen Wert darauf, Paketfragmentierung wenn möglich zu vermeiden, um die Netzwerkperformance hoch zu halten. Daher sollten zu übertragende Dateien möglichst nicht größer als die maximale Übertragungseinheit (engl. Maximum Transmission Unit (MTU)) der Verbindung sein. Das gesamte CoAP-Paket sollte also in ein einzelnes UDP-Paket passen, welches wiederum in ein IP-Paket passen sollte. Je nach Übertragunsmedium und Headergrößen der genannten Protokolle empfiehlt die CoAP-Spezifikation, dass ein CoAP-Paket 1280 Byte nicht überschreitet [3, vgl. S. 21]. Größere Pakete müssen in Block Transfer übertragen werden, wodurch sich der Anteil der Nutzdaten am Gesamtdatenverkehr verringert und somit die Performance des Netzwerkes schmälert. Die WADL-Beschreibungen der Test-Ressourcen aus Anhang A.5 sind zwischen 463 und 1593 Byte groß. Bei Übertragung der CoAP-Pakete via Ethernet mit typischer MTU von 1500 Byte [20, vgl. S. 2] und einem UDP-Header von 12 Byte [21, vgl. S. 2] passen die größeren Beschreibungen schon nicht mehr in ein einzelnes Paket.
Werden WADL-Beschreibungen für CoAP-Ressourcen genutzt, sollte daher darauf geachtet werden, die Beschreibungen möglichst klein zu halten. Datentypen, Media-Types und Repräsentationsformate für Ressourcen können in WADL direkt definiert werden oder die WADL-Datei enthält einen Verweis auf eine entsprechende Definition. Werden Verweise statt direkter Definitionen genutzt, so kann die WADL-Datei kleiner gehalten werden. Der Client muss jedoch zur Ermittlung der Definition weitere Netzwerkanfragen absenden, sofern die Definitionen ihm nicht bekannt sind. Netzwerkknoten mit sehr kleinem Speicher sind ggf. nicht in der Lage, eine umfangreiche WADL-Datei zu speichern. In diesem Fall könnten diese Netzwerkknoten nicht von allen Schnittstellenbeschreibungen profitieren. Bietet ein Server WADL-Dateien für CoAP-Ressourcen an, so sollten daher kleine Beschreibungen für einzelne Ressourcen großen und mehrere Ressourcen umfassenden Beschreibungen vorgezogen werden.

Zusammenfassend lässt sich also festhalten, dass WADL als Schnittstellenbeschreibung für CoAP-Geräte in automatisierten Netzwerken viele Vorteile bietet. Clients können per M2M-Kommunikation durch WADL die Bedienung neu erkundeter Ressourcen erlernen. Die Definition von Datentypen und verwendeten Parametern ist möglich und

unterstützt die individuelle Verwendung von WADL-Beschreibungen. Womöglich muss ein Netzwerkknoten die Reihenfolge der erhaltenen Parameter ohne Beschreibung ermitteln können. Die Anzahl und Art der übermittelten Parameter sind jedoch aus der WADL-Beschreibung ermittelbar. Sollten die Schnittstellenbeschreibungen sehr umfangreich sein, führen sie ggf. zu unerwünschter Verschlechterung der Netzwerkauslastung. Server sollten also kleine WADL-Beschreibungen einzelner Ressourcen anbieten. WADL-Schnittstellenbeschreibungen sind jedoch nicht gut geeignet, um unterschiedlichen Clients einen individuellen Funktionsumfang zu gewährleisten. Hierfür werden andere Mechanismen benötigt.

6. Zusammenfassung

Die Vorliegende Arbeit dokumentiert die Spezifikation und Implementierung eines Entwickler-Tools zum Erkunden und Testen von Ressourcen eines CoAP-Netzwerkes. Das hierbei genutzte CoAP-Protokoll implementiert die REST-Architektur für verteilte hypermediale Systeme unter besonderer Berücksichtigung der Einschränkungen ressourcenlimitierter Netzwerkknoten, wie sie im Umfeld der Gebäudeautomation und in Sensornetzwerken oft auftreten. Jede Komponente, welche Ziel einer Adressierung sein könnte, wird im REST-Architekturstiel als Ressource aufgefasst und kann über das einheitliche Interface von REST bedient werden. Dabei stützt sich CoAP im Wesentlichen auf die vier von HTTP bekannten Verben „GET“, „POST“, „PUT“ und „DELETE“, um Repräsentationen von Ressourcen anzufragen, diese zu aktualisieren, neue Ressourcen anzulegen oder bestehende zu entfernen.
Darüber hinaus ermöglicht CoAP die maschinelle Erkundung unbekannter Ressourcen in einem Netzwerk unter Nutzung des CoRE-Link-Format-Discovery-Mechanismus. Einzelne Netzwerkteilnehmer werden somit in die Lage versetzt, alle verfügbaren Ressourcen in ihrer Umgebung ohne Interaktion von außen ausfindig zu machen. Über die maschinenverständliche Beschreibungssprache WADL können die Ressourcen via M2M-Kommunikation Schnittstellenbeschreibungen für die durch sich angebotenen Ressourcen austauschen. Anhand dieser Beschreibungen können CoAP-Clients die Benutzung unbekannter Ressourcen erlernen. WADL-Beschreibungen dokumentieren Ressourcen und die darauf anwendbaren Methoden. Durch solche Beschreibungen kann dokumentiert werden, welche Anfragen sich wie parametrisieren lassen, welche Datentypen und Repräsentationen für Parameter und Ressourcen verwendet werden und welche Informationen in Antworten zu gültigen Anfragen enthalten sind. Somit können Geräte unterschiedlicher Hersteller sich automatisch gegenseitig erkunden und Vernetzen. Durch das von CoAP angebotene Multicast-Discovery ist dies sogar in dynamisch veränderten Netzwerken und ohne eine zentrale Registrierung aller im Netzwerk verfügbarer Ressourcen möglich.
Das im Rahmen der Arbeit entstandene Entwickler-Tool – Corn Editor – implementiert die Erkundung von CoAP-Ressourcen unter Nutzung von WADL-Schnittstellenbeschreibungen und stellt die Ergebnisse auf einer GUI grafisch dar. Es soll Entwicklern von CoAP-Geräten eine Unterstützung beim Test der durch sie entworfenen Komponenten bieten. Zudem soll es dabei helfen, CoAP als einheitlichen Standard zur maschinellen Erkundung von Ressourcen verschiedener Hersteller in ressourcenlimitierten Netzwerken voranzutreiben.

7. Ausblick

Die Entwicklung des Corn Editors ist ein erster Schritt zur intensiven Anwendung von WADL-Dateien als Schnittstellenbeschreibung unbekannter Ressourcen in RESTful CoAP-Netzwerken. Das Entwickler-Tool beherrscht bereits die Erkundung von Ressourcen via Unicast-Discovery und kann vorhandene WADL-Schnittstellenbeschreibungen einer Ressource in grafische Komponenten umsetzten. Ein erster Nachweis der Tauglichkeit von WADL zur Schnittstellenbeschreibung unbekannter Ressourcen ist somit erbracht.
Damit Entwickler von CoAP-Ressourcen das Verhalten ihrer Produkte jedoch umfassend und unter realitätsnahen Bedingungen testen können, sind noch viele Verbesserungen des Tools nötig. Neben der Integration von Multicast-Discovery zur automatischen Erkundung von Ressourcen mehrerer Server steht zudem die Implementierung der tatsächlichen Übertragung vorhandener Schnittstellenbeschreibungen von Server zu Client aus. Diese beiden Aspekte stellen eine zentrale Aufgabe der weiteren Entwicklung des Corn Editor dar. Erst mit diesen Funktionalitäten können in Testnetzwerken alle vorhandenen Ressourcen ohne Vorkenntnisse über Server-IP-Adressen erkundet und konkrete Analysen über den Einfluss der Übertragung unterschiedlich großer WADL-Beschreibungen auf die Netzwerkauslastung durchgeführt werden.
Zum gegenwärtigen Entwicklungsstand können zwar einzelne erkundete Methoden auf Ressourcen ausgeführt werden, jedoch ist eine Aktualisierung der grafischen Darstellung der Ressourcen nur durch eine komplett neue Erkundungsanfrage möglich. Eine Möglichkeit zur Aktualisierung der Darstellung einzelner Ressourcen fehlt dem Corn Editor aktuell noch.
Die Erstellung grafischer Komponenten zur Repräsentation der Parameter für Anfragen und Antworten im Corn Editor ist schon umgesetzt worden, jedoch wurden einzelne Attribute dieser Parameter noch nicht umfassend berücksichtigt. So existiert z. B. noch kein Vergleich eingegebener Parameterwerte mit dem jeweilig gültigen Wertebereich oder eine Unterscheidung zwischen optionalen und zwingend benötigten Parametern und die Unterstützung anderer Antwort-Parameter als die der „Content-Type“-Option stehen ebenfalls noch aus.
Ressourcen können über das IF-Attribut des CoRE-Link-Formates zwar auf ihre Schnittstellenbeschreibungen verweisen, es ist im CoAP-Protokoll jedoch nicht vorgeschrieben, dass darin eine WADL-Beschreibung enthalten sein muss. Das Verhalten des Corn-Editors im Falle nicht vorhandener WADL-Beschreibungen ist gegenwärtig noch nicht spezifiziert. Eine entsprechende Überprüfung und daraus resultierendes Verhalten des Corn Editors könnten in Zukunft noch umgesetzt werden.

Zudem ist es denkbar, die Funktionalität des Corn Editors noch zu erweitern und dadurch CoAP-Entwicklern noch umfassendere Möglichkeiten zu bieten, ihre Geräte zu testen. Interessante, derzeit nicht realisierte Funktionalitäten wären u.a. Syntaxhervorhebung für angezeigte WADL-Beschreibungen, eine Steuermöglichkeit zur Versendung von Nachrichten als Confirmable bzw. Non-Confirmable, die Option, neue Ressourcen auf einem Server vom Corn Editor aus anzulegen oder gar die Möglichkeit, Anfragen zu Testzwecken an Ressourcen zu senden, welche nicht in der Interfacebeschreibung aufgeführt sind.
Erste Funktionalitäten sind also bereits umgesetzt, jedoch sind noch weitere wichtige Bausteine wie die Erkundung von Ressourcen in Multicast-Gruppen und eine Übertragung der Schnittstellenbeschreibungen von Server zu Client nachzuholen. Der Corn Editor bietet damit eine erste Grundlage für ein noch auszureifendes Softwareprodukt zum Testen von CoAP-Geräten und der Interaktion mit diesen über die durch WADL beschriebenen Schnittstellen erkundeter Ressourcen.

Literaturverzeichnis

[1] CoRE Working Group. Constrained application protocol (coap) draft-ietf-core-coap-18. IETF, core WG, Jun 2013. Proposed Standard.

[2] Roy T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, Irvine - Irvine, CA 92697, USA, 2000.

[3] CoRE Working Group. Constrained application protocol (coap) draft-ietf-core-coap-11, Jul 2012. Proposed Standard.

[4] Simens Aktiengesellschaft. Das siemens-umweltportfolio beispiele für nachhaltige technologien. pdf on Company‘s website. Bestellnummer L2-Z800.

[5] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann. Restful web services vs. big web services: Making the right architectural decision. In *17th International World Wide Web Conference (WWW2008)*, pages 805–814, Beijing, China, April 2008 2008.

[6] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. Technical report, Sun Microsystems Laboratories, Inc., Nov. 1994.

[7] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O’Reilly Media, 1 edition, May 2007.

[8] A. Heil. *Anwendungsentwicklung Für Intelligente Umgebungen Im Web Engineering*. Springer Vieweg. in Springer Fachmedien Wiesbaden GmbH, 2012.

[9] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform resource identifier (URI): Generic syntax. RFC 3986, RFC Editor, Fremont, CA, USA, January 2005.

[10] Z. Shelby. Constrained RESTful Environments (CoRE) Link Format. RFC 6690 (Proposed Standard), August 2012.

[11] Simeon Wiedenmann. Analyse der resizrce discovery-mechanismen in automatisierungsnetzten. Mai 2013.

[12] Marc Hadley. Web application description language. WC3 Member Submission, August 2009. http://www.w3.org/Submission/wadl/.

[13] W3C Frank Manola, Eric Miller. Rdf primer. WC3 Recommendation, February 2004. http://www.w3.org/TR/rdf-primer/.

[14] WS4D initiative. Jcoap repository. https://code.google.com/p/jcoap/, April 2011.

[15] Reihaneh Rabbany. Wadlshallowparser. http://webdocs.cs.ualberta.ca/ rabbany-k/research/WS/, March 2011.

[16] Apache license, version 2.0. http://www.apache.org/licenses/LICENSE-2.0, January 2004.

[17] Reihaneh Rabbany. Wadlshallowparser doc. http://webdocs.cs.ualberta.ca/ rabbanyk/research/WS/doc/, March 2011.

[18] Reihaneh Rabbany. Wadlshallowparser license. http://webdocs.cs.ualberta.ca/ rabbanyk/research/WS/license.txt, March 2011.

[19] Java standard edition development kit. http://www.oracle.com/technetwork/java /javase/downloads/jdk7-downloads-1880260.html. Linux x86 - jdk-7u40-linux-i586.tar.gz.

[20] C. Hornig. A Standard for the Transmission of IP Datagrams over Ethernet Networks. RFC 894 (INTERNET STANDARD), April 1984.

[21] J. Postel. User Datagram Protocol. RFC 768 (INTERNET STANDARD), August 1980.

A. Anhang

A.1. Klassendiagramme

A.1.1. Klasse CornCoAPClient

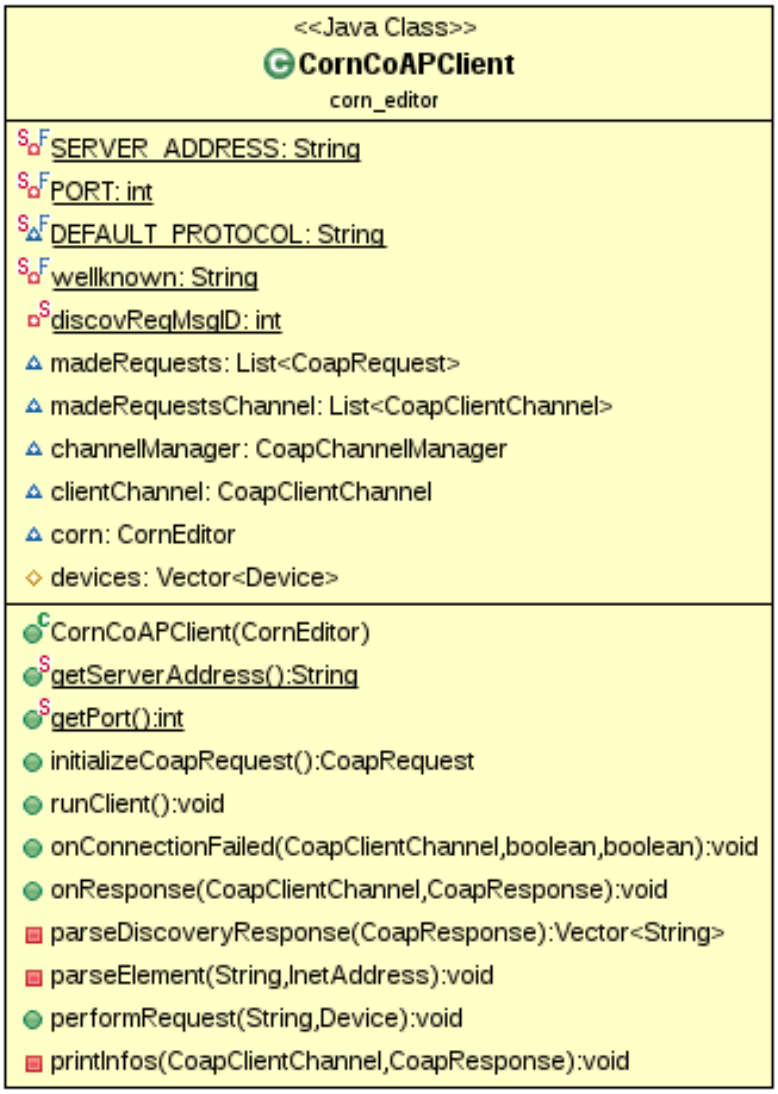

Abbildung A.1.: Klassendiagramm der CornCoAPClient Klasse.

A.1.2. Die Klassen CornEditor & RowListener

Abbildung A.2.: Klassendiagramm der CornEditor Klasse.

A.1.3. Klasse Device

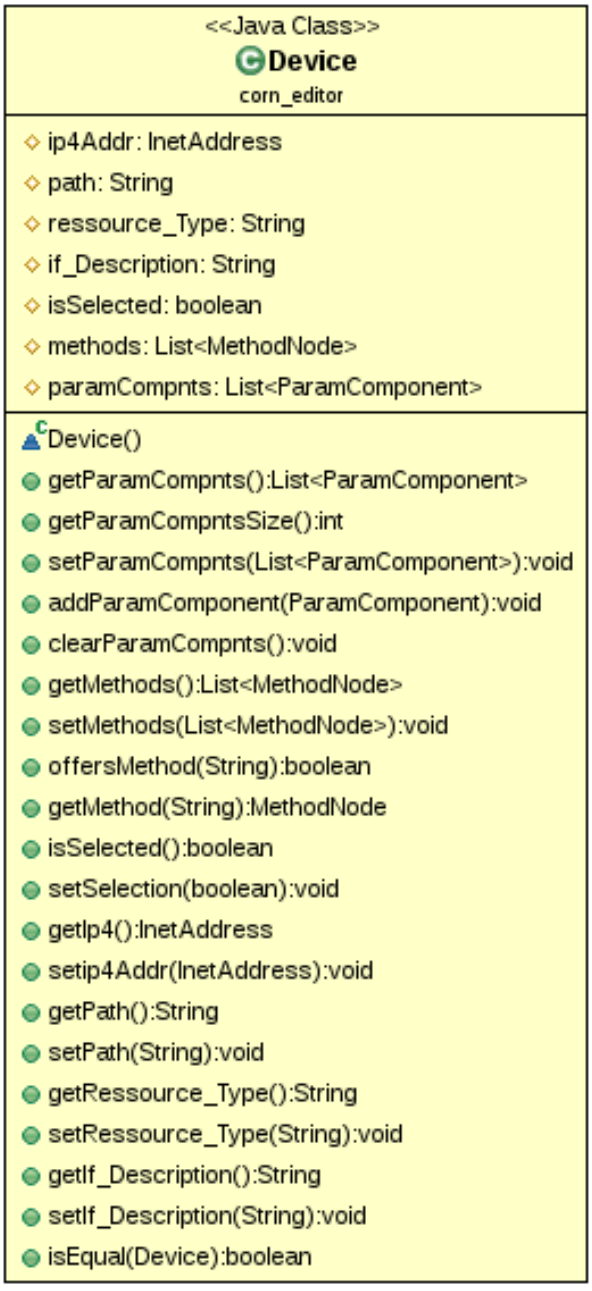

Abbildung A.3.: Klassendiagramm der Device Klasse.

A.1.4. Klasse ParamComponent

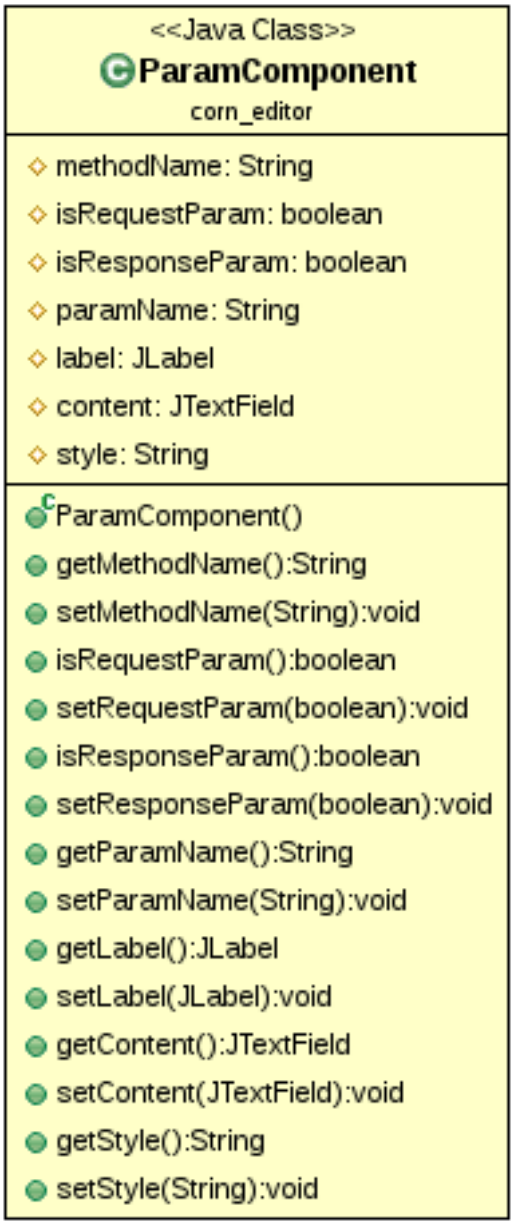

Abbildung A.4.: Klassendiagramm der ParamComponent Klasse.

A.1.5. Klasse CoapServerExampleRessources

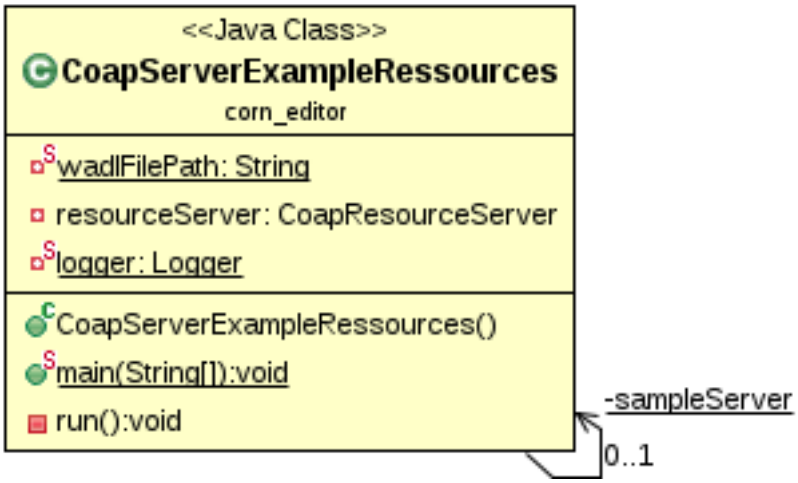

Abbildung A.5.: Klassendiagramm der CoapServerExampleRessources Klasse.

A.1.6. Corn Editor

Abbildung A.6.: Klassendiagramm des Corn-Projektes.

A.2. Call-Graphen

Die Grafiken in diesem Abschnitt entsprechen einem älteren Implementierungsstand der Software. Die wesentlichen Unterschiede bestehen darin, dass die Funktion „runTest-Client()“ nun „runClient()“ heißt und die „performeRequest2()“ umbenannt wurde in „performeRequest()“. Diese Namensänderungen ändern jedoch nicht den wesentlichen Inhalt der Darstellungen und werden daher dennoch aufgeführt.

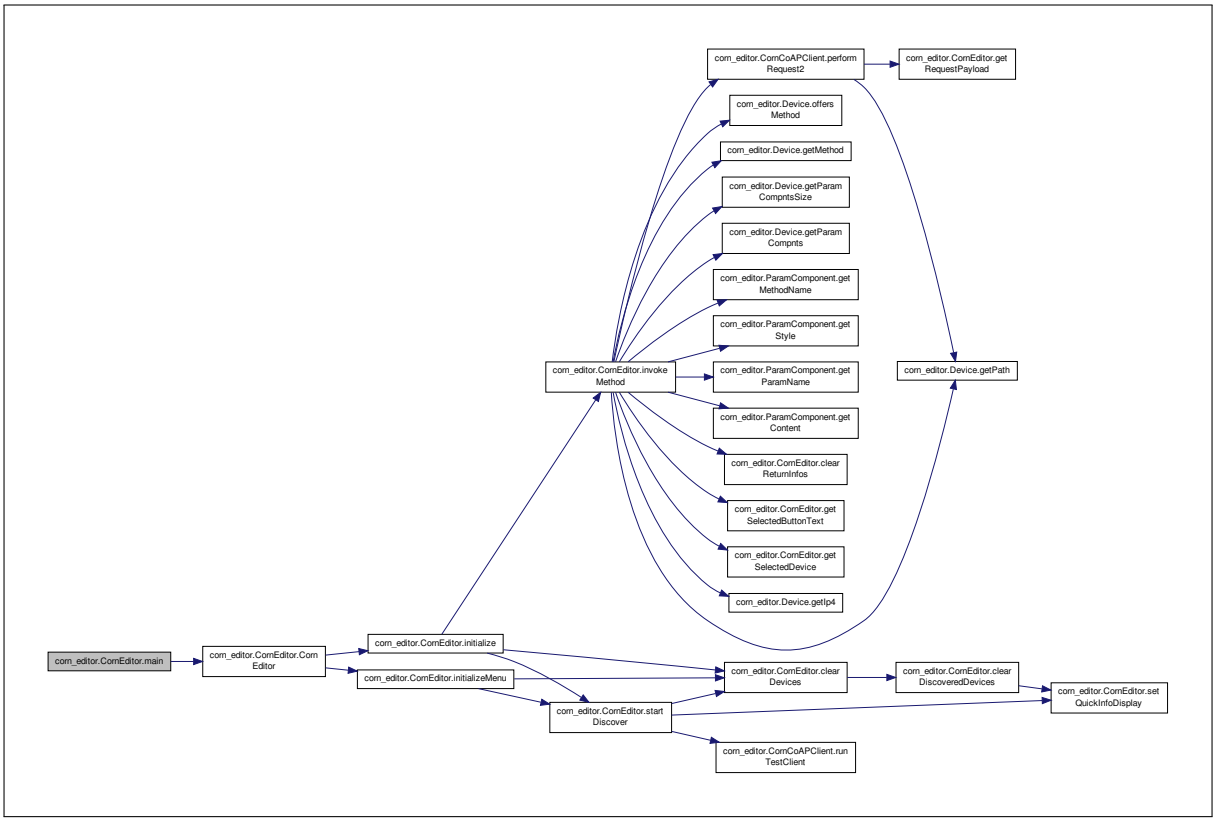

Abbildung A.7.: Call-Graph der Hauptrutine main.

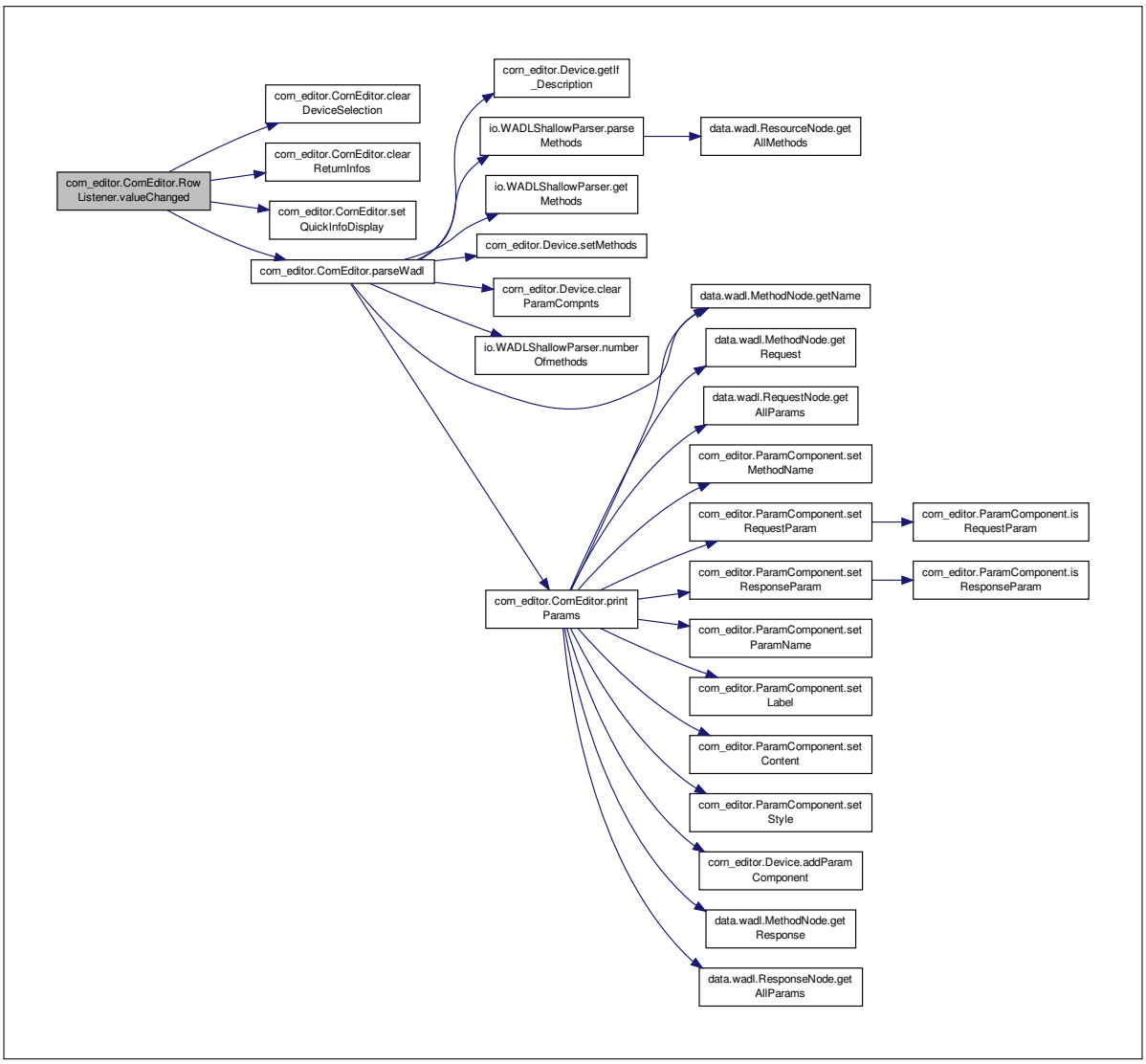

Abbildung A.8.: Call-Graph der Funktion „valueChanged()“ aus der Klasse „RowListener“.

A.3. CoAP-Details

A.3.1. CoAP-Response Codes

Code	Description
2.01	Created
2.02	Deleted
2.03	Valid
2.04	Changed
2.05	Content
3.00 - 3.31	Reserved
4.00	Bad Request
4.01	Unauthorized
4.02	Bad Option
4.03	Forbidden
4.04	Not Found
4.05	Method not Allowed
4.06	Not Acceptable
4.12	Precondition Failed
4.13	Rewuest Entity Too Large
4.15	Unsupported Content-Format
5.00	Internal Server Error
5.01	Not Implemented
5.02	Bad Gateway
5.03	Service Unavailable
5.04	Gateway Timeout
5.05	Proxying Not Supported
6.00 - 7.31	Reserved

Tabelle A.1.: CoAP-Response-Codes [1, vgl. Tabelle 6, S. 86].

A.3.2. CoAP-Optionen-Nummern

CoAP Option Nummer	Version 11 Options-Name	Version 18 Options -Name
0	Reserviert	Reserviert
1	Content-Type	If-Match
2	Max-Age	nicht zugewiesen
3	Proxy-Uri	Uri-Host
4	ETag	ETag
5	Uri-Host	If-Non-Match
6	Location-Path	nicht zugewiesen
7	Uri-Port	Uri-Port
8	Location-Query	Location-Path
9	Uri-Path	nicht zugewiesen
11	Token	Uri-Path
12	Accept	Content-Format
13	If-Match	nicht zugewiesen
14	nicht zugewiesen	Max-Age
15	Uri-Query	Uri-Query
17	nicht zugewiesen	Accept
20	nicht zugewiesen	Location-Query
21	If-Non-Match	nicht zugewiesen
35	nicht zugewiesen	Proxy-Uri
39	nicht zugewiesen	Proxy-Scheme
44	Reserviert	nicht zugewiesen
46	Reserviert	nicht zugewiesen
48	Reserviert	nicht zugewiesen
60	nicht zugewiesen	Size1
128	nicht zugewiesen	Reserviert
132	nicht zugewiesen	Reserviert
136	nicht zugewiesen	Reserviert
140	nicht zugewiesen	Reserviert

Tabelle A.2.: CoAP-Optionen-Nummern nach Version 11 [3, vgl. Tabelle 4, S. 73] bzw. 18 [1, vgl. Tabelle 7, S. 86 f.].

A.4. Corn Editor Shortcuts

Hotkey	Description	Constraints
ALT + A	open „About“ infobox	inside „Help“ menu
ALT + C	Clear all discovered devices	inside „View“ menu and global
ALT + D	select „DELETE“ method	on active „Methods & Parameter“ tap
ALT + E	Exit application	inside „File“ menu
ALT + F	open „File“ menu	
ALT + G	select „GET“ method	on active „Methods & Parameter“ tap
ALT + H	open „Help“ menu	
ALT + I	klick button „Invoke Method“	on active „Methods & Parameter“ tap
ALT + L	open „License Information“ infobox	inside „Help“ menu
ALT + M	display „Methods & Parameter“ pannel	of selected resource
ALT + N	display Corn Editor info text	inside „Help“ menu
ALT + O	select „POST“ method	on active „Methods & Parameter“ tap
ALT + P	display „Payload“ pannel	of selected resource
ALT + Q	display „Quick Info“ pannel	of selected resource
ALT + S	Start resource discovery	inside „File“ menu and global
ALT + U	select „PUT“ method	on active „Methods & Parameter“ tap
ALT + V	open „View“ menu	
ALT + W	display „WADL“ pannel	of selected resource
ESC	ESCape open menu	

Tabelle A.3.: Corn Editor Shortcuts.

A.5. WADL-Beschreibungen der Test-Ressourcen

```
<?xml version="1.0"?>
<application  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://wadl.dev.java.net/2009/02">
  <!-- XML namespace for all WADL elements -->

  <doc xml:lang="EN" title="An example wadl description of a CoAP Resource
    simulating a temperature sensor"/>

  <!-- Server hosting the resource -->
  <resources base="coap://127.0.0.1/">
      <!-- relative path to resource on server -->
    <resource path="test/temperature">

      <!-- method offered by the resource -->
      <method name="GET">
        <request>
          <!-- specifying request paramter -->
          <param name="unit" type="xsd:string"
               style="query" required = "true" default="Celsius">
                 <option value="Celsius"/>
                 <option value="Fahrenheit"/>
          </param>
        </request>

        <response>
          <!-- specifying response paramter -->
          <param name="Content-Type" type="xsd:unsignedInt">
          </param>
        </response>
      </method>

      <!-- method offered by the resource -->
      <method name="DELETE">
      </method>
    </resource>

  </resources>
</application>
```

Listing A.1: WADL-Beschreibung der Test-Ressource „temperature“.

```
<?xml version="1.0"?>
<application  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://wadl.dev.java.net/2009/02">
  <!-- XML namespace for all WADL elements -->

  <doc xml:lang="EN" title="An example wadl description of a CoAP Resource
    simulating a light sensor"/>

  <resources base="coap://127.0.0.1/test/">

    <resource path="light">
      <method name="GET">
        <request>
          <param name="unit" type="xsd:string"
              style="query" fixed="Lux">
          </param>
          <param name="decPlaces" type="xsd:integer"
              style="query" required="false">
          </param>
        </request>
      </method>
      <method name="POST">
        <request>
          <param name="postParam1" type="xsd:string"
              style="query" default="" required="false">
          </param>
        </request>
      </method>
      <method name="PUT">
        <request>
          <param name="resourceType" type="xsd:string"
              style="query" required="false">
          </param>
          <param name="interfaceDescription" type="xsd:string"
              style="query" required="false">
          </param>
          <param name="Content-Type" type="xsd:unsignedInt"
              style="query" fixed="0">
          </param>
        </request>
        <response>
          <param name="putResponse1" type="xsd:string">
          </param>
          <param name="putResponse2" type="xsd:string">
          </param>
        </response>
      </method>
      <method name="DELETE">
      </method>
    </resource>

  </resources>
</application>
```

Listing A.2: WADL-Beschreibung der Test-Ressource „light“.

```
<?xml version="1.0"?>
<application  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://wadl.dev.java.net/2009/02">
  <!-- XML namespace for all WADL elements -->

  <doc xml:lang="EN" title="An example wadl description of a CoAP Resource
    simulating a humidity sensor"/>

  <resources base="coap://127.0.0.1/test/">

    <resource path="humidity">
      <method name="GET">
        <request>
        </request>
        <response>
          <param name="Content-Type" type="xsd:unsignedInt">
          </param>
        </response>
      </method>

      <method name="DELETE">
      </method>
    </resource>

  </resources>
</application>
```

Listing A.3: WADL-Beschreibung der Test-Ressource „humidity“.

```
<?xml version="1.0"?>
<application xmlns="http://wadl.dev.java.net/2009/02">
<!-- XML namespace for all WADL elements -->

  <doc xml:lang="EN" title="An example wadl description of a CoAP Resource
    that does not offer any CoAP method"/>

  <resources base="coap://127.0.0.1/test/">

    <resource path="noMethod">
      <!-- This Resource does not offer any CoAP method in it's
          interface describing WADL file -->
    </resource>

  </resources>
</application>
```

Listing A.4: WADL-Beschreibung der Test-Ressource „noMethod“.

```
<?xml version="1.0"?>
<application  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://wadl.dev.java.net/2009/02">
<!-- XML namespace for all WADL elements -->

  <doc xml:lang="EN" title="An example wadl description of a CoAP Resource
    that's method does not offer any in- or output Parameters"/>

  <resources base="coap://127.0.0.1/test/">

    <resource path="noParameters">
      <method name="DELETE">
          <!-- This method does not offer any Parameters in it's
            interface describing WADL file -->
          <request>
          </request>
          <response>
          </response>
      </method>
    </resource>

  </resources>
</application>
```

Listing A.5: WADL-Beschreibung der Test-Ressource „noParameter“.

Eidesstaatliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit selbständig verfasst wurde und nur die aufgeführten Quellen und Hilfsmittel genutzt wurden. Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, sind entsprechend kenntlich gemacht und die Arbeit ist in gleicher oder ähnlicher Form noch nicht Bestandteil einer Studien- oder Prüfungsleistung.

Rostock, der 21. Oktober 2013