FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Gebäudeautomation und RESTful BACnet/WS

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software & Information Engineering

eingereicht von

Ing. Christoph Stampfel
Matrikelnummer 1125580

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dr. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Daniel Schachinger

Wien, 21. April 2015

Christoph Stampfel	Wolfgang Kastner

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Building Automation Systems and RESTful BACnet/WS

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software & Information Engineering

by

Ing. Christoph Stampfel
Registration Number 1125580

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dr. Wolfgang Kastner
Assistance: Dipl.-Ing. Daniel Schachinger

Vienna, 21st April, 2015

Christoph Stampfel

Wolfgang Kastner

Erklärung zur Verfassung der Arbeit

Ing. Christoph Stampfel
Humboldtgasse 31/6, 1100 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 21. April 2015

Christoph Stampfel

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Tätigkeiten für diese Bachelorarbeit unterstützt und motiviert haben. Ein ganz besonderer Dank geht an Wolfgang Kastner, der meine Arbeit betreut hat und es mir ermöglicht hat meine Bachelorarbeit am Institut für Rechnergestützte Automation zu schreiben. Ein großes Dankeschön möchte ich auch an Daniel Schachinger richten, der mich während der Arbeiten immer unterstützt und mir wertvolle Tipps gegeben hat.

Ein großer Dank gebührt auch meiner Freundin Julia, die während meiner Studienzeit auf viel gemeinsame Zeit verzichten musste und mich trotzdem stets unterstützt hat.

Nicht zuletzt möchte ich auch meiner Familie und meinen Freunden danken, da sie mir im Laufe des Studiums immer wieder unter die Arme gegriffen haben und emotional für mich da gewesen sind.

Einen ganz besonderen Dank möchte ich an meinen kürzlich verstorbenen Papa richten. Sein Stolz, seine Anerkennung und seine Stütze haben mir den Weg zu diesem Abschluss geebnet. Leider ist er viel zu früh von uns gegangen.

Kurzfassung

Der Begriff *Gebäudeautomation* gewinnt in Kombination mit dem *Internet of Things* (IoT) immer mehr an Bedeutung in der heutigen Welt. Das Potential, das mit diesen Technologien ausgeschöpft werden kann, besteht im Wesentlichen aus dem Erreichen einer höheren Energieeffizienz von Gebäuden und dem erhöhten Komfort für die Menschen. Zahlreiche Technologien stehen zur Verfügung, die mehr oder weniger miteinander kompatibel sind. Dabei ist es für die Umsetzung des IoT wesentlich, dass *Gebäudeautomationssysteme* (GAS) auch Technologie- und Hersteller-übergreifend miteinander kommunizieren können. In der vorliegenden Arbeit wird ein möglicher Ansatz für die Integration von GAS über *Webservices* (WSs), basierend auf dem BACnet/WS Standard, vorgestellt. Die Grundlage für diesen Ansatz bildet dabei ein *generisches Gebäudeautomationsmodell* (GGM), welches die grundlegenden Aspekte eines GAS abbildet. Ausgehend von diesem Modell wird ein Mapping beschrieben, um Instanzen des GGM in Instanzen des BACnet/WS Objektmodells zu überführen. Das BACnet/WS Objektmodell bildet die Basis für die Referenzimplementierung, die im Rahmen dieser Arbeit erstellt wurde. Die verwendeten Technologien, Tools und Frameworks werden ebenso vorgestellt wie die Konzepte und die Architektur der Referenzimplementierung. Ausgewählte Codebeispiele werden präsentiert und diskutiert. Die Interaktion eines Web-Clients mit einem Beispielnetzwerk über den BACnet/WS Server wird anhand ausgewählter Anwendungsfälle im Rahmen einer Fallstudie demonstriert.

Abstract

Building automation in combination with the Internet of Things is becoming increasingly important in today's world. The main advantages of these technologies are the higher energy efficiency of smart buildings on the one hand and the increased comfort on the other hand. There are numerous technologies available which are more or less compatible with each other. For the implementation of the Internet of Things, it would be important that different building automations systems are working together in a technology and vendor independent way. In this thesis, a possible approach for the integration of building automations systems via Web services, based on the BACnet/WS specification is shown. This approach is based on a generic model which represents the common aspects of a modern building automation system. Based on this model, a mapping is described in order to convert instances of the generic model to instances of the BACnet/WS object model. Additionaly, a reference implementation of the BACnet/WS specification was created as part of this thesis, which will also be presented. Finally, the interaction of a Web client with an example building automation network is shown in the context of a case study.

Inhaltsverzeichnis

KAPITEL 1

Einführung

1.1 Motivation

Der Begriff Gebäudeautomation gewinnt in Kombination mit dem IoT immer mehr an Bedeutung in der heutigen Welt. Die Vernetzung verschiedenster Objekte, beginnend von Haushaltsgeräten bis hin zu ganzen Gebäuden, ermöglicht neue Anwendungsszenarien und legt somit den Grundstein für vielfältig einsetzbare IT-Lösungen. Dabei steht der Komfortgewinn und die Eröhung der Energieeffizienz im Vordergrund, wodurch letztlich auch die Umwelt profitiert. Vor allem die Vernetzung von GAS ermöglicht den intelligenten Einsatz von erneuerbaren Energien, indem Verbraucher gezielt dann aktiviert werden, wenn erneuerbare Energie zur Verfügung steht. Ein weiterer wichtiger Aspekt in Hinblick auf Gebäudeautomation ist die Sicherheit der Menschen. Durch die Vernetzung der GAS können Einsatzkräfte sehr schnell informiert werden, wenn Sensoren innerhalb des Gebäudes einen Alarm auslösen. Bei einem Feueralarm können die einzelnen Anlagen gezielt so gesteuert werden, dass die Ausbreitung des Rauchs minimiert wird. GAS werden heute auch immer mehr im privaten Umfeld eingesetzt und sind somit nicht mehr nur auf Zweckbauten beschränkt. Im privaten Bereich sind die Anforderungen an solche Systeme nicht die gleichen wie im gewerblichen Umfeld, wo geschultes Personal im Facility Management eingesetzt wird, stattdessen muss die Bedienung einfach, von überall zugänglich und trotzdem sicher erfolgen können. Durch die hochgradige Vernetzung vieler Geräte und Objekte wird die klassische Mensch-Maschine Kommunikation in Zukunft in den Hintergrund rücken, während die Maschine-Maschine Kommunikation in den Vordergrund tritt. Diese Allgegenwart von Computersystemen (Ubiquitous Computing) wird das Leben und das Kommunikationsverhalten der Menschen erheblich verändern. [1].

1.2 Problemstellung

Heutzutage sind viele verschiedene Komponenten und Netzwerkprotokolle in GAS im Einsatz, welche nur teilweise auf offenen Standards basieren. Aufgrund zahlreicher Implementierungen verschiedener Hersteller und der vielen verschiedenen Standards ist die Interoperabilität einzelner Geräte nur bedingt gewährleistet, was dem Ziel der Vernetzung verschiedener GAS und damit der Implementierung des IoT entgegenwirkt. Viele Standards haben sich im Laufe der Zeit entwickelt, da bisher das Hauptaugenmerk nicht auf die Kommunikation mit verschiedenen Systemen sondern eher auf Wirtschaftlichkeit und Leistungsfähigkeit gelegt wurde und die Systeme deshalb für spezielle Anwendungsgebiete optimiert wurden [2]. Zur Lösung dieses Problems stehen drei Ansätze zur Verfügung. Einer dieser Ansätze ermöglicht die IT-Integration über einen Gateway-Server, der sich um die Übersetzung der eingesetzten Netzwerkprotokolle und die Kommunikation mit den einzelnen Komponenten kümmert. Ein großer Vorteil dieses Ansatzes besteht darin, dass bestehende Systeme nicht verändert werden müssen. Es muss lediglich ein Gateway-Server zum GAS hinzugefügt werden, der mit dem bestehenden System kompatibel ist. Im BACnet/WS Standard werden Webservice-Schnittstellen definiert, welche das Protokoll eines Gateway-Servers standardisieren. Da Webservices im Internet mittlerweile weit verbreitet sind, eignet sich BACnet/WS für die Umsetzung eines solchen Gateways.

1.3 Zielsetzung

In dieser Arbeit wird ein möglicher Ansatz präsentiert, um ein GAS über einen Gateway mit dem Internet zu verbinden. Dabei wird im Rahmen der Arbeit ein GGM präsentiert, welches den Aufbau eines GAS generisch beschreibt und damit als Ausgangspunkt für weitere Anwendungsmöglichkeiten verwendet werden kann. Ein weiteres Ergebnis dieser Arbeit ist die Beschreibung des Mappings zwischen diesem GGM und dem BACnet/WS Objektmodell. Auf dieser Basis wird im letzten Abschnitt eine Referenzimplementierung des BACnet/WS Standards [3] vorgestellt, welche es ermöglicht, ein GAS über eine REST-Schnittstelle anzusprechen. Anschließend wird die Interaktion eines Web-Clients mit einem KNX-Beispielnetzwerk über den BACnet/WS Server anhand drei ausgewählter Anwendungsfälle demonstriert.

1.4 Methodische Vorgehensweise

Im Rahmen dieser Arbeit werden Fachartikel zu den Themen Gebäudeautomation und IT-Integration diskutiert, die im Zuge einer umfassenden Literaturrecherche ausgewählt wurden. Anhand der Literatur wird der aktuelle Stand der Technik und derzeitige Entwicklungen im Bereich der Gebäudeautomation vorgestellt. Dabei wird auf die Architektur von GAS eingegangen und mögliche Ansätze vorgestellt, welche die Integration mit externen Systemen ermöglichen. Da es das Ziel dieser Arbeit ist, eine Referenzimplementierung auf Basis von BACnet/WS vorzustellen, wird anschließend der BACnet/WS Standard [3] diskutiert, um die dahinterliegenden Konzepte und Datenstrukturen aufzuzeigen. Die

daraus gewonnenen Erkenntnisse, spiegeln sich im Metamodell wider, das im Rahmen dieser Arbeit vorgestellt wird und die wichtigsten Konzepte beinhaltet, die für diese Arbeit benötigt werden. Angelehnt an aktuelle Standards von GAS wird ein GGM präsentiert, welches als Ausgangsbasis für die weiteren Schritte dient. Ausgehend vom GGM wird untersucht, wie Instanzen dieses Modells in Instanzen des BACnet/WS Objektmodells überführt werden können. UML-Diagramme unterstützen die Visualisierung des Mappings zwischen den beiden Modellen. Basierend auf diesen Erkenntnissen wird eine Referenzimplementierung entwickelt, die Teile des BACnet/WS Standards implementiert. Im Zuge der Evaluierung wird der Zugriff auf ein KNX-Netzwerk über diesen BACnet/WS Server realisiert und in einer Fallstudie analysiert.

1.5 Struktur der Arbeit

In Kapitel 2 wird der aktuelle Stand der Technik im Bereich der Gebäudeautomation und die Möglichkeiten der Integration von GAS mit externen Systemen dargestellt. In Kapitel 3 wird das Objektmodell von BACnet/WS vorgestellt und die einzelnen Konzepte anhand eines Metamodells erklärt. Danach wird ein GGM präsentiert, welches ein GAS allgemein und Technologie-unabhängig beschreibt. In den darauffolgenden Abschnitten wird gezeigt, wie Instanzen des GGM in Instanzen des BACnet/WS Objektmodells überführt werden können. In Kapitel 4 wird eine Referenzimplementierung des BACnet/WS Standards vorgestellt. Im ersten Abschnitt wird auf die Architektur des BACnet/WS Servers eingegangen und die verwendeten Technologien, Tools und Konzepte genau erklärt. Anschließend werden ausgewählte Stellen des Quellcodes vorgestellt und disktutiert. Im letzten Abschnitt wird die Interaktion eines Web-Clients mit einem KNX-Netzwerk über den BACnet/WS Server anhand von drei ausgewählten Anwendungsfällen demonstriert. Abschließend wird in Kapitel 5 die vorliegende Arbeit nochmals reflektiert und dabei auf Probleme und offene Punkte eingegangen.

KAPITEL 2

Stand der Technik

In diesem Kapitel wird der aktuelle Stand der Technik zum Thema Gebäudeautomation und die Integration verschiedener GAS beschrieben. Im ersten Abschnitt wird der Begriff Gebäudeautomation erklärt und die geschichtliche Entwicklung von GAS erläutert. Anschließend wird der Aufbau eines typischen GAS anhand eines Modells vorgestellt und die grundlegende Bedeutung einzelner Bereiche beschrieben. Im zweiten Abschnitt wird auf die Integration zwischen GAS und anderen IT-Systemen, wie z.B. ERP-Systeme oder GAS, die auf anderen Standards basieren, eingegangen, mögliche Lösungsansätze präsentiert und diskutiert.

2.1 Gebäudeautomation

Unter dem Begriff Gebäudeautomation versteht man alle Einrichtungen der *technischen Gebäudeautomation* (TGA), die sich um die Überwachung, Steuerung, Regelung und Optimierung einzelner Anlagen im Gebäude kümmern. In den meisten Fällen regelt ein GAS die Funktionen Heizung, Lüftung, Klimatechnik, Licht, Feuerschutz und Sicherheit eines Gebäudes [4]. Der Anwendungsbereich ist aber nicht auf diese Funktionalitäten begrenzt, sondern kann auf viele weitere Gebiete wie z.B. Unterhaltungselektronik oder Verwaltung von Haushaltsgeräten ausgedehnt werden.

Die ersten GAS waren pneumatische Systeme, die Druckluft nutzten, um die komplexen Kontrollsequenzen auszuführen. Diese Systeme waren sehr Hardware-intensiv, wodurch hohe Anschaffungs- und Wartungskosten entstanden. Der mechanische Verschleiß und die begrenzte Genauigkeit wie auch die Inflexibilität waren Probleme, die diese Systeme mit sich brachten. Die pneumatischen Steuerungen waren im gesamten Gebäude verteilt und aufgrund der Komplexität der Systeme war es notwendig, Personal einzustellen, um die Systeme zu überwachen. Im Laufe der Zeit wurden die einzelnen Komponenten der Systeme immer zentraler verbaut, sodass die Steuerung und Überwachung über Steuerkonsolen durchgeführt werden konnte. Um die Anschaffungs- und

Wartungskosten zu reduzieren, wurden elektromechanische Multiplexverfahren eingesetzt, sodass es möglich wurde, die installierten Leitungen mehrfach zu benutzen. Nach der Einführung des Personal Computers in den 1980er Jahren wurden Mikroprozessoren auch in der Gebäudeautomation häufiger verwendet, wodurch es nun möglich war, die Steuerung der Systeme über flexiblere Softwarekomponenten zu realisieren. Heutzutage besteht ein GAS in der Regel aus vielen intelligenten Komponeten, die verteilt in einem Gebäude installiert sind und über ein gemeinsames Bussystem miteinander kommunizieren. Diese Systeme sind sehr flexibel, da die Konfiguration meist über eine spezielle Software kostengünstig angepasst werden kann, ohne die Komponenten im Feld oder deren Verkabelung verändern zu müssen. [4]

Ursprünglich wurden GAS installiert, um die Gebäudeverwaltung zu vereinfachen und damit die Kosten zu senken. Durch den Einsatz intelligenter Steuerungen ist es auch möglich, Energie einzusparen, wodurch sich die hohen Anschaffungskosten am Beginn durch die verminderten Betriebskosten amortisieren können. Heute werden GAS auch vermehrt im privaten Umfeld genutzt, wobei sich die Anforderungen, die an das GAS gestellt werden, vom Einsatz in einem Zweckgebäude unterscheiden. Beispielsweise entfallen regelmäßige Wartungsaufgaben von geschultem Personal im privaten Bereich, wodurch die Systeme so ausgerichtet sein müssen, dass auch Laien diese Tätigkeiten durchführen können. Die Benutzerfreundlichkeit und die Bedienbarkeit der Systeme hat somit im privaten Umfeld einen höheren Stellenwert als im kommerziellen Bereich. Die Hauptfaktoren für den Einsatz eines GAS im Eigenheim sind der erhöhte Komfort und die Energieeffizienz [4]. Ein Wohnhaus, das mit einem intelligenten GAS ausgestattet ist, wird auch als *Smart Home* bezeichnet. In einer Studie, die in einem kleinen italienischen Dorf durchgeführt wurde, konnte festgestellt werden, dass es durchaus möglich ist, eine große Menge an Energie in einem Smart Home einzusparen. In diesem Versuch wurden mehrere Häuser mit einem vernetzten GAS und einem Web-Interface für Remote Management ausgestattet. Dabei wurde in den ersten Jahren festgestellt, dass der Energieverbrauch um ca. 50% und der Wasserverbrauch um ca. 30% reduziert werden konnte. [5]

In einem GAS sind verschiedenste Komponenten im Einsatz, die über verschiedene Protokolle miteinander kommunizieren. Um den Aufbau eines GAS strukturiert darzustellen und damit die Funktionsweise besser verstehen zu können, wird zwischen der *Feldebene*, der *Automationsebene* und der *Managementebene* unterschieden, welche in Abbildung 2.1 exemplarisch dargestellt sind [2]. Die Steuerung der Prozesse im Gebäude erfolgt innerhalb der Feldebene durch Einsatz von Sensoren und Aktoren, welche die Steuerung und die Auswertung der Prozesse ermöglichen. Sensorwerte und die verfügbaren Kommandos der Aktoren werden in Form von Datenpunkten (Datapoints) logisch repräsentiert. In der Automationsebene befinden sich Steuereinheiten, die auf die Datenpunkte der Feldebene zugreifen und diese weiterverarbeiten können. Diese Steuereinheiten werden auch *Direct Digital Control* (DDC) genannt [6]. Ein typisches Beispiel für eine Steuerung in der Automationsebene wäre die Regelung der Wohnraumlüftung anhand der Daten eines Luftgütesensors. Im Gegensatz zur Feld- und Automationsebene werden in der Managementebene keine Prozessdaten ausgetauscht, sondern die gesamten Informationen des GAS gesammelt und weiterverarbeitet. In vielen Fällen werden die gesammelten

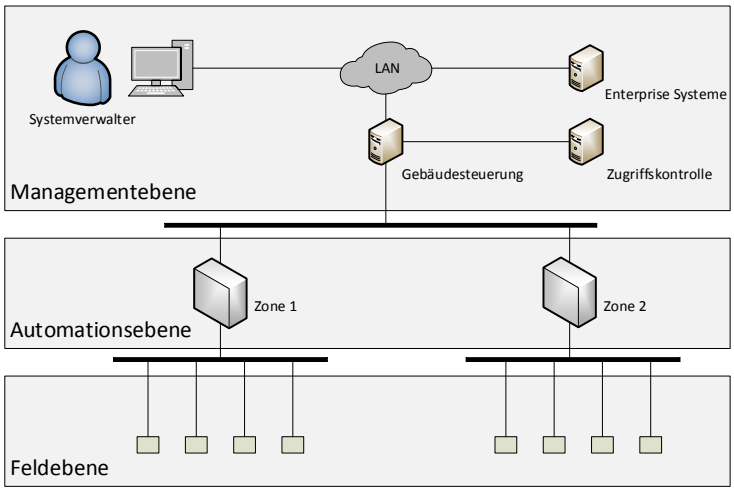

Abbildung 2.1: Modell für Gebäudeautomationssysteme [2]

Daten auch längerfristig gespeichert, um z.B. Trends im Energieverbrauch oder andere Merkmale des GAS über die Zeit auswerten zu können [6]. Diese Daten können dann eingesetzt werden, um den Energieverbrauch des Gebäudes gezielt zu optimieren. Aufgrund der größeren Menge an Daten sind in der Managementebene leistungsfähigere Geräte, mehr Speicher und breitbandigere Netzwerke notwendig. Die Administration des GAS erfolgt ebenfalls in der Managementebene durch den Systemverwalter, unter Verwendung von spezieller Software für die Konfiguration. In den unteren Ebenen kommunizieren die Geräte untereinander meist über ein eigenes Busprotokoll, welches für Geräte mit begrenzter Leistungsfähigkeit optimiert ist. Im Gegensatz dazu wird in der Managementebene bzw. heutzutage teilweise auch bereits in der Automationsebene das *Internet Protocol* (IP) für die Netzwerkkommunikation eingesetzt, wodurch eine einfache Anbindung ans Office-Netzwerk bzw. ans Internet ermöglicht wird. [6] Die drei Ebenen des Modells lassen sich mittlerweile nicht mehr eindeutig trennen, da die Geräte der unteren Ebenen immer leistungsfähiger werden und damit Funktionen der übergeordneten Ebenen übernehmen [7].

In Abbildung 2.2 sind ausgewählte Beispiele aktueller Technolgien für GAS abgebildet. Die grünen Flächen deuten an, in welcher Ebene diese Technologien im Modell angesiedelt sind. KNX [8] ist in Europa und vor allem in deutschsprachigen Ländern weit verbreitet und operiert hauptsächlich in der Feld- bzw. Automationsebene. Der Standard wird von der *KNX Association* betreut und wird typischerweise für Heizung, Belüftung, Klimatechnik, Beleuchtung und Verschattung eingesetzt. Jedoch ist die Funktionalität nicht nur auf diese Bereiche beschränkt, sondern kann beispielsweise auch für Zutrittskontrollsysteme bzw. intelligente Stromzähler eingesetzt werden. Ähnlich wie KNX bietet der BACnet Standard [9] Technologien für die Feld- und Automationsebene an. Der Standard wird von der *American Society of Heating, Refrigerating and Air-*

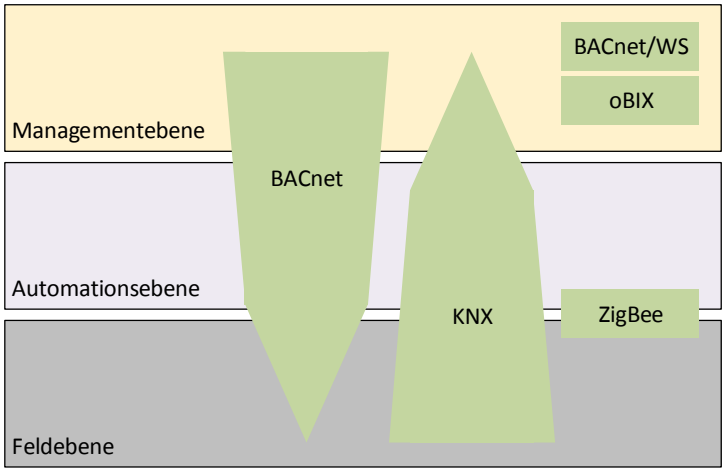

Abbildung 2.2: Beispiele für Technologien in Gebäudeautomationssystemen [7]

Conditioning Engineers (ASHRAE) entwickelt und betreut. BACnet bietet zusätzlich eine abstrakte und objektorientierte Repräsentation der übertragenen Informationen zwischen den einzelnen Peripheriegeräten an. ZigBee [10] ist im Modell zwischen der Feld- und der Automationsebene angesiedelt und ermöglicht die drahtlose Anbindung von Komponononenten in der Feldebene mit Komponenten in der Automationsebene. Im Gegensatz zu anderen Wireless-Technologien ist ZigBee auf leistungsärmere Hardware ausgelegt und benötigt weniger Energie. Die beiden Standards BACnet/WS [3] und oBIX [11] ermöglichen die Integration von GAS mit externen IT-Systemen oder anderen GAS. Sie werden im nächsten Abschnitt genauer behandelt. [7]

2.2 IT-Integration

Ein GAS ist heute in der Regel auf die Installation in einem Gebäude beschränkt und somit meist nur lokal verfügbar. Ein eventuell vorhandener Remote-Zugriff ist vorwiegend für die Überwachung oder Fernkonfiguration vorgesehen. Integriert man die Komponenten eines GAS ins Internet, so wird durch die Anbindung eine systemübergreifende Kommunikation ermöglicht. Das entspricht der Grundidee des IoT bezüglich einer nahtlosen Integration von physischen Objekten ins Internet über eine virtuelle Darstellung [2]. Durch die Einbindung von Gebäuden ins IoT werden viele verschiedene Anwendungszenarien ermöglicht, wodurch der Komfort gesteigert und Kosten durch Energie- und Personaleinsparungen gesenkt werden können. Geräte können z.B. anstehende Wartungsarbeiten signalisieren, wodurch ein ERP-System automatisiert Ersatzteile bestellen und Wartungstermine mit einem Techniker vereinbaren kann. Ein weiterer Anwendungsfall wäre die Umsetzung eines intelligenten Stromnetzes, wo durch die Verbindung mehrerer GAS eine höhere Energieeffizienz erreicht werden könnte [2]. Wie bereits im vorigen

Abschnitt erwähnt, kommunizieren die Geräte in heutigen GAS meist über eigene Busprotokolle, welche auf die geringe Leistungsfähigkeit der Komponenten im Feld ausgelegt sind. In der Managementebene wird aufgrund der unterschiedlichen Anforderungen meist ein anderes Netzwerkprokoll eingesetzt. Durch die Verwendung von unterschiedlichen Protokollen in den verschiedenen Ebenen ist eine direkte Adressierung der Komponenten von außen meist nicht möglich, wodurch in der Managementebene ein Intermediär in Form eines Gateways installiert werden muss, der die Anfragen von Außen entgegennimmt und an die entsprechenden Komponenten über das Busprotokoll weiterleitet. Um ein GAS ins Internet zu integrieren, stehen grundsätzlich drei Ansätze zur Verfügung, die in nachfolgender Auflistung zusammengefasst sind [7].

Zentraler Server Ein zentraler Server (Gateway) verbindet das GAS mit dem Internet über eine gemeinsame IP-Adresse. Der Server abstrahiert dabei das zugrundeliegende Protokoll des GAS und ermöglicht somit einen standardisierten Zugriff auf die einzelnen Komponenten [7].

IP-Router Die einzelnen Komponenten werden mit einem IP-Router verbunden, der als Proxy zwischen dem Internet und dem GAS fungiert. Dabei ordnet der Router jeder Komponente eine IP-Adresse zu und immitiert die Kommunikation über das IP-Protokoll [7].

Direkt Bei dieser Variante können die Komponenten direkt aus dem Internet adressiert werden, ohne einen Intermediär einzusetzen, der das Routing übernimmt [7].

Zur Integration eines GAS über einen zentralen Server stehen heute bereits einige Ansätze wie z.B. BACnet/WS [3] oder oBIX [11] zur Verfügung. Beide Ansätze definieren Webservice-Schnittstellen basierend auf aktuellen Standards wie *Representational State Transfer* (REST) und dem *Simple Object Access Protocol* (SOAP) und erleichtern dadurch die Integration ins Internet, da für diese Standards bereits bewährte Tools und Frameworks zur Verfügung stehen. Die Verwendung von Webservice-Schnittstellen bietet zusätzlich den Vorteil, dass die Anbindung von *Enterprise Systemen* z.B. über Enterprise Service Busse einfach ist, was die Integration von GAS in Geschäftsprozesse, die über ERP-Systeme abgewickelt werden, ermöglicht. Die Nachteile dieser Variante sind einerseits die höhere Last, die durch die Übersetzung der Netzwerkprotokolle am Gateway entsteht und andererseits, dass durch die fehlende Möglichkeit der direkten Adressierung die Idee des IoT nicht zur Gänze umgesetzt wird, da die Kommunikation nur über den Gateway möglich ist [7].

Der oBIX Standard definiert Schnittstellen sowohl für REST als auch für SOAP. Im BACnet/WS Standard (Addendum 135-2004c) aus dem Jahr 2004 wurde nur SOAP für die Webservice Kommunikation verwendet, jedoch hat sich das mit der jüngeren Version des Standards (Addendum 135-2012am) aus dem Jahr 2012 geändert, sodass derzeit nur mehr REST für die Kommunikation vorgesehen ist. Sowohl BACnet/WS als auch oBIX verwenden die *Extensible Markup Language* (XML) als primäres Datenformat zur Übertragung. Beide Standards sind jedoch nicht darauf beschränkt, sondern unterstützen auch

andere Formate wie z.B. das *Constrained Application Protocol* (CoAP), *Efficient XML Interchange* (EXI) oder die *JavaScript Object Notation* (JSON) bei oBIX. BACnet/WS ermöglicht zusätzlich die Übertragung der Daten in JSON, *JSON with padding* (JSONP) und in Plain-Text. Aus diesem Grund ist das Objektmodell von BACnet/WS im Standard [3] sehr allgemein beschreiben, um eine gemeinsame Basis für die verschiedenen Formate zu schaffen. Die Unterstützung von JSON und Plain-Text hat den Vorteil, dass der Overhead, der in der Regel durch die Verwendung von XML entsteht, reduziert werden kann und damit der Durchsatz erhöht wird. Beide Standards stellen ein flexibles Objektmodell zur Verfügung, um die Daten eines GAS repräsentieren zu können. Dabei können Vererbungshierarchien gebildet werden, die die Daten strukturieren, wobei oBIX im Gegensatz zu BACnet/WS die Möglichkeit von Mehrfachvererbung bietet. oBIX ist in Bezug auf Objekte sehr streng, da jede Information in Form eines Objekts abgebildet wird. In BACnet/WS können die Informationen in verschiedene Typen von s.g. *Data Items* abgebildet werden und müssen somit nicht in typisierte Objekte verpackt werden. Ein großer Vorteil von oBIX ist die Abbildung von Funktionen, welche im oBIX Objektmodell zusammen mit ihren Parametern definiert werden können. Im BACnet/WS Standard ist lediglich spezifiziert, wie Methoden über die Webservice-Schnittstelle aufgerufen werden können, nicht jedoch, wie man sie im Objektmodell mit ihren Parametern definieren kann [12]. Abgesehen von den Vor- und Nachteilen der einzelnen Standards erlaubt jeder der beiden eine flexible Repräsentation moderner GAS, wodurch sie sehr gut dazu geeignet sind, um die Probleme der Integration von GAS zu lösen.

Die direkte Einbindung der Komponenten eines GAS ins Internet wird über den dritten Ansatz ermöglicht, wo die Komponenten über eindeutige Adressen angesprochen werden können. Zurzeit wird die Kommunikation im Internet über das *Internet Protokoll in der Version 4* (IPv4) abgewickelt, welches aber nur ca. 4,3 Milliarden mögliche Adressen zur Verfügung stellt. Die letzten Adressbereiche wurden im Jahr 2011 von der *Internet Assigned Numbers Authority* (IANA) ausgegeben, sodass keine neuen Adressen mehr zur Verfügung stehen [13]. Aus diesem Grund wird derzeit in vielen Fällen eine *Network Address Translation* (NAT) durchgeführt, wodurch die Anzahl der benötigten IP-Adressen reduziert werden kann. Dabei wird ein Netzwerk über eine oder wenige öffentliche IP-Adressen ans Internet angebunden, anstatt für jedes Endgerät eine öffentliche IP-Adresse zu verwenden. Innerhalb des Netzes werden s.g. private IP-Adressbereiche eingesetzt, wodurch eine Adressübersetzung am Internet-Gateway notwendig wird, um die privaten Adressen in die öffentlichen Adressen zu übersetzen und umgekehrt [14]. Die Adressübersetzung erzeugt jedoch eine hohe Last am Gateway und verhindert die direkte Adressierung der einzelnen Komponenten im GAS. Eine mögliche Lösung für dieses Problem wäre der Einsatz von *IP version 6* (IPv6), da es einen ausreichend großen Adressraum zur Verfügung stellt, sodass jedem internetfähigen Gerät eine eigene IP-Adresse zugeordnet werden kann. Der weltweite Einsatz von IPv6 würde aber nur die Adressierbarkeit aller Komponenten sicherstellen, was aber noch keine nahtlose Integration ins IoT ermöglichen würde. Zurzeit sind noch viele verschiedene Komponenten im Einsatz, die über unterschiedliche Netzwerkprotokolle kommunizieren, sodass eine direkte, systemübergreifende Kommunikation noch nicht so einfach möglich ist. Es sind deshalb

noch weitere Standardisierungsschritte notwendig, um die Integration verschiedener Systeme zu ermöglichen, sodass ein globales System nach der Idee des IoT funktionieren kann [15].

KAPITEL 3

BACnet/WS Objektmodell

Das nachfolgende Kapitel erklärt einerseits das grundlegende Konzept, wie Daten in BACnet/WS strukturiert sind und andererseits wie diese Daten von einem GGM ins BACnet/WS Objektmodell übernommen werden können. Der grundsätzliche Aufbau der Daten wird in Form eines Metamodells erläutert, wobei auf die einzelnen Konzepte genauer eingegangen wird. Im letzten Teil dieses Kapitels wird das Mapping zwischen dem GGM und dem BACnet/WS Objektmodell anschaulich visuell und textuell beschrieben.

3.1 Objektmodell

Der BACnet/WS Standard [3] unterscheidet im Wesentlichen zwischen Daten und Metadaten. Die Werte einer betrachteten Größe werden in BACnet/WS als Daten bezeichnet, wobei die Bedeutung der Werte vom Kontext abhängig ist. Beispielsweise kann ein ganzzahliger Wert die Luftfeuchtigkeit in einem Raum oder die aktuelle Einstellung eines Dimmers repräsentieren. Metadaten charaktierisieren die Daten näher, indem sie zum Beispiel Einschränkungen, Semantik oder Beziehungen zwischen Daten beschreiben. Einschränkungen können Schranken wie z.B. Minimum und Maximum sein, die für einen bestimmten Wert vorgegeben werden. Die Bedeutung der Daten kann durch Metadaten wie z.B. *Tags* oder *Node-Types* beschrieben werden, indem passende Klassifizierungen für die Daten angegeben werden. Neben der Eltern-Kind Beziehung können Daten auch untereinander verlinkt und damit in Beziehung gesetzt werden. Alle Daten sind in BACnet/WS typisiert. Man unterscheidet zwischen einfachen und komplexen Datentypen. Zu den einfachen Datentypen zählen u.a. Zeichenketten, ganzzahlige Werte und boolesche Werte. Komplexe Datentypen entstehen durch die Zusammenfassung mehrerer einfacher und komplexer Datentypen zu komplexeren Strukturen wie z.B. Sequenzen, Objekten oder Collections. Die Daten und Metadaten werden in Form von Data Items abgebildet, welche die wichtigsten Komponenten im Objektmodell von BACnet/WS darstellen. Ein Data Item besitzt einen Namen und optionale Kindelemente, welche wiederum Data Items

sind. Bis auf das Wurzelelement besitzt jedes Data Item ein Elternelement. Der Wert eines Data Items wird im Attribut `value` gespeichert, welches den aktuellen Wert der betrachteten Größe widerspiegelt. Der Name, die Kindelemente und das Elternelement eines Data Items stellen Metadaten dar. Der BACnet/WS Standard sieht eine Reihe von vordefinierten Metadaten vor, die aber durch herstellerspezifische Metadaten erweitert werden können. Durch diesen allgemeinen Aufbau können mittels BACnet/WS baumartige Strukturen gebildet werden, welche sich besonders dazu eignen, um Netzwerke für die Gebäudeautomation abzubilden.

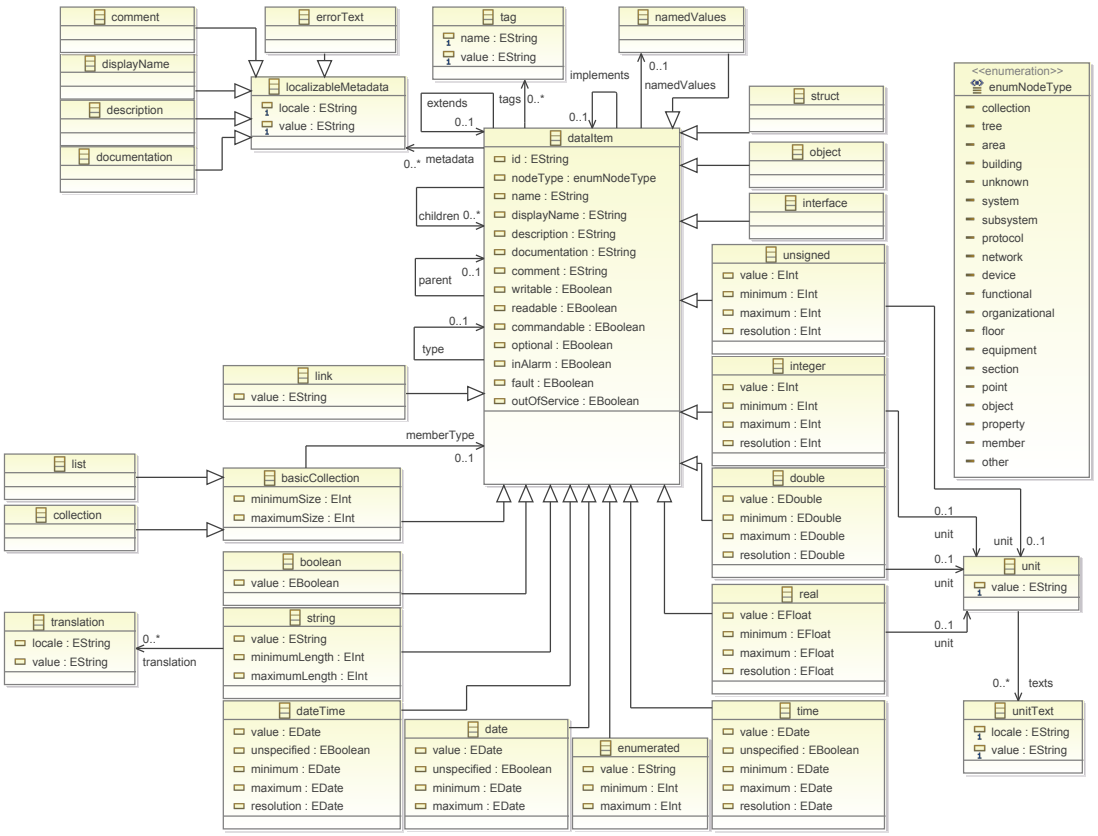

Abbildung 3.1: Auszug aus dem BACnet/WS Objektmodell

Das im Rahmen dieser Arbeit entwickelte Metamodell ist in Abbildung 3.1 dargestellt. Entsprechend dem Ziel der Arbeit stellt es lediglich einen Auszug aus dem Objektmodell von BACnet/WS dar und ist auf jene Elemente beschränkt, die für das Mapping zwischen dem GGM und dem BACnet/WS Objektmodell benötigt werden. Wie bereits erwähnt, ist das Data Item das zentrale Element in BACnet/WS. Alle Datentypen sind von `dataItem` abgeleitet, welche im Metamodell als Klassen dargestellt sind, die über Generalisierung mit `dataItem` verbunden sind. Die einzige Ausnahme bildet das Element `namedValues`, welches bei der Definition von Enumerationen eine große Rolle spielt. Dieses Element stellt keinen eigenen Datentyp dar, sondern es handelt sich hierbei um ein spezielles

Metadatum welches später noch genauer behandelt wird. Es wurde aus Gründen der Einfachkeit über die Generalisierung abgebildet. Im Metamodell stellen die Attribute von `dataItem` bzw. der davon abgeleiteten Datentypen die Metadaten dar. Eine Ausnahme bildet dabei das Attribut `value`, welches den Wert eines Data Items widerspiegelt und somit kein Metadatum ist.

Um den Unterschied zwischen einfachen und komplexen Datentypen besser erläutern zu können, werden exemplarisch die beiden Datentypen `Object` und `Boolean` aus Abbildung 3.1 herangezogen. `Boolean` wird zu den einfachen Datentypen gezählt, da der Wert im Feld `value` direkt beim Data Item selbst abgelegt ist. Beim Datentyp `Object` ist das nicht der Fall, da der Wert aus den Werten der Kindelemente gebildet wird, welche bei den Datentypen `Object` und `Interface` auch *Properties* genannt werden. Die komplexen Datentypen werden deshalb auch oft als *zusammengesetzte Datentypen* bezeichnet. Komplexe Datentypen können viele weitere einfache oder komplexe Daten als Kindelemente beinhalten, wodurch eine beliebig tiefe Hierarchie gebildet werden kann. Im Gegensatz dazu können einfache Datentypen keine anderen Daten in Form von Kindelementen beinhalten. Diese Einschränkung spiegelt sich jedoch nicht im Metamodell wider, sodass eine zusätzliche Modellvalidierung durchgeführt werden muss. In Listing 3.1 ist ein komplexer Datentyp in XML dargestellt, sodass man die hierarchische Struktur besser erkennen kann.

Numerische Datentypen sind eine Spezialform der einfachen Datentypen, da sie im Normalfall eine Angabe über die verwendete Einheit wie z.B. Meter, Zentimeter oder Kilogramm besitzen. Damit ein Client numerische Werte richtig interpretieren kann, ermöglicht BACnet/WS die Angabe der Einheit bei den numerischen Datentypen `Double`, `Integer`, `Real` und `Unsigned` in Form eines Metadatums. Im Metamodell ist die Zuordnung der Einheit über die Assoziation zur Klasse `unit` dargestellt. Die Einheit wird über das Attribut `value` in der Klasse `unit` festgelegt, wobei die möglichen Werte für `value` vom Standard vorgegeben werden. BACnet/WS ermöglicht zusätzlich die Festlegung eines Textes, der die Einheit in einer vom Menschen lesbaren Form darstellt. Dieser Text kann auch in verschiedene Sprachen übersetzt werden, sodass ein Client immer die gewünschte Darstellung geliefert bekommt. Im Metamodell werden diese Texte über die Klasse `unitText` abgebildet.

Listing 3.1: Hierarchische Netzwerkstruktur

```
1 <Collection name="networks">
2   <Object name="Office">
3     <Enumerated name="standard" />
4     <Collection name="views">
5       <Object name="functional">
6       <!-- ... -->
7     </Collection>
8   </Object>
9 </Collection>
```

Zur Beschreibung der Daten sieht der BACnet/WS Standard bereits viele vordefinierte

Metadaten vor. Die Metadaten werden in allgemeine und spezielle Metadaten unterteilt. Allgemeine Metadaten können in allen Data Items verwendet werden, wohingegen spezielle Metadaten nur bei bestimmten Elementen vorgesehen sind. Der Standard erlaubt es auch, die vordefinierten Metadaten um herstellerspezifische Metadaten zu erweitern. Beispiele für allgemeine Metadaten sind u.a. `name`, `type`, `extends`, `displayName` und `description`. Das Metadatum `name` nimmt in BACnet/WS eine sehr wichtige Rolle ein, weshalb jedes Data Item auch einen Namen besitzen muss. Dies ist vor allem notwendig, da das dargestellte Netzwerk über Pfade angesprochen wird, welche sich aus den Namen der einzelnen Data Items zusammensetzen. Das Data Item mit dem Namen `functional` aus Listing 3.1 lässt sich beispielsweise über den Pfad `/networks/Office/views/functional` ansprechen.

Listing 3.2: Typisierung und Vererbung

```
1  <!-- Definition -->
2  <Object name="AddressableEntity">
3    <Unsigned name="address" />
4  </Object>
5
6  <Object name="BooleanEntity" extends="AddressableEntity">
7    <Boolean name="value" />
8  </Object>
9
10 <Integer name="Percentage" minimum="0" maximum="100" />
11
12 <!-- Instanz -->
13 <Object name="Office">
14   <Object name="Button" type="BooleanEntity">
15     <Unsigned name="address" value="2048" />
16     <Boolean name="value" value="true" />
17   </Object>
18   <!-- ... -->
19   <Integer name="Dimming" type="Percentage" value="55" />
20 </Object>
```

Um eine Netzwerkrepräsentation für einen Client entsprechend aufbereiten zu können, bietet BACnet/WS, ähnlich wie objektorientierte Programmiersprachen, die Möglichkeit der Typisierung und der Vererbung an. Zur Unterscheidung zwischen Typdefinition und deren Verwendung werden Typdefinitionen im s.g. *Definition Context* abgelegt. Instanzen dieser Typen verweisen entsprechend auf die Definitionen im Definition Context. Diese Möglichkeit erleichtert es Client-Anwendungen, ein Netzwerk z.B. in einer grafischen Oberfläche für den Benutzer aufzubereiten, da die Typinformationen bereits im Vorfeld aus dem Definition Context ausgelesen werden können. Ein Data Item, welches eine Typdefinition darstellt, sollte einen global eindeutigen Namen besitzen, um eine eindeutige Referenzierung gewährleisten zu können. Ein typisiertes Data Item (Instanz) verweist auf den vordefinierten Typ, indem es den Namen des Typs im `type`-Metadatum ablegt. In Abbildung 3.1 wird die Typreferenz in Form einer Assoziation dargestellt. Der

Vererbungsmechanismus ist so realisiert, dass der Untertyp sämtliche Metadaten, Werte und Kindelemente des Obertyps erbt und diese durch Neudefinition überschreiben kann. Der abgeleitete Typ im Definition Context verweist auf den Obertyp durch Angabe des Namens im `extends`-Metadatum. In Abbildung 3.1 wird diese Referenz wieder in Form einer Assoziation dargestellt. Ein Beispiel für die Typisierung und Vererbung ist in Listing 3.2 abgebildet. Der Datentyp mit dem Namen `AddressableEntity` besitzt lediglich ein Kindelement, welches eine Adresse als `Unsigned`-Wert aufnimmt. Der abgeleitete Datentyp `BooleanEntity` erbt das Kindelement `address` und definiert zusätzlich das Kindelement `value` vom Typ Boolean. Die Instanz vom Typ `BooleanEntity` enthält somit beide Kindelemente und definiert zusätzlich jeweils einen bestimmten Wert im `value`-Attribut. Der Typ `Percentage` schränkt den vordefinierten Typ `Integer` ein, indem die Metadaten `minimum` und `maximum` definiert werden. Instanzen von diesem Typ dürfen nur Werte annehmen, die sich innerhalb des angegebenen Intervalls befinden. Wie man anhand des vorigen Beispiels erkennen kann, erleichtert die Typisierung dem Client nicht nur die Repräsentation der Daten, sondern ermöglicht dem Server zusätzlich eine Validierung, sodass im Beispiel keine ungültigen Werte bei Instanzen vom Typ `Percentage` akzeptiert werden.

Diese Validierung ist vor allem bei Enumerationen interessant, wo in der Regel eine bestimmte Menge von gültigen Werten vordefiniert ist. Eine Instanz einer nicht erweiterbaren Enumeration darf nur Werte annehmen, die aus dieser vordefinierten Menge stammen. Für die Definition von Enumerationen ist das Metadatum `namedValues` von großer Bedeutung, da die Menge der möglichen Werte über dieses Element abgebildet wird. Ein Beispiel für die Definition einer Enumeration und deren Instanzen ist in Listing 3.3 dargestellt. Die möglichen Werte für die Enumeration werden als Kindelemente des Elements `NamedValues` angegeben, wobei es sich bei der Definition von Enumerationen um Elemente vom Typ `Unsigned` handeln muss. Im Metamodell ist diese Einschränkung jedoch aus Gründen der Einfachheit nicht abgebildet, sodass diese durch eine zusätzliche Modellvalidierung sichergestellt werden muss. Bei der Definiton eines Werts wird der Name und eine numerische Repräsentation angegeben. Zusätzlich ist es über das `displayName`-Metadatum möglich von Menschen lesbare Texte anzugeben.

Listing 3.3: Definition von Enumerationen

```
1  <!-- Definition -->
2  <Enumerated name="enumPriority">
3    <NamedValues>
4      <Unsigned displayName="Low" name="Low" value="1" />
5      <Unsigned displayName="High" name="High" value="2" />
6      <Unsigned displayName="Alert" name="Alert" value="3" />
7    </NamedValues>
8  </Enumerated>
9
10 <!-- Instanz -->
11 <Enumerated name="priority" value="High" type="enumPriority" />
12 <Enumerated name="priority" value="1" type="enumPriority" />
```

BACnet/WS ermöglicht eine logische Klassifizierung der Data Items durch Angabe eines s.g. Node-Type. Die Klassifizierung ermöglicht einem Client eine bessere Visualisierung der Daten, da Data Items mit unterschiedlicher Klassifikation z.B. durch verschiedene Symbole dargestellt werden können. Der Node-Type wird dabei über das `nodeType`-Metadatum angegeben und ist auf Werte beschränkt, die in der Enumeration `enumNodeType` in Abbildung 3.1 angegeben sind. Kindelemente vom Typ `Object` oder `Interface` erhalten standardmäßig den NodeType `Property` [3]. Möchte man die Klassifikation um herstellerspezifische Einträge erweitern, ist das nicht über Node-Types sondern über Tags möglich. Tags sind Name/Wert-Paare, die einem Data Item zugeordnet werden. Sie können vom Hersteller beliebig definiert werden, sollten jedoch laut Standard eindeutige Namen besitzen, um eine Zuordnung zu Herstellern gewährleisten zu können. In Abbildung 3.1 sind Tags über die Klasse `tag` definiert und ein Data Item kann eine beliebig lange Liste an Tags enthalten.

Eine weitere wichtige Eigenschaft von BACnet/WS ist die Unterstützung von Internationalisierung, sodass es möglich ist, manche der vordefinierten Metadaten und Werte von Data Items vom Typ `String` in andere Sprachen zu übersetzen. Im Metamodell in Abbildung 3.1 spiegelt sich diese Möglichkeit in den Elementen `localizableMetadata` und `translation` wider. Man beachte, dass die Menge der internationalisierbaren Metadaten in der Abbildung nicht vollständig ist und somit nur einen Auszug darstellt. Die möglichen Werte für das `locale`-Attribut werden vom Standard vorgegeben. Der Server besitzt jedoch eine voreingestellte Sprache, die verwendet wird, wenn der Client keine konkrete Sprache anfordert.

3.2 Generisches Gebäudeautomationsmodell

Das GGM in Abbildung 3.2 ist ein Datenmodell, welches die Struktur eines GAS abstrakt beschreibt. Jedes Element im GGM ist von `Element` abgeleitet, womit diesem Element eine ähnlich zentrale Rolle wie dem Data Item im BACnet/WS Objektmodell zukommt. Die Struktur eines GAS wird im Modell in Form eines Netzwerks dargestellt. Ein Netzwerk besteht im Wesentlichen aus zwei Teilen, nämlich den *Inventory-Listen* und den *Views*. Eine Inventory-Liste stellt eine Sammlung von so genannten *Items* dar, welche die einzelnen Komponenten des GAS repräsentieren. Es wird zwischen verschiedenen Typen unterschieden, die im GGM alle vom Element `Item` abgeleitet sind. Eine Ausnahme bildet der Untertyp `Reference`, da hier keine Komponenten abgebildet werden, sondern es sich um eine Art *Repository* handelt, wo allgemeinere Konzepte wie z.B. Datenpunkttypen, Medientypen oder Hersteller gespeichert werden, die an anderer Stelle in den konkreten GAS-Standards spezifiziert sind. Das GGM unterscheidet zwischen den beiden Komponententypen `Datapoint` und `Device`. Datapoints stellen die Kommunikationsendpunkte dar, wodurch die Kommunikation der Komponenten im Netzwerk erst ermöglicht wird. Dabei dienen unterschiedliche Datenpunkttypen dazu, um verschiedene Anwendungen zu realisieren. Beispielsweise wird ein Schaltaktor, der nur die Zustände Ein/Aus kennt, anders angesprochen als ein Dimmer, der einen größeren Wertebereich entgegennehmen kann. Die verschiedenen Datenpunkttypen beinhalten somit ein Regelwerk, welches die

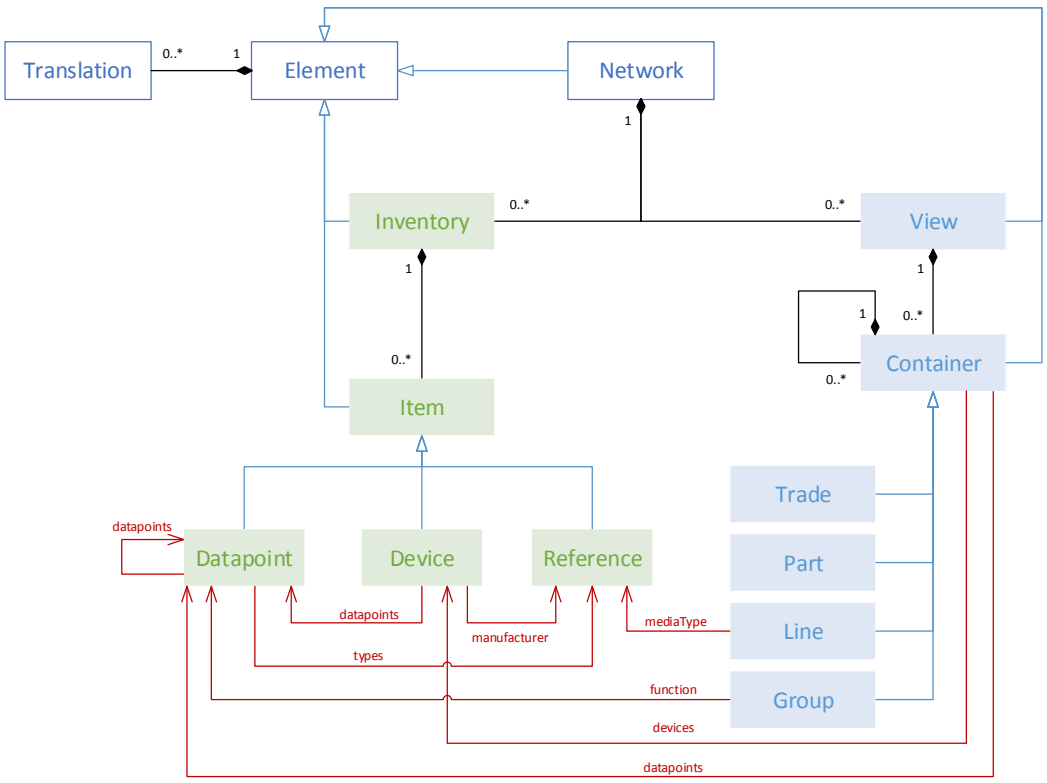

Abbildung 3.2: Generisches Gebäudeautomationsmodell

Kommunikation und die Darstellung der Daten für den jeweiligen Datenpunkt steuert. Ein physisches Gerät im Netzwerk wird mit der Klasse `Device` modelliert und enthält ein oder mehrere Datenpunkte, dargestellt durch die Referenzen auf `Datapoint`.

Views stellen das zugrundeliegende Netzwerk hierarchisch dar und referenzieren die einzelnen Items aus den Inventory-Listen. Die hierarchische Darstellung wird über das Element `Container` gebildet, wobei jeder Container weitere untergeordnete Container enthalten kann. Das GGM unterscheidet zwischen vier verschiedenen Typen von Containern, wobei man sich bei der Modellierung des Netzwerks nicht auf diese vier Typen beschränken muss. Der Typ `Part` bildet die Gebäudeteile ab. Somit können Räume zu Stockwerken und Stockwerke zu einzelnen Gebäuden zusammengefasst werden. Die physische Topologie des Netzwerks wird über den Typ `Line` abgebildet. Ein Container vom Typ `Group` bildet Gruppen von Komponenten, die gemeinsam über das Netzwerk angesprochen werden können. Eine Gliederung nach den Funktionstypen, wie z.B. Licht, Heizung oder Klimatechnik kann über den Container-Typ `Trade` realisiert werden. Hinter der Trennung der Inventory-Listen und den Views steckt die Idee, dass die Items der Inventory-Listen in den Views referenziert werden können und somit nicht doppelt

definiert werden müsssen. Zusätzlich hat diese Trennung den Vorteil, dass die Vergabe von Berechtigungen auf View-Ebene erteilt werden kann.

3.3 Mapping

In diesem Abschnitt wird das Mapping zwischen dem in Abbildung 3.2 dargestellten GGM und dem BACnet/WS Objektmodell aus Abbildung 3.1 erläutert. Zunächst erfolgt eine allgemeine Beschreibung über die Vorgehensweise beim Mapping. Das Mapping der Inventory-Listen, Views, Datapoints und Devices wird in den anschließenden Unterabschnitten erläutert.

3.3.1 Allgemeines

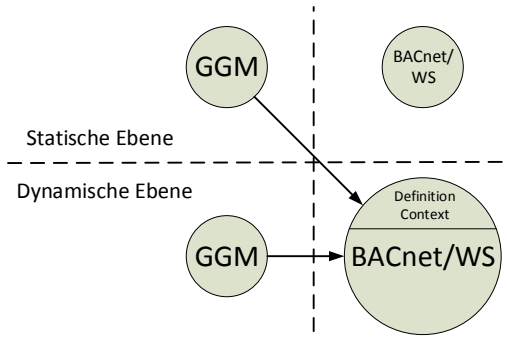

Abbildung 3.3: Mapping

In Abschnitt 3.1 wurde bereits erwähnt, dass BACnet/WS ein Typsystem und ein Vererbungskonzept besitzt. Diese Funktionalität wird für das Mapping herangezogen, indem die Elemente aus dem GGM in Data Items überführt und diese als Typen im Definition Context definiert werden. Es wird also zunächst die statische Struktur des GGM in das dynamische Typsystem von BACnet/WS überführt. Die statische Struktur wird dabei in den Definition Context übernommen und die konkrete Ausprägung des Netzwerks erfolgt auf Instanzebene. Diese Vorgehensweise ist in Abbildung 3.3 dargestellt. Die Namen der Typen im Definition Context werden im `name`-Metadatum abgelegt und die Instanzen verweisen über das `type`-Metadatum auf die Typdefinition. Die Elemente werden in Data Items vom Typ `Object` übersetzt, da dieser Typ den Anforderungen am ehesten entspricht. Im BACnet/WS Standard wird erwähnt, dass Objekte dann verwendet werden sollen, wenn eine eindeutige Identifikation möglich ist und die Objektdefinitionen öffentlich verfügbar sind. Genau das ist der Fall, da die Struktur des GGM in Form von identifizierbaren Objekten abgebildet und öffentlich zur Verfügung gestellt werden kann.

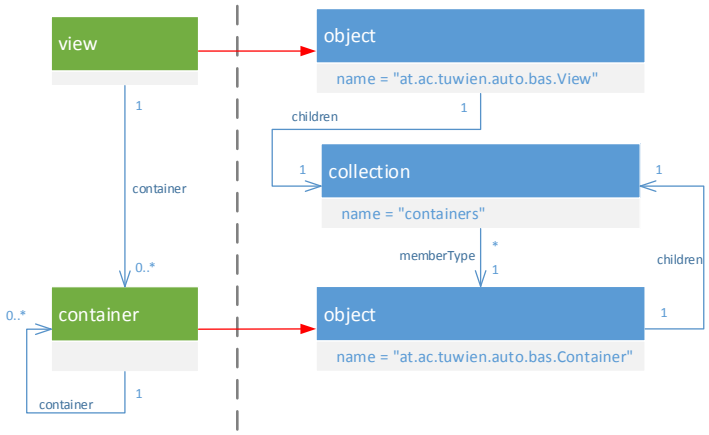

Abbildung 3.4: Mapping von Views und Containern auf Typebene

3.3.2 Views

Da Views für die Darstellung eines Netzwerks eine besondere Bedeutung haben, wird nachfolgend genauer auf das Mapping der Views eingegangen. In Abbildung 3.4 wird dargestellt, wie Views und Container in BACnet/WS auf Typebene abgebildet werden. Die Elemente `View` und `Container` werden in Data Items mit den entsprechenden Typnamen übersetzt. Typnamen beginnen mit dem Präfix `at.ac.tuwien.auto.bas`, um eine globale Eindeutigkeit zu gewährleisten. Die Namen für die Typen View und Container sind deshalb `at.ac.tuwien.auto.bas.View` bzw. `at.ac.tuwien.auto.bas.Container`. Die Abbildung von 1:n Beziehungen erfolgt über Collections. Listen, die im GGM auf Komponenten innerhalb von Inventory-Listen verweisen, werden zu Listen im BACnet/WS Objektmodell. Der wesentliche Unterschied zwischen Collections und Listen in BACnet/WS ist, dass bei Collections die Kindelemente über ihren Namen angesprochen werden und bei Listen die Namen einer fortlaufenden Nummerierung beginnend bei 1 entsprechen. Genau genommen sind die Elemente einer Liste unsortiert und besitzen keinen Namen. Sie erhalten aber für die Darstellung eine Nummerierung, um die Konvention, dass jedes Data Item einen Namen besitzen muss, einzuhalten. Dieser Unterschied wird in Listing 3.4 examplarisch dargestellt. Die Beziehung zwischen `View` und `Container` wird deshalb als Collection abgebildet, die sich als Kindelement im Objekt vom Typ `at.ac.tuwien.auto.bas.View` befindet. Container besitzen optional mehrere untergeordnete Container, deshalb ist hier ebenfalls eine Container-Collection enthalten. Der Typ der Kindelemente einer Collection kann über das Metadatum `memberType` festgelegt werden, sodass die Collection nur Kinder von diesem Typ oder einem Untertyp akzeptiert. Deshalb wird das Metadatum `memberType` in der Collection, die die Container enthält, auf den Typnamen des Containers gesetzt.

Im GGM besitzt das Element `Element` die Attribute `id`, `name` und `description`. `Element` ist abstrakt und muss somit nicht in den Definition Context von BACnet/WS

Listing 3.4: Unterschied zwischen Collections und Listen in BACnet/WS

```
1 <Collection name="containers">
2   <Object name="Dimming On / Off outer right">
3     <List name="types">
4       <String name="1" value="DPST-1-1" />
5     </List>
6   </Object>
7 </Collection>
```

übernommen werden, da diese Attribute direkt in der entsprechenden Instanz als Metadatum im Data Item gespeichert werden können. Das Mapping auf Instanzebene ist in Abbildung 3.5 dargestellt. Die View- und Container-Instanzen enthalten jeweils einen Verweis auf die vordefinierten Typen `at.ac.tuwien.auto.bas.View` und `at.ac.tuwien.auto.bas.Container` über das `type`-Metadatum, dargestellt als Assoziation zwischen den Objekten.

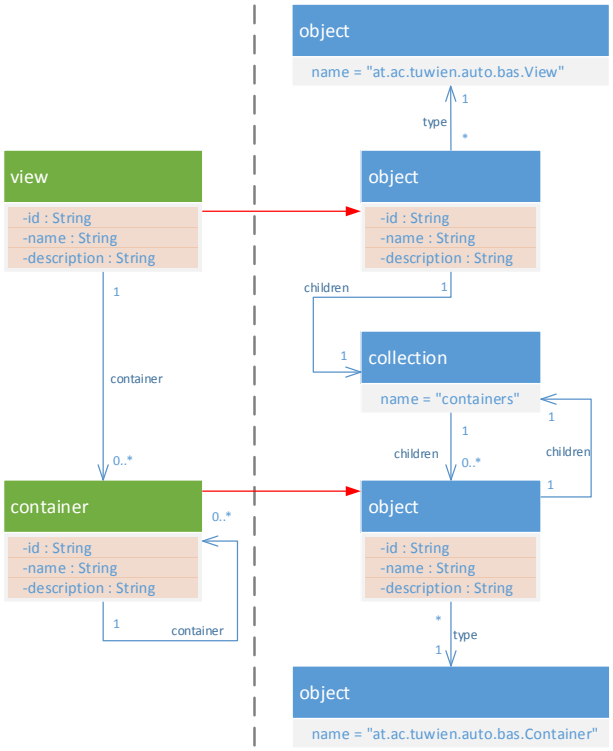

Abbildung 3.5: Mapping von Views und Containern auf Instanzebene

Wie in Abbildung 3.2 ersichtlich ist, besitzt ein Container Referenzen auf Datapoints und Devices aus den Inventory-Listen, dargestellt durch die Assoziationen `datapoints`

und devices. Diese Referenzen werden in BACnet/WS als Listen abgebildet, die als Kindelemente im Container abgelegt sind. Leider bietet der BACnet/WS Standard keine Möglichkeit, um typisierte Referenzen darzustellen. Die Typisierung der Referenz ist aber sinnvoll, um den Typ des referenzierten Data Items einschränken zu können. Das Problem wird so gelöst, indem der Datentyp *Link* aus dem Standard herangezogen wird, um davon eigene Referenztypen abzuleiten. Die Definition der Referenztypen für die Verweise auf Datapoints und Devices ist in Listing 3.5 dargestellt. Dabei bildet der Typ at.ac.tuwien.auto.bas.Reference den Basistyp für alle benötigten Referenzen in unserem Mapping. Um den Typ des referenzierten Objektes festlegen zu können, führen wir das neue Metadatum referenceType ein, welches auf den entsprechenden Typ verweist. Leitet man nun vom Referenztyp at.ac.tuwien.auto.bas.Reference ab, dann kann man den Wert von referenceType im Definition Context festlegen und die Referenzen somit typisieren.

Listing 3.5: Definition von Referenztypen

```
1  <!-- Definition -->
2  <Link name="at.ac.tuwien.auto.bas.Reference" />
3
4  <Link name="at.ac.tuwien.auto.bas.DatapointReference"
5      referenceType="at.ac.tuwien.auto.bas.Datapoint"
6      extends="at.ac.tuwien.auto.bas.Reference" />
7
8  <Link name="at.ac.tuwien.auto.bas.DeviceReference"
9      referenceType="at.ac.tuwien.auto.bas.Device"
10     extends="at.ac.tuwien.auto.bas.Reference" />
```

Die Verweise auf Datapoints bzw. Devices werden nun über die Referenztypen at.ac.tuwien.auto.bas.DatapointReference bzw. at.ac.tuwien.auto.bas.DeviceReference abgebildet, wobei der konkrete Referenztyp im memberType-Metadatum der Liste festgelegt wird. Das Verweisziel wird über das value-Attribut von Link abgebildet, indem man einen *Uniform Resource Identifier* (URI) angibt, der das Verweisziel darstellt. Der URI wird dabei folgendermaßen gebildet: inventory/$inventorylist/$item. Der Platzhalter $inventorylist steht dabei für den Namen der Inventory-Liste und $item steht für den Namen des entsprechenden Eintrags innerhalb der Inventory-Liste. Das vollständige Mapping auf Typebene ist in Abbildung 3.6 dargestellt.

Im GGM in Abbildung 3.2 wird zwischen verschiedenen Typen von Containern unterschieden. Da die verschiedenen Container-Typen im GGM über die Generalisierung abgebildet werden, muss diese Typhierarchie mit dem Vererbungsmechanismus von BACnet/WS abgebildet werden. Das Mapping ist in Abbildung 3.7 dargestellt.

Nachfolgend wird der Container-Typ *Group* genauer behandelt. Die anderen Container-Typen werden analog ins BACnet/WS Objektmodell übersetzt. Der Untertyp Group wird in BACnet/WS als Objekt vom Typ at.ac.tuwien.auto.bas.Group übersetzt, welches vom Typ Container über die extends-Beziehung abgeleitet ist. Somit

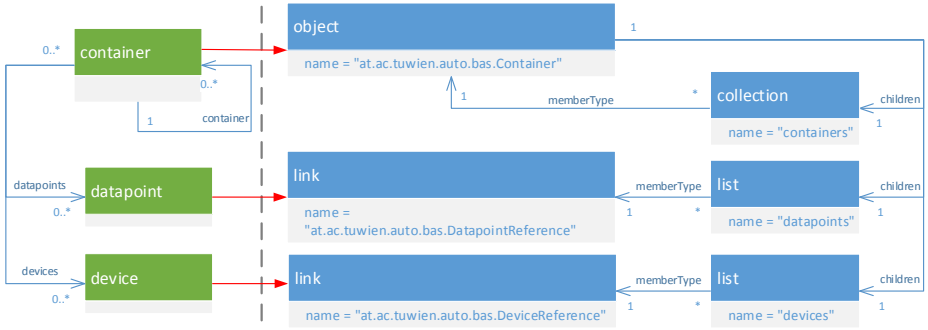

Abbildung 3.6: Mapping der Referenzen zum Inventory des Container Elements auf Typebene

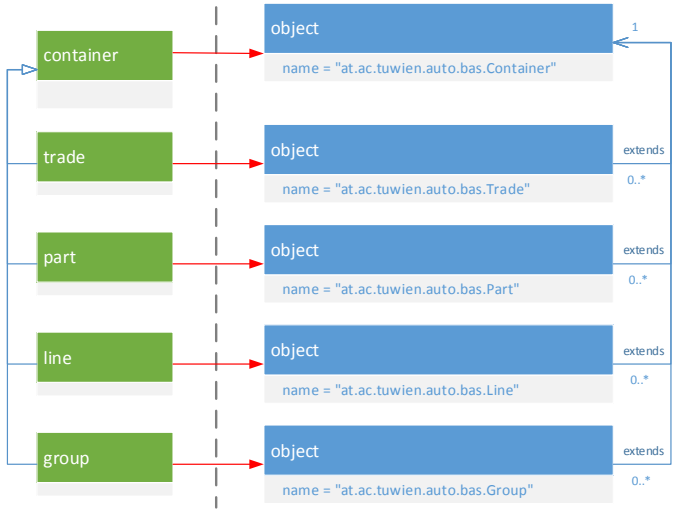

Abbildung 3.7: Mapping der verschiedenen Container-Typen auf Typebene

erbt `Group` alle Kindelemente des Containers. Wie in Abbildung 3.8 zu sehen ist, wird die Gruppenadresse in `Group` als Kindelement vom Typ `Unsigned` abgebildet. Die Referenz auf die Komponente vom Typ `datapoint` wird in Form einer typisierten Referenz `at.ac.tuwien.auto.bas.DatapointReference` als Kindelement in `Group` aufgenommen. Der Name `function` der Beziehung zwischen `Group` und `Datapoint` wird ins `name`-Metadatum der Referenz aufgenommen.

In Abschnitt 3.2 wurde bereits erwähnt, dass im GGM eine Art Repository mit Elementen vom Typ `Reference` gebildet wird. Referenzen auf dieses Repository werden bei der Umsetzung in BACnet/WS nicht übernommen, stattdessen werden die Werte direkt als Zeichenketten übernommen. Im Container-Typ *Line* wird beispielsweise eine

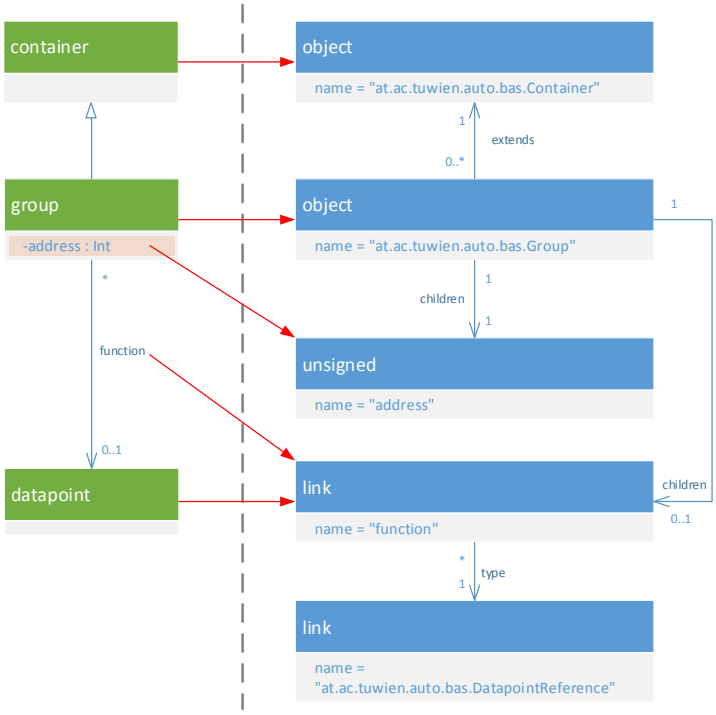

Abbildung 3.8: Mapping des Container-Typs Group in BACnet/WS auf Typebene

solche Referenz verwendet. Das Mapping ist in Abbildung 3.9 dargestellt.

3.3.3 Datapoints

Datenpunkte stellen die Kommunikationsendpunkte des Netzwerks dar und übernehmen deshalb eine wichtige Funktion. Die Umsetzung der Datenpunkte ins BACnet/WS Objektmodell ist in Abbildung 3.10 dargestellt. Um Beziehungen zwischen Datenpunkten darzustellen, besitzt das Element `Datapoint` optional Referenzen auf andere Datenpunkte. Diese Referenzen werden in BACnet/WS in Form einer Liste abgebildet, welche typisierte Referenzen vom Typ `at.ac.tuwien.auto.bas.DatapointReference` auf andere Datapoints enthält. Ein Datenpunkt verweist auf einen oder mehrere Datenpunkttypen, welche als Elemente vom Typ `Reference` in den Inventory-Listen abgelegt sind. Diese Referenzen werden nicht abgebildet, sondern der Wert der Referenz wird in eine `String`-Instanz übernommen. Diese Instanzen sind als Kindelement in einer Liste mit dem Namen `types` im Datapoint abgelegt. `Datapoint` enthält neben den bisher erwähnten Daten auch weitere kommunikationsspezifische Informationen, die den Kommunikationsablauf steuern. Es handelt sich dabei um die Attribute `writable`, `readable`, `priority`, `communication`, `transmittable` und `updatable`. Das Attribut `priority` ist vom Typ *enumPriority*, welcher einer Enumeration mit den möglichen Werten `Low`, `High` und

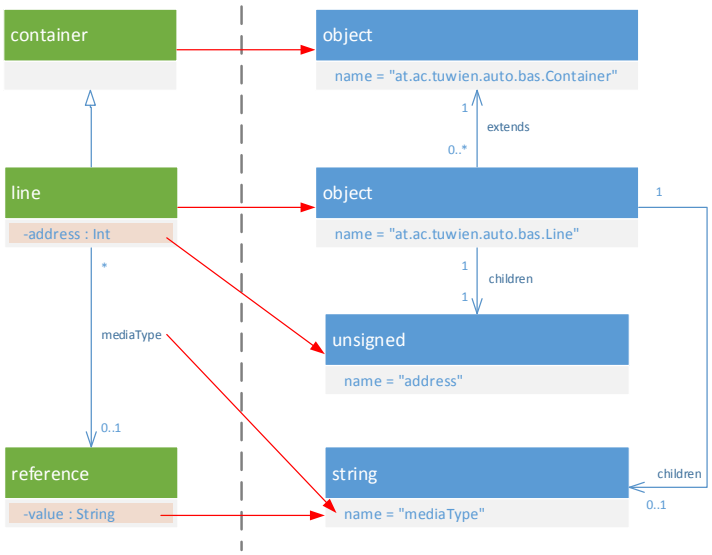

Abbildung 3.9: Mapping des View-Typs Line auf Typebene

`Alert` entspricht. Diese Enumeration wird auch in BACnet/WS in eine Enumeration unter Verwendung des `Enumerated`-Typs übersetzt. Die anderen Attribute sind vom Typ `boolean` und werden ebenfalls in Instanzen vom Typ `Boolean` in BACnet/WS übernommen. Wichtig ist noch anzumerken, dass die Attribute `writable` und `readable` nicht als Kindelement von `Datapoint` abgebildet, sondern direkt übernommen werden, da Metadaten dafür vorhanden sind.

3.3.4 Device

Neben dem Datapoint ist das `Device` ein weiterer Komponententyp im GGM und modelliert ein physisches Gerät im Netzwerk. Es fasst mehrere Datenpunkte zusammen, über die das Gerät mit anderen Geräten kommuniziert. Das Mapping ist in Abbildung 3.11 dargestellt. Das Device besitzt die Attribute `address`, `orderNumber`, `serialNumber` und `applicationNumber`, welche in die entsprechenden Objekttypen in BACnet/WS übersetzt und als Kindelemente in `Device` abgelegt werden. Da es sich bei Werten des Attributs `address` stets um natürliche Zahlen handelt, wird das Attribut in den Typ `Unsigned` umgewandelt. Die anderen Attribute sind vom Typ String und werden deshalb als Instanzen von `String` abgelegt. Der Herstellername des Device wird im GGM im Repository abgelegt und von einer Instanz vom Typ `Device` referenziert. Da bei der Umsetzung ins BACnet/WS Objektmodell das Repository nicht abgebildet wird, wird der Herstellername direkt in eine Instanz vom Typ `String` überführt und als Kindelement im Device gespeichert. Das Device besitzt Referenzen auf Objekte vom Typ `Datapoint`, welche in BACnet/WS als Liste übersetzt werden, welche Referenzen vom Typ `at.ac.tuwien.auto.bas.DatapointReference` enthält.

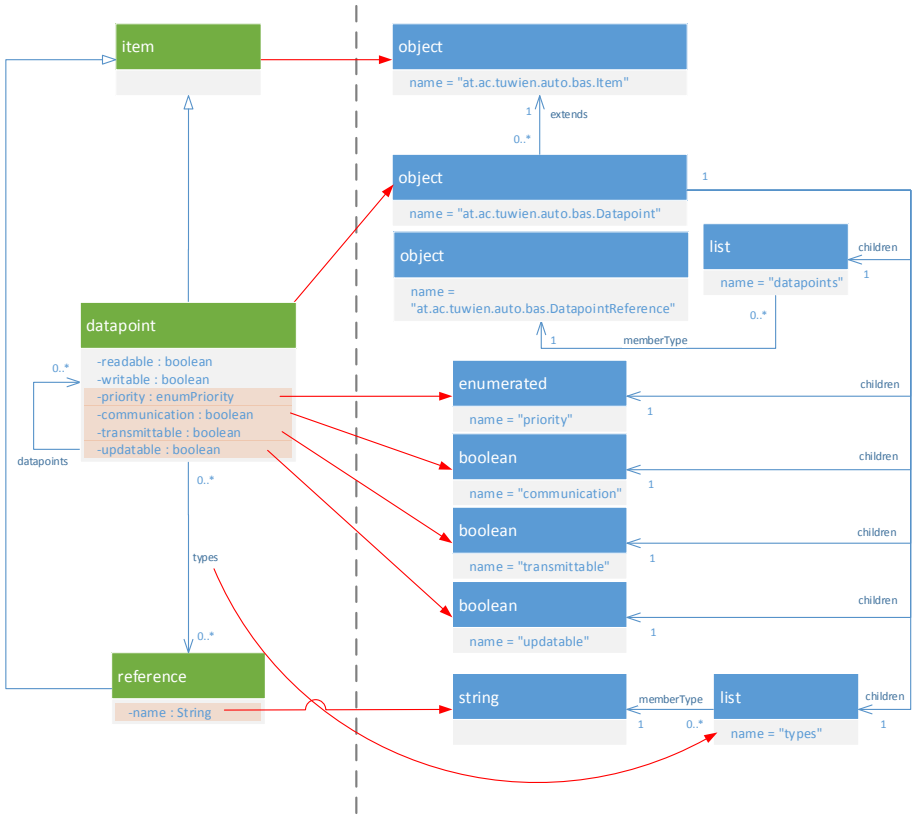

Abbildung 3.10: Mapping der Datapoints in BACnet/WS auf Typebene

Das Mapping der Elemente `Network` und `Inventory` erfolgt analog zu den bisherigen Überlegungen, indem entsprechende Typen im Definition Context definiert werden. Auf das Mapping des Elements `Translation` wird nicht weiter eingegangen, da dies über den Rahmen dieser Arbeit hinausgehen würde.

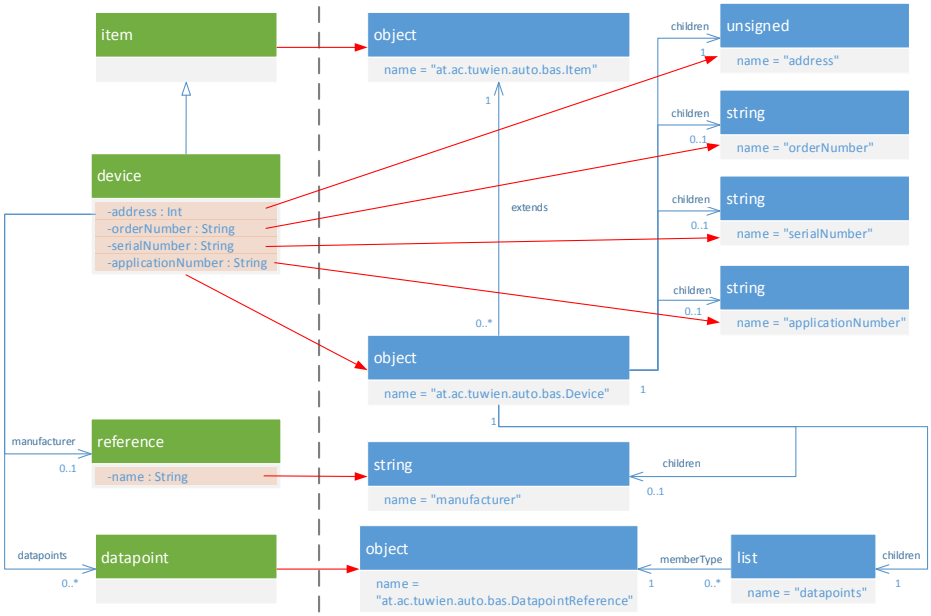

Abbildung 3.11: Mapping des Device in BACnet/WS auf Typebene

KAPITEL 4

Referenzimplementierung

Im Rahmen dieser Arbeit wurde eine Referenzimplementierung für BACnet/WS erstellt, die es ermöglicht ein Netzwerk der Gebäudeautomation über eine BACnet/WS Webservice-Schnittstelle anzusprechen. Dieses Kapitel beschreibt die Konzepte dieser Referenzimplementierung. Am Beginn wird die Architektur und die Technologiewahl der Implementierung ausführlich diskutiert. Ausgewählte Passagen der Implementierung werden im darauffolgenden Abschnitt präsentiert, um danach das Kapitel mit einer Fallstudie, in der die Interaktion mit einem KNX-Beispielnetzwerk vorgeführt wird, abzuschließen.

4.1 Architektur

4.1.1 Technologien

Im Zuge der Referenzimplementierung wurde eine Serverumgebung erstellt, die es ermöglicht ein GAS über eine BACnet/WS Webservice-Schnittstelle anzusprechen. Der BACnet/WS Standard sieht vor, dass die Kommunikation mit dem Server über eine REST-Schnittstelle durch Austausch von XML-, JSON-, JSONP- oder Plain-Text-Daten erfolgen soll. REST steht für Representational State Transfer und ist ein Paradigma, das mittlerweile auch als Basis für Webservices weit verbreitet ist und von vielen Tools unterstützt wird. REST stellt dabei eine Abstraktion der Funktionsweise des World Wide Web dar, sodass alle herkömmlichen Webseiten bereits dem REST-Paradigma entsprechen [16]. In REST werden alle Datenelemente, die über einen Namen angesprochen werden können, *Ressourcen* genannt und die Adressierung erfolgt über s.g. *Resource Identifiers*. Im Internetumfeld wird für die Adressierung ein URI verwendet, welcher die Ressourcen identifiziert. Ressourcen können durch den Einsatz verschiedener Formate unterschiedliche Repräsentation haben, wie z.B. XML, HTML oder PDF. Ein weiteres architektonisches Prinzip von REST ist, dass jede Interaktion mit einem REST-Service statuslos erfolgt, d.h., alle vom Service benötigten Daten müssen in der Anfrage enthalten sein. Dies hat den Vorteil, dass der Service vom Client nie durch vorangehende Anfragen

in einen bestimmten Status versetzt werden muss. Dem Client stehen für die Interaktion mit einem REST-Service einige der bekannten HTTP-Methoden wie z.B. GET, PUT, POST oder DELETE zur Verfügung, um die Ressourcen abzurufen bzw. zu manipulieren.

Die Programmiersprache Java unterstützt das REST-Paradigma durch die Spezifikation der Programmierschnittstelle *Java API for RESTful Services* (JAX-RS) [17]. Aufgrund der Plattformunabhängigkeit von Java und der umfangreichen Unterstützung von REST wurde Java als Programmiersprache für die Referenzimplementierung gewählt. Da die Programmierschnittstelle JAX-RS nur Interfaces definiert, nicht jedoch eine Implementierung bereitstellt, muss eine geeignete Implementierung gewählt werden. In der Beispielimplementierung wird Jersey [18] verwendet, da es sich dabei um die Referenzimplementierung des JAX-RS Standard handelt. In der Anwendungsschicht wird in der Beispielimplementierung HTTP verwendet, wodurch es für unsere Implementierung notwendig ist, auf HTTP-Anfragen reagieren zu können. Die Servlet-API von Java stellt eine Abstraktionsschicht dar, die es ermöglicht, Java-Klassen als *Servlets* zu implementieren, welche bei HTTP-Anfragen aufgerufen werden. Damit diese Servlets aufgerufen werden können, ist ein Servlet-Container notwendig, der die Anfragen empfängt, aufbereitet und an die jeweiligen Java-Klassen weiterleitet. Im Beispiel wurde *Apache Tomcat 8.0* [19] als Servlet-Container verwendet, es sollte aber auch möglich sein, die Anwendung auf anderen Servlet-Containern laufen zu lassen, da keine Tomcat-spezifischen Funktionalitäten verwendet werden. Um die Abhängigkeiten der einzelnen Klassen besser konfigurieren zu können, wird der *Dependency-Injection* Mechanismus des Spring Frameworks [20] genutzt. Im Gegensatz zum klassischen Ansatz ermöglicht Dependency-Injection eine losere Kopplung der Komponenten einer Anwendung, indem die Abhängigkeiten zwischen den Software-Komponenten von einem Container verwaltet werden, anstatt die Abhängigkeiten der Klassen selbst zu verwalten, indem diese selbst instanziert werden. Außerdem unterstützt das Design Pattern Dependency-Injection eine bessere Strukturierung des Quellcodes, wodurch die Lesbarkeit erhöht und die Möglichkeit automatisierter Tests verbessert wird.

Zusätzlich zu den bisher genannten Technologien werden noch weitere Frameworks zur Unterstützung eingesetzt. Der Technologie-Stack des BACnet/WS Servers ist in Abbildung 4.1 dargestellt. Um Fehlermeldungen und wichtige Nachrichten loggen zu können, wird das bekannte Logging-Framework *log4j* [21] der Apache Software Foundation eingesetzt. Als zusätzliche Unterstützung werden Tools der Apache Commons Bibliothek verwendet, da diese sehr viele allgemeine Funktionalitäten wie z.B. Utilities für IO, Collections und Logging zur Verfügung stellt, sodass diese nicht neu implementiert werden müssen.

Eines der wichtigsten Frameworks ist die Java-Bibliothek *Calimero* [22]. Sie wurde am Institut für Rechnergestützte Automation der Technischen Universität Wien entwickelt und ermöglicht die Kommunikation mit einem KNX-Netzwerk über Java. Calimero erlaubt die Netzwerkanbindung auf einer abstrahierten Ebene, sodass die genaue Implementierung des zugrundeliegenden Netzwerkprotokolls dem Entwickler nicht unmittelbar bekannt sein muss. Es eignet sich deshalb sehr gut, um mit relativ wenig Aufwand ein KNX-Netzwerk aus einer Java-Anwendung anzusprechen. Im Zuge der Fallstudie wird Calimero für das

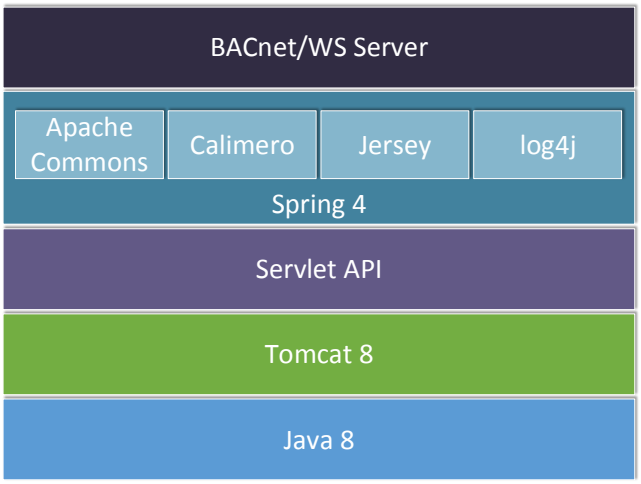

Abbildung 4.1: Technologie-Stack des BACnet/WS Servers

KNX-Beispielnetzwerk eingesetzt.

4.1.2 Entwicklungsumgebung

Entwickelt wurde der BACnet/WS Server mithilfe der Entwicklungsumgebung *Eclipse Luna* [23]. Als Build-Management Tool wurde *Apache Maven 3* [24] eingesetzt, sodass die Projekte auch in anderen Entwicklungsumgebungen, welche Apache Maven unterstützen, kompiliert werden können. Maven bietet u.a. den Vorteil, dass Projekteinstellungen und die Konfiguration von Abhängigkeiten unabhängig von der Entwicklungsumgebung vorgenommen werden können. Das jeweilige Maven-Plugin der Entwicklungsumgebung kümmert sich dann um die spezifischen Projekteinstellungen der zugrundeliegenden Plattform. Ein weiterer Vorteil von Maven ist, dass die Bibliotheken, von denen das Projekt abhängig ist, automatisch aus dem Internet heruntergeladen und eingebunden werden. Auch transitive Abhängigkeiten werden dabei berücksichtigt.

Der Quellcode des BACnet/WS Servers wurde aus Gründen der Übersichtlichkeit in mehrere Module aufgeteilt, deren inhaltliche Bedeutung in nachfolgender Auflistung erklärt wird:

bacnetws-rest-server Dieses Projekt stellt aus Sicht von Apache Maven das *Parent-Projekt* dar, welches allgemeine Projekteinstellungen und die Versionsnummern für die gesamten Abhängigkeiten verwaltet.

bacnetws-rest-server-config Sämtliche Konfigurationsdateien für die Anwendung und die Definitionen für das Beispielnetzwerk sind in diesem Projekt enthalten. Auch die Konfigurationsdateien für das Spring Framework sind hier abgelegt.

bacnetws-rest-server-data In diesem Projekt befinden sich alle Klassen, welche mit dem Objektmodell von BACnet/WS in Verbindung stehen.

bacnetws-rest-server-network Dieses Projekt enthält Schnittstellendefinitionen und Implementierungen für die Kommunikation mit dem KNX-Netzwerk.

bacnetws-rest-server-util Allgemeine Funktionalitäten, die von mehreren Projekten verwendet werden, sind in diesem Projekt enthalten.

bacnetws-rest-server-web In diesem Projekt ist der Quellcode der eigentlichen Webanwendung abgelegt. Daraus wird das s.g. *Web Application Archive* (WAR) erstellt, welches dann in einem Servlet-Container installiert werden kann. Hier sind Implementierungen für die REST-Schnittstelle, Marshalling, Unmarshalling, Fehlerbehandlung und das Einlesen der Netzwerkdefinition enthalten.

4.1.3 Datenstruktur

Um den definierten Funktionsumfang aus dem BACnet/WS Standard umsetzen zu können bzw. die Kommunikation mit dem KNX-Netzwerk überhaupt zu ermöglichen, muss der BACnet/WS Server die Struktur des zugrundeliegenden Netzwerks kennen. Die Netzwerkdefinition wird in der Implementierung aus den beiden XML-Dateien `definitions.xml` und `networks.xml` eingelesen. Diese Dateien müssen sich in einem gemeinsamen Ordner befinden, dessen Pfad dem Server bekannt ist. Die Inhalte der einzelnen Dateien sind in nachfolgender Tabelle zusammengefasst:

Datei	Inhalt
definitions.xml	In dieser Datei wird der Definition Context von BACnet/WS festgelegt. Die Datei enthält somit das Mapping auf Typebene, d.h., Objekte, Enumerationen und andere Datentypen werden hier definiert.
networks.xml	Die Netzwerkstruktur wird in dieser Datei festgelegt, indem die Views definiert werden. Zusätzlich enthält diese Datei die Definition der Inventory-Listen aus dem GGM. Das Mapping auf Instanzebene ist somit in dieser Datei zu finden.

Da es sich bei den o.g. Dateien um XML-Dateien handelt, müssen die Definitionen der jeweiligen Data Items ebenfalls im XML-Format, welches im BACnet/WS Standard festgelegt ist, angegeben werden. Ein Beispiel für den Inhalt der Datei `definitions.xml` ist in Listing 4.1 dargestellt. Das Wurzelelement `Collection` mit dem Namen `.definitions` fungiert als Container für die Typdefinitionen. Bei der Abfrage des Definition Context vom BACnet/WS Server wird die `Collection` genau so zurückgeliefert, wie sie in dieser Datei definiert wurde.

In Kapitel 3 wurde bereits erwähnt, dass Datapoints und Devices in den Inventory-Listen abgebildet und in den verschiedenen Views referenziert werden. Der Endbenutzer kann die Inventory-Listen aber nicht über die Webservice-Schnittstelle ansprechen, sondern die entsprechenden Elemente werden direkt in den Views ausgegeben, wo sie referenziert werden. Diese Umsetzung hat den Vorteil, dass es für den Endbenutzer transparent ist, wie er auf die einzelnen Elemente zugreifen kann und muss somit nur die

Listing 4.1: Auszug der Datei `definitions.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<Collection name=".definitions">
  <Object name="at.ac.tuwien.auto.bas.Network">
    <Collection name="views"
       memberType="at.ac.tuwien.auto.bas.View" optional="true" />
    <Enumerated name="standard" type="at.ac.tuwien.auto.bas.enumStandard" />
  </Object>
  <!-- ... -->
</Collection>
```

Views des Netzwerks betrachten. Diese Transparenz hat aber den Nachteil, dass sich die Serverimplementierung darum kümmern muss, die Elemente im Inventory synchron zu halten, wenn Schreibzugriffe aus verschiedenen Views erfolgen. Um die Synchronisierung zu ermöglichen, werden die bereits bekannten, typisierten Referenzen verwendet. Ein Beispiel der internen Darstellung ist in Listing 4.2 dargestellt.

Listing 4.2: Referenzen auf Elemente im Inventory

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- ... -->
<List name="datapoints">
  <Link name="1"
     value="inventory/invlist1/P-0341-0_DI-3_M-0001_A-9803-03-3F77_O-3_R-4"
     type="at.ac.tuwien.auto.bas.DatapointReference" />

  <Link name="2"
     value="inventory/invlist1/P-0341-0_DI-10_M-0001_A-2413-01-B14F_O-4_R-27"
     type="at.ac.tuwien.auto.bas.DatapointReference" />
</List>
<!-- ... -->
```

In der Datei `networks.xml` werden die Netzwerke mit ihren Inventory-Listen und Views definiert. Die Struktur der Datei ist in Listing 4.3 dargestellt. Das Wurzelelement bildet ein Element vom Typ `Collection` mit dem Namen `networks`, welches eine Liste der Netzwerkdefinitionen in Form von Objekten vom Typ `at.ac.tuwien.auto.bas.Network` enthält. Jede Netzwerkdefinition besitzt in der Regel eine oder mehrere Inventory-Listen, welche sich innerhalb der `Collection` mit dem Namen `inventories` befinden. Eine Inventory-Liste ist ebenfalls eine Collection, welche Objekte vom Typ `at.ac.tuwien.auto.bas.Item` aufnehmen kann. Elemente vom Typ `Item` besitzen die Attribute `id`, `name` und `description`, wobei das `name`-Attribut zwingend notwendig ist, da die Referenzierung in den Views über dieses Attribut erfolgt. Aus diesem Grund ist es auch erforderlich, dass die Namen innerhalb eines Netzwerks eindeutig sind.

Beim Starten des Servers werden beide Dateien eingelesen und dabei die interne Repräsentation des Netzwerks in Form einer Baumstruktur aufgebaut, die der Server

Listing 4.3: Auszug der Datei `networks.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<Collection name="networks"
            memberType="at.ac.tuwien.auto.bas.Network">
  <Object name="Office" displayName="Office"
          type="at.ac.tuwien.auto.bas.Network">
    <!-- Views -->
    <!-- ..... -->
    <Collection name="inventories">
      <Collection id="inv1" memberType="at.ac.tuwien.auto.bas.Item" ...>
        <Object id="P-0341-0_DI-3_M-0001_A-9803-03-3F77_O-3_R-4"
          name="Switch, Channel A" displayName="Switch, Channel A"
          description="On / Off" readable="false" writable="true"
          type="at.ac.tuwien.auto.bas.DPST-1-1">
          <List name="types">
            <String name="1" value="DPST-1-1" />
          </List>
          <Enumerated name="priority" value="Low" />

          <Boolean name="value" readable="false" writable="true" />
          <Enumerated name="encoding" type="at.ac.tuwien.auto.bas.Encoding" />
        </Object>
      </Collection>
  </Object>
</Collection>
```

benötigt, um auf Anfragen reagieren zu können. Diese Baumstruktur inklusive Beispielen für die Inventory-Listen und den Definiton Context ist in Abbildung 4.2 dargestellt. Der Wurzelknoten wird dabei aus einem Data Item vom Typ *Struct* gebildet, da dies im BACnet/WS Standard so vorgesehen ist. Der Standard definiert einige Kindelemente des Wurzelknotens, die eine Referenzimplementierung zur Verfügung stellen muss. In der Beispielimplementierung wurden jedoch nur die beiden Kindelemente `.definitions` und `.trees` umgesetzt. Die Collection mit dem Namen `.definitions` aus der Datei `definitions.xml` wird deshalb direkt als Kindelement in den Wurzelknoten eingehängt. Analog dazu wird die Collection mit dem Namen `networks` der Datei `networks.xml` als Kindelement an das Element `.trees` angehängt. Das Element `.trees` wird deshalb für die Liste der Netzwerke verwendet, da es laut BACnet/WS Standard als Container für die logische Strukturierung der Daten dient. Die beiden Elemente sind in der Abbildung in der zweiten Baumebene ersichtlich. Die Verweise auf die einzelnen Elemente im Inventory werden in Form von Referenz-Objekten im Baum hinterlegt, sodass im Falle einer Anfrage das entsprechende Element aus dem Inventory ermittelt werden kann. In Abbildung 4.2 werden diese Referenzen als rote Pfeile dargestellt. Wie bereits erwähnt, sind die Inventory-Listen für den Benutzer unsichtbar, da er über die Webservice-Schnittstelle nicht darauf zugreifen kann. Deshalb ist es auch nicht sinnvoll bei einer Anfrage die Referenz als Link auf die Inventory-Liste auszugeben, da der Benutzer ohnehin nicht darauf zugreifen könnte. Aus diesem Grund wird anstatt der Referenz der Inhalt des

referenzierten Objektes aus der Inventory-Liste ausgegeben, so als würde das Objekt in der View als Kopie vorliegen. Data Items aus den Views können typisiert sein und entsprechend auf einen vordefinierten Typ im Definition Context verweisen. Dies wird in Abbildung 4.2 durch die schwarzen Pfeile visualisiert.

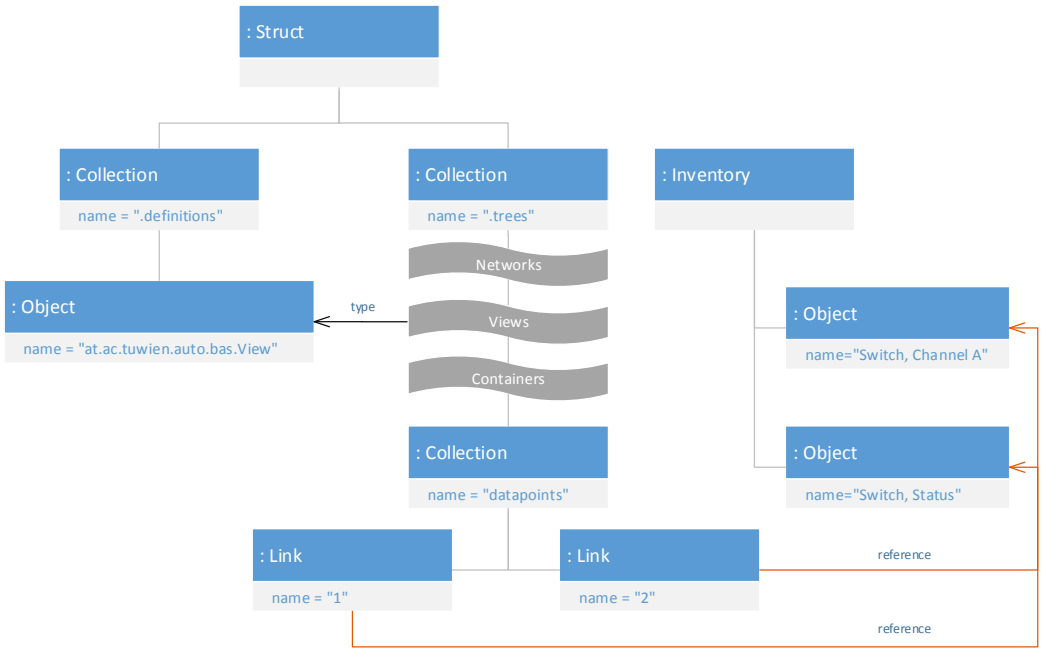

Abbildung 4.2: Interne Baumstruktur der BACnet/WS-Objekte

4.1.4 Serialisierung

Der BACnet/WS Standard unterscheidet zwischen vier verschiedenen Formaten, in denen die interne Datenstruktur dargestellt werden kann. Das Standard-Format stellt dabei eine Serialisierung der Daten in XML dar, weshalb in der vorliegenden Referenzimplementierung vorerst auf die anderen Formate verzichtet wurde. Neben XML besteht die Möglichkeit, die interne Datenstruktur in JSON, JSONP oder in reinem Text (Plain-Text) darzustellen, wobei die Ausgabe in reinem Text nur Werten von einfachen Datentypen vorbehalten ist. Ein Client kann durch Verwendung des Parameters `alt` das Ausgabeformat für eine Service-Anfrage festlegen. Da REST nicht statusbehaftet ist, muss der Parameter in jedem Request mitgeschickt werden, bei dem ein alternatives Format verwendet werden soll. Aufgrund der Möglichkeit in unterschiedlichen Formaten mit dem Server zu interagieren, ist ein Konzept notwendig, wie diese verschiedenen Formate so abstrahiert werden, dass die Anwendungslogik unabhängig vom Format implementiert werden kann. In der Referenzimplementierung wurde die Problematik so gelöst, dass die Daten vor der Abarbeitung in die interne Darstellung umgewandelt und nach der

Abarbeitung wieder in das gewünschte Format transformiert werden. Im Detail bedeutet das, dass die Nutzdaten in einem Request von einem *Unmarshaller* in das interne Format umgewandelt werden. Die Service-Schicht des BACnet/WS Servers erhält die Nutzdaten bereits in der internen Darstellung und kann diese für die gewünschten Operationen verwenden. Handelt es sich z.B. um eine Anfrage zur Manipulation eines Data Items in der internen Baumdarstellung, dann werden die entsprechenden Teile im Baum durch die Daten aus der Anfrage ersetzt. Da die Daten der Anfrage in diesem Fall bereits im internen Format vorliegen, sind diese Operationen einfach möglich. Werden nach Abarbeitung einer Anfrage wieder Daten an den Client zurückgeschickt, dann müssen diese wieder in das gewünschte Format umgewandelt werden. Dieser Vorgang wird *Marshalling* genannt. Zum besseren Verständnis ist das Marshalling/Unmarshalling eines Requests in Abbildung 4.3 vereinfacht als Aktivitätsdiagramm dargestellt.

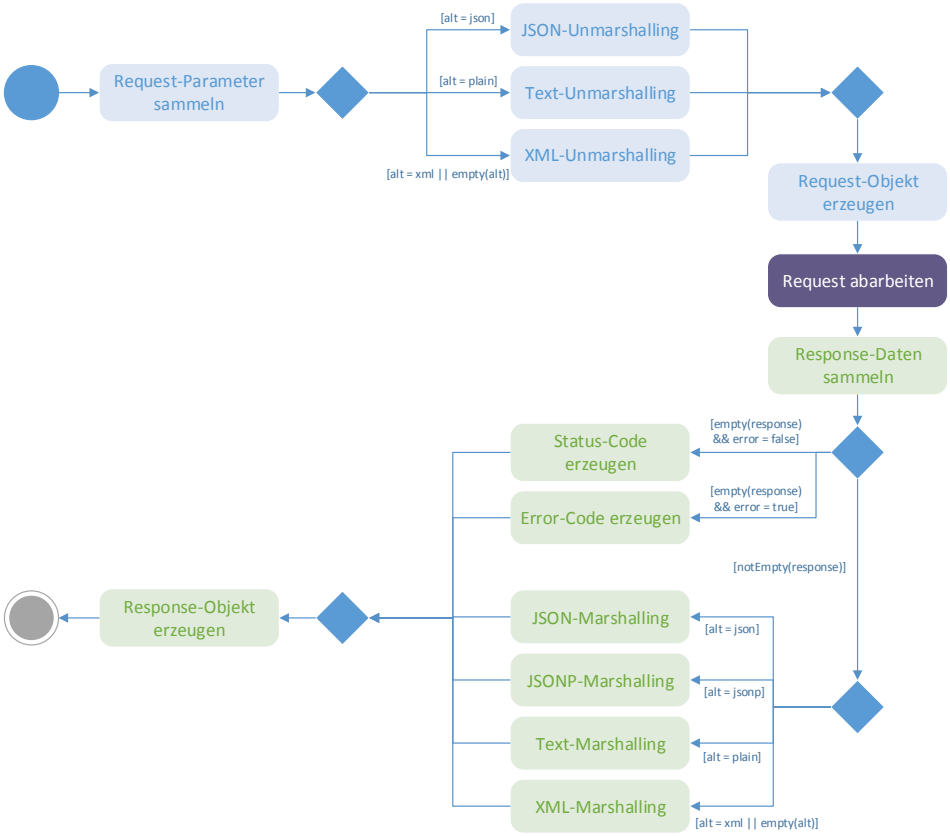

Abbildung 4.3: Ablauf des Marshalling/Unmarshalling

Alle Aktivitäten, die in hellblauer Farbe dargestellt sind, sind für die Aufbereitung der Request-Parameter und etwaiger Nutzdaten im Request zuständig. Die violette Aktivität stellt die eigentliche Abarbeitung der Serveranfrage dar und umfasst die Funktionalität

des BACnet/WS Servers. Das Aufbereiten der Response-Daten wird mit den hellgrünen Aktivitäten dargestellt. Das JSONP-Format kann nicht für die Kodierung von Nutzdaten im Request verwendet werden, da dieses Format lediglich für Response-Daten vorgesehen ist. Es wird nicht immer ein Response generiert, welcher Data Items als Nutzdaten enthält, da z.B. Manipulationsanfragen, Funktionsaufrufe oder Fehlerzustände keine Rückgabewerte liefern. In diesen Fällen wird lediglich ein passender Statuscode erzeugt und dem Client zurückgeliefert. Die Unterstützung der verschiedenen Formate über Marshalling hat den Vorteil, dass für ein zusätzlich zu unterstützendes Format nur zwei weitere Klassen, nämlich ein Marshaller und ein Unmarshaller implementiert werden müssen. Der Server muss sich nur mehr darum kümmern, dass die richtigen Implementierungen für Marshalling bzw. Unmarshalling verwendet werden. Die eigentliche Funktionalität ist somit vom verwendeten Format unabhängig und es ist mit relativ wenig Aufwand möglich ein zusätzliches Format zu unterstützen. Die Inhalte der Dateien `definitions.xml` und `networks.xml` entsprechen dem XML-Format des BACnet/WS Standards. Durch diesen Umstand hat man zusätzlich den Vorteil, dass der XML-Unmarshaller auch für das Auslesen der Netzwerkdefinition verwendet werden kann, wodurch Implementierungsaufwand eingespart kann. Man erhält dadurch auch die Möglichkeit, bei Bedarf, die Formate für die Definition von Netzwerken erweitern zu können.

4.2 Implementierung

Dieser Abschnitt dient dazu, um die wichtigsten Teile der Beispielimplementierung genauer zu erklären. Dabei werden Ausschnitte des Quellcodes gezeigt und anschließend diskutiert.

4.2.1 REST-Schnittstelle

Listing 4.4: BACnetWSREST.java

```
1  @Path("/")
2  public class BACnetWSREST {
3        @Context
4        private ServletContext servletContext;
5
6        @Path("/")
7        public DataItemREST getDataItem() {
8              Network network = getNetwork();
9              DataItem rootDataItem = network.getRoot();
10             return new DataItemREST(network, rootDataItem);
11       }
12       /* ... */
13 }
```

Wie bereits in Abschnitt 4.1 erwähnt, wurde die REST-Schnittstelle über die Java API JAX-RS implementiert. JAX-RS ermöglicht es, auf entsprechende Ressourcenanfragen

über annotierte Java-Methoden reagieren zu können. In Listing 4.4 ist ein Ausschnitt der Klasse `BACnetWSREST` dargestellt, welche den Einsprungspunkt der REST-Schnittstelle darstellt. Damit die JAX-RS Implementierung weiß, welche Klasse oder Methode für einen Pfad, der eine Ressource adressiert, zuständig ist, wird die `Path`-Annotation verwendet. Die Annotation kann sowohl auf Klassenebene als auch auf Methodenebene verwendet werden. Annotiert man eine Klasse z.B. mit `@Path("/ressource")` dann ist sie für alle Ressourcen zuständig, die über die Adresse `http://localhost:8080/ressource/` aufgerufen werden. Geschwungene Klammern haben innerhalb der `Path`-Annotation eine besondere Bedeutung und dienen als Platzhalter für beliebige Zeichenketten. Wird z.B. eine Methode innerhalb der vorher erwähnten Klasse mit `@Path("/{name}")` annotiert, dann ist diese Methode für alle Ressourcen zuständig, die über die Adresse `http://localhost:8080/ressource/beliebigezeichenkette` aufgerufen werden. Möchte man auf den verwendeten Namen innherhalb der Methode zugreifen, dann kann man einen Parameter definieren, der die Annotation `@PathParam("name")` erhält. Der Name `beliebigezeichenkette` wird dann von der JAX-RS Implementierung bei Aufruf der vorher genannten Adresse als Parameter übergeben. Die Annotation `@Path("/")` vor der Methode `getDataItem()` bedeutet, dass Aufrufe ab der Basisadresse des Webservices in erster Linie an diese Methode weitergeleitet werden. D.h., bei einem Aufruf von `http://localhost:8080/bacnetws-rest-server-web/bacnetws/...` wird als erstes die Methode `getDataItem()` aufgerufen, welche ein Objekt vom Typ `DataItemREST` zurückliefert. Dabei muss die JAX-RS Implementierung eindeutig entscheiden können, welche Methode für den angegebenen Pfad aufgerufen werden soll, d.h., die angegebenen Pfade in der `Path`-Annotation müssen innerhalb der Klasse eindeutig sein. Je nach Netzwerkdefinition kann die Tiefe der Baumdarstellung beliebig groß werden, wodurch ein Mechanismus notwendig wird, um beliebig tiefe Pfade über REST ansprechen zu können. Die gewünschte Funktionalität wird durch so genannte *Subresource Locators* ermöglicht, indem eine Methode eine *Subresource* zurückliefert, also ein Objekt, welches ebenfalls über JAX-RS Annotationen in der Klassendefinition verfügt. In unserem Fall ist die Methode `getDataItem()` so ein Subresource Locator, da sie die Subresource `DataItemREST` zurückliefert. Die Methode stellt somit die Wurzel der REST-Schnittstelle dar, die ein Absteigen im Baum erst ermöglicht. In Listing 4.5 ist ein Ausschnitt der Klasse `DataItemREST` dargestellt, der die Methode `getChild(String)` zeigt.

Die Methode ist notwendig, um zum gewünschten Kindelement des Netzwerks zu gelangen. Dabei wird die Methode `getChild(String)` pro Kindelement im Pfad einmal aufgerufen und liefert rekursiv eine neue Instanz der Subressource `DataItemREST` zurück, wodurch das Absteigen in einem beliebig tiefen Baum ermöglicht wird. Der Parameter `childName` wird vom REST-Framework befüllt und stellt den Namen des Kindes in der aktuellen Rekursionsebene dar. Die Methode sucht im aktuellen Data Item das entsprechende Kind-Element und instanziert ein neues Objekt vom Typ `DataItemREST` mit dem gefundenen Kindelement als Parameter im Konstruktor. Wurde das letzte Element des Pfades verarbeitet, ist man beim tiefsten Kindelement angelangt und es wird die Methode `get` aufgerufen, sofern keine andere Methode für den aktuellen Request

Listing 4.5: DataItemREST.java - getChild

```
1 @Path("/{childName}")
2 public DataItemREST getChild(@PathParam("childName") String childName) {
3       for (DataItem dataItem : dataItem.getChildren()) {
4             if (dataItem.getName().equals(childName)) {
5                   return new DataItemREST(network, dataItem);
6             }
7       }
8       return null;
9 }
```

heranzuziehen ist. Die Methode ist in Listing 4.6 dargestellt. Die Daten des Requests werden in einem Objekt vom Typ `RequestInfo` zusammengefasst und weitergereicht. Der Response wird direkt an den Client weitergegeben und besteht im Normalfall aus einem Data Item, welches im Ausgabeformat vorliegt.

Listing 4.6: DataItemREST.java - get

```
1 @Path("/")
2 public Response get(@Context UriInfo uriInfo, @Context Request request) {
3       RequestInfo requestInfo = createRequestInfo(uriInfo, request);
4       Response response = RESTBacnetWSService.getInstance().service(
            requestInfo);
5       return response;
6 }
```

Die Service-Methode `getValue` aus dem BACnet/WS Standard ist in Listing 4.7 dargestellt und ermöglicht das Abrufen des aktuellen Werts eines Metadatums. Die Adressierung erfolgt so, dass der Name des Metadatums mit vorangestelltem `@`-Symbol an die Adresse des Data Items angehängt wird. z.B. `.../dataItem/@metadataName`. Standardmäßig wird der Wert als Data Item im XML-Format zurückgeliefert, es besteht aber die Möglichkeit, durch Angabe des Parameters `alt=plain`, das Ergebnis in Textform zu erhalten.

Listing 4.7: DataItemREST.java - getValue

```
1 @GET
2 @Path("/@{metadataName}")
3 public Response getValue(@Context UriInfo uriInfo, @Context Request request,
      @PathParam("metadataName") String metadataName) {
4       RequestInfo requestInfo = createRequestInfo(uriInfo, request,
            metadataName);
5       Response response = RESTBacnetWSService.getInstance().service(
            requestInfo);
6       return response;
7 }
```

Anhand der bisherigen Listings kann man bereits erkennen, dass die Klassen `BACnetWSREST`

und `DataItemREST` lediglich als Schnittstelle zwischen REST und der eigentlichen Anwendungslogik des BACnet/WS Servers dienen. Die Klassen enthalten bis auf Weiterleitungen von Daten und das Zusammenfassen von Kontext-Variablen und Parametern zu `RequestInfo`-Objekten, keinerlei zusätzliche Logik. Damit die Funktionalität des Servers keinen direkten Kontext zur REST-Schnittstelle besitzt, wird noch eine zusätzliche Abstraktionsschicht verwendet, die sich lediglich um die Umwandlung der Request- und Response-Objekte kümmert. D.h., REST-spezifische Instanzen vom Typ `RequestInfo` werden vor Übergabe an die Anwendungslogik in REST-unabhängige Objekte transformiert. Nach Abarbeitung des Requests durch die Anwendungslogik muss das REST-unabhängige Response-Objekt in ein REST-spezifisches Response-Objekt umgewandelt werden. Ein Ausschnitt dieser Abstraktionsschicht ist in Listing 4.8 dargestellt.

Listing 4.8: RESTBacnetWSService.java

```
1 public Response service(RequestInfo requestInfo) {
2     /* ... */
3     bacnetws.rest.server.web.service.request.Request bacnetWSRequest =
          getRestRequestCreator().createRequest(requestInfo);
4     bacnetws.rest.server.web.service.request.Response response =
          getBacnetWSService().service(bacnetWSRequest);
5     return getRestResponseCreator().createResponse(requestInfo.
          getNetworkContext(), response);
6     /* ... */
7 }
```

Die beiden Klassen `RESTRequestCreator` und `RESTResponseCreator`, die in Zeile 3 und 5 verwendet werden, übernehmen die Aufgabe der Umwandlung für die Anwendungslogik. Der in Abschnitt 4.1 besprochene Marshalling-Mechanismus wird ebenfalls von diesen beiden Klassen übernommen, sodass die Anwendungslogik nicht mehr vom verwendeten Format abhängig ist. Im Falle von Funktionsaufrufen übernimmt der `RESTRequestCreator` auch das Parsen und Validieren der Parameter. Die eigentliche Anwendungslogik wird in Zeile 4 über die Methode `service` aufgerufen und erhält Informationen über den Request bereits in REST-unabhängiger Form. Abschließend ist in Abbildung 4.4 das besprochene Schichtenkonzept und der Fluss der Daten nochmals anschaulich zusammengefasst. Bei einem Request werden die Daten von der REST-Schnittstelle über die Abstraktionsschicht zur Anwendungslogik weitergereicht und die Response-Daten werden über den umgekehrten Weg wieder zum Client transportiert.

4.2.2 Metadaten

Ein weiteres wichtiges Konzept, welches in diesem Abschnitt behandelt wird, ist der Umgang mit Metadaten in der Referenzimplementierung. In BACnet/WS ist die technische Handhabung der Metadaten relativ komplex, da es Metadaten gibt, die vererbt werden können und welche die nicht vererbt werden. Beispiele für nicht vererbbare Metadaten sind u.a. `id`, `name`, `parent`, `type` und `extends`. Ein Data Item kann auf einen vordefinierten Typ verweisen, welcher bereits einige Metadaten festlegt. Bei strukturierten

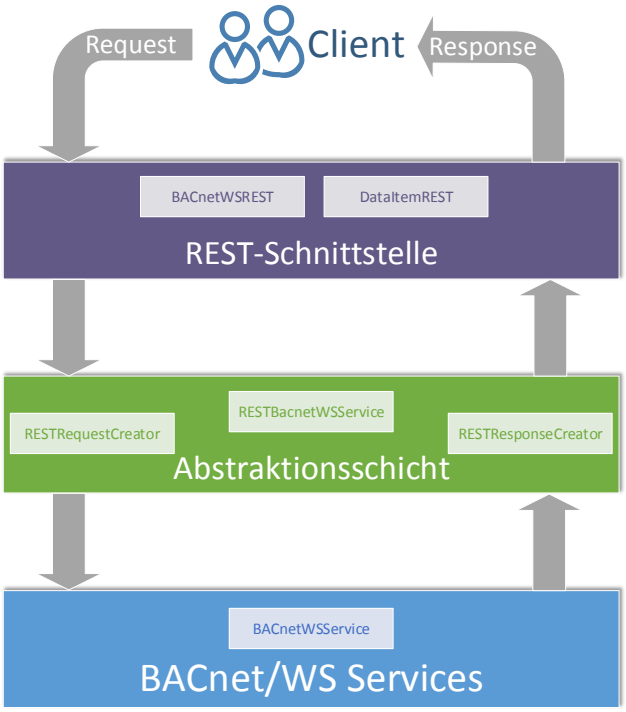

Abbildung 4.4: Schichtenkonzept

Typen, wie z.B. Object oder Interface, werden zusätzlich die vorgesehenen Kindelemente festgelegt. Wurden die vordefinierten Metadaten im Rahmen der Instanzierung nicht überschrieben, dann gelten sie weiterhin auch für die erstellte Instanz. Das selbe Verfahren wird auch bei Vererbungshierarchien angewendet. D.h., vordefinierte Metadaten und festgelegte Kindelemente werden auch über die Vererbungshierarchie nach unten propagiert, sofern sie nicht in der Instanz selbst oder weiter unten in der Vererbungshierarchie überschrieben werden. Zusätzlich sind auch allgemeine Standardwerte für Metadaten vorgesehen, die wirksam werden, wenn kein Wert in der aktuellen Instanz, in der Typdefinition oder an anderer Stelle in der Vererbungshierarchie gesetzt ist. Beispielsweise werden in der Referenzimplementierung die Standardwerte für `readable` bzw. `writable` mit `true` bzw. `false` festgelegt.

Aufgrund der Komplexität wurde die Verwaltung der Metadaten in eine eigene Klasse mit dem Namen `MetadataHolder` ausgelagert. Diese Klasse kümmert sich um die o.g. Funktionalitäten und wird von den Data Item Implementierungen verwendet. Die Methode `getMetadata` der Klasse `MetadataHolder` ist in Listing 4.9 dargestellt. Sie kümmert sich um die Ermittlung des Metadatums mit dem Namen `metadataName`, der als Parameter übergeben wird. In Zeile 2 wird überprüft, ob das Metadatum dem BACnet/WS

Listing 4.9: MetadataHolder.java - getMetadata

```
1  public <E> Metadata<E> getMetadata(String metadataName) {
2        if (isMetadataNotKnown(metadataName)) {
3           return null;
4        }
5
6        Metadata<?> foundMetadata =
7           metadataMap.get(metadataName);
8
9        if (foundMetadata == null) {
10             return null;
11       } else if (isMetadataValueSet(foundMetadata)) {
12             return (Metadata<E>) foundMetadata;
13       } else if (isInheritableMetadata(foundMetadata.getMetadataInfo())) {
14             Metadata<?> inheritedMetadata = getInheritedMetadata(metadataName
                   );
15
16             if (isMetadataValueSet(inheritedMetadata)) {
17                   return (Metadata<E>) inheritedMetadata;
18             }
19       }
20       return (Metadata<E>) getDefaultMetadata(foundMetadata);
21 }
```

Server generell bekannt ist und die Abarbeitung wird abgebrochen, wenn es sich um ein unbekanntes Metadatum handelt. Alle Metadaten, die im BACnet/WS Server verwendet werden, müssen vorher in einer globalen Instanz der Klasse `MetadataInfo` mit ihren Metainformationen bekannt gemacht werden, damit sie auch vom `MetadataHolder` verwendet werden können. Dies ist unbedingt notwendig, da die Klasse `MetadataInfo` wichtige Metainformationen, wie z.B. die Vererbbarkeit von Metadaten oder ob ein Metadatum optional ist oder nicht, verwaltet. Die Klasse `MetadataInfo` wurde in der Referenzimplementierung eingeführt, damit die Metainformationen der Metadaten überall dort abgefragt werden können, wo sie benötigt werden. Ist das Metadatum generell bekannt, dann wird in den Zeilen 6 und 7 das Metadatum mit dem Namen `metadataName` aus dem aktuellen Kontext ermittelt. Die Klasse Metadata enthält Metainformationen und den eigentlichen Wert des Metadatums. Wurde kein Objekt vom Typ `Metadata` gefunden, dann wird in Zeile 10 die weitere Abarbeitung abgebrochen. Das trifft zu, wenn das Metadatum mit dem Namen `metadataName` im aktuellen Kontext nicht bekannt ist und somit nicht verwendet werden kann. Z.B. ist das Metadatum `maximum` für den Typ `String` nicht zulässig, findet jedoch beim Typ `Integer` Verwendung. D.h., vor der Verwendung des `MetadataHolder` müssen alle Metadaten bekannt gegeben werden, die für den jeweiligen Typ von Data Item verwendet werden dürfen. Wenn die Instanz von Data Item den Wert für das Metadatum selbst definiert, dann wird in Zeile 12 dieses Metadatum zurückgeliefert. Handelt es sich um ein vererbbares Metadatum, dann wird in den Zeilen 14 - 18 ein entsprechendes Metadatum in der Typdefinition bzw. in der Vererbungshierarchie gesucht. Konnte bisher kein Metadatum für das Data Item ermittelt

werden, dann wird in Zeile 20, sofern vorhanden, der Standardwert zurückgegeben. Jede Data Item Implementierung verwendet eine eigene Instanz des `MetadataHolder` und leitet jeden Getter- und Setter-Aufruf an diesen weiter. Man beachte, dass durch diese Vorgehensweise der gesamte Zustand des Data Items im `MetadataHolder` gehalten wird und die Methoden in der Klasse `DataItem` lediglich den Zugriff auf diese Zustände vereinfachen. Im Gegensatz zur Definition wird der Wert im Attribut `value` aus Gründen der Einfachheit als Metadatum behandelt und analog über den `MetadataHolder` verwaltet.

4.3 Fallstudie

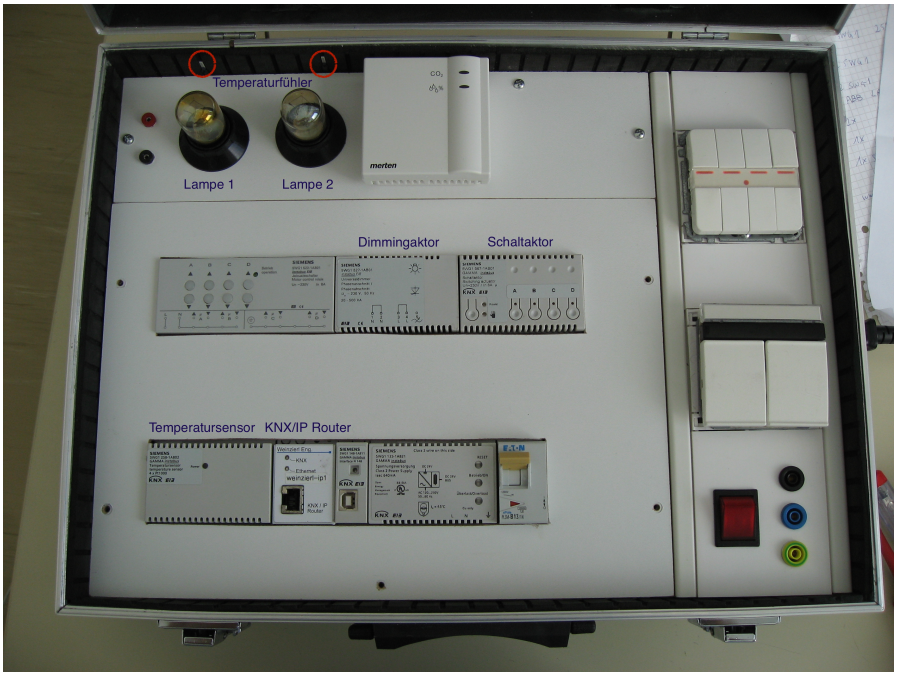

Abbildung 4.5: Aufbau des KNX-Netzwerks

In diesem Abschnitt wird die Interaktion eines Web-Clients mit einem KNX-Beispielnetzwerk über den BACnet/WS Server anhand drei ausgewählter Anwendungsfälle demonstriert. Der Aufbau des KNX-Netzwerks ist in Abbildung 4.5 dargestellt. Die drei vorgestellten Anwendungsfälle sind *Licht einschalten*, *Licht dimmen* und *Temperatur abfragen*. Das KNX-Netzwerk enthält u.a. die Komponenten *Schaltaktor*, *Dimmingaktor* und *Temperatursensor*, welche für die betrachteten Anwendungsfälle erforderlich sind. Man beachte, dass die Temperaturfühler so angeordnet sind, dass nach dem Einschalten einer Lampe sofort eine Temperaturerhöhung registriert wird. Die wichtigste Komponente ist jedoch der *KNX/IP Router*, der es erst ermöglicht, eine Verbindung zwischen dem BACnet/WS Server und dem KNX-Netzwerk über das IP-Protokoll her-

zustellen. In Abbildung 4.5 sind die relevanten Komponenten für die Anwendungsfälle zur besseren Lesbarkeit beschriftet. Die Kommunikation mit dem KNX-Netzwerk über den BACnet/WS Server ist in Abbildung 4.6 schematisch dargestellt. Das Modell des KNX-Netzwerks besteht aus einem Netzwerk `Office` und enthält eine funktionale View mit dem Namen `functional`. Diese View fasst alle Komponenten des Netzwerks nach ihrer Funktion wie z.B. Licht oder Temperatur zusammen. Beispielsweise verweist die Adresse `$base/.trees/networks/Office/views/functional/containers/Light` auf Komponenten, die für die Beleuchtung im KNX-Netzwerk zuständig sind. Der Platzhalter `$base` steht für die Adresse des Servers.

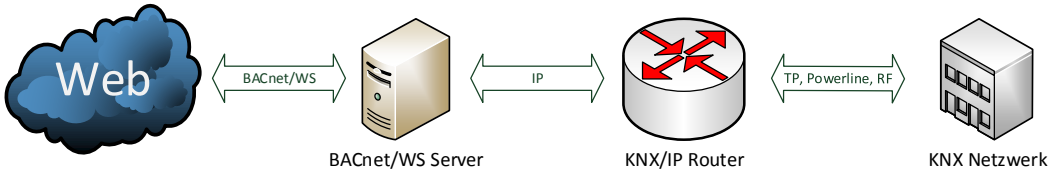

Abbildung 4.6: Schematische Darstellung der Kommunikation mit dem Netzwerk

4.3.1 Anwendungsfall: Licht einschalten

In diesem Anwendungsfall soll die Lampe 2 aus Abbildung 4.5 von einem Web-Client über die BACnet/WS Schnittstelle eingeschaltet werden. Der Datenpunkt zum Einschalten des Lichts ist im Beispielnetzwerk über die folgende Adresse erreichbar: `$base/.trees/networks/Office/views/functional/containers/Light/containers/Light onoff/function`. Der Aufbau der Adresse ergibt sich dabei aus der View-Hierarchie.

Ruft man die o.g. Addresse über den Internetbrowser auf, erhält man ein Ergebnis, welches in Listing 4.10 dargestellt ist.

Listing 4.10: Datenpunkt Switch, Channel A

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Object description="On / Off" displayName="Switch, Channel A" id="P-0341-0
    _DI-3_M-0001_A-9803-03-3F77_O-3_R-4" name="function" nodeType="Point"
    readable="false" type="at.ac.tuwien.auto.bas.DPST-1-1" writable="true">
  <List name="types" writable="false">
    <String name="1" value="DPST-1-1" writable="false">
  </List>
  <Enumerated name="priority" value="Low" writable="false">
  <Boolean name="value" readable="false" writable="true"/>
  <Enumerated name="encoding" readable="false" type="at.ac.tuwien.auto.bas.
      Encoding" value="off" writable="true"/>
</Object>
```

Der Zustand des Schaltaktors kann zwar über das Web-Interface ausgelesen werden, jedoch liefert ein Lesezugriff am KNX-Netzwerk kein Ergebnis zurück, da dieser Datenpunkt lediglich für Schreibzugriffe konfiguriert ist. Das Metadatum `readable` ist

deshalb auf `false` gesetzt. Das Licht, das über den Kanal des Aktors angeschlossen ist, kann nun eingeschaltet werden, indem das Metadatum `value` des Kindelements `value` auf den Wert `true` geändert wird. Um den Wert zu ändern, wird ein PUT-Request über die Adresse `$base/.trees/networks/Office/views/functional/containers/Light/containers/Light onoff/function/value/@value` mit den Daten aus Listing 4.11 ausgeführt. Nach dem Absetzen des Requests schickt der Server den aktualisierten Wert an den KNX/IP-Router und die Lampe 2 beginnt zu leuchten.

Listing 4.11: Request zum Einschalten der Lampe 2

```
<?xml version="1.0" encoding="UTF-8"?>
<Boolean name="value" value="true" />
```

4.3.2 Anwendungsfall: Licht dimmen

In diesem Anwendungsfall soll die Lampe 1 über die Dimming-Funktion des Dimming-Aktors aktiviert werden. Der Datenpunkt zum Dimmen des Lichts ist in unserem Beispiel über die Adresse `$base/.trees/networks/Office/views/functional/containers/Light/containers/Light Dimming/function` erreichbar. Ruft man diese Adresse über einen GET-Request auf, dann erhält man das Ergebnis aus Listing 4.12. Im Gegensatz zum Einschalten des Lichts ist beim Dimmen ein Funktionsaufruf notwendig. Funktionsaufrufe sind im BACnet/WS Standard definiert und erfolgen durch Angabe des Funktionsnamens und der Parameter innerhalb der URL. Es handelt sich also beim Aufruf der Funktion um einen GET-Request und nicht um einen PUT-Request. Beim Dimming sind die zwei Funktionsaufrufe `increase` und `decrease` möglich, die jeweils einen ganzzahligen Wert im Intervall 0 und 100 als Parameter akzeptieren. Bei Aufruf von `increase` wird die Helligkeit um den angegebenen Prozentwert erhöht und bei Aufruf von `decrease` entsprechend verringert.

Listing 4.12: Datenpunkt Dimming

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Object description="Brighter / Darker" displayName="Dimming" id="P-0341-0_DI
    -2_M-0001_A-6102-01-A218_O-1_R-1" name="function" nodeType="Point"
    readable="true" type="at.ac.tuwien.auto.bas.DPST-3-7" writable="false">
  <List name="types" writable="false">
    <String name="1" value="DPST-3-7" writable="false" />
  </List>
  <Enumerated name="priority" value="Low" writable="false" />
</Object>
```

Die eingeschaltete Lampe 1 kann nun durch Aufruf der Funktion `increase` mittels Adresse `$base/.trees/networks/Office/views/functional/containers/Light/containers/Light Dimming/function/increase(10)` zum Leuchten gebracht werden.

4.3.3 Anwendungsfall: Temperatur abfragen

Bei diesem Anwendungsfall wird die Temperatur eines Kanals des Temperatursensors über die BACnet/WS Schnittstelle abgelesen. Um die Temperatur abgreifen zu können, muss der Wert des folgenden Datenpunktes abgefragt werden: `$base/.trees/networks/Office/views/functional/containers/Temperature/containers/Temperature 1/function`. Das Ergebnis ist in Listing 4.13 abgebildet.

Listing 4.13: Datenpunkt Temperature, Channel A

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Object description="C-value (EIS5)" displayName="Temperature, Channel A" id=
    "P-0341-0_DI-11_M-0001_A-9814-01-5F7E_O-0_R-2" name="function" nodeType="
    Point" readable="true" type="at.ac.tuwien.auto.bas.DPST-9-1" writable="
    false">
  <List name="types" writable="false">
  <String name="1" value="DPST-9-1" writable="false"/>
  </List>
  <Enumerated name="priority" readable="true" value="Low" writable="false"/>
  <Real name="value" readable="true" value="25.92" writable="false"/>
</Object>
```

Der Temperaturwert kann vom Metadatum `value` des Kindelements `value` abgelesen werden. Möchte man den Temperaturwert direkt erhalten, dann ist der Aufruf der folgenden Adresse notwendig: `$base/.trees/networks/Office/views/functional/containers/Temperature/containers/Temperature 1/function/value/@value`. Der Wert wird dann als Data Item vom Typ `Real` in XML zurückgeliefert. Möchte man das Ergebnis als einfachen Text erhalten, dann ist die Angabe des Parameters `alt=plain` notwendig.

KAPITEL 5

Zusammenfassung und Reflexion

5.1 Zusammenfassung

Gebäudeautomation gewinnt in der heutigen Zeit immer mehr an Bedeutung. Dabei stehen der Komfortgewinn und die Erhöhung der Energieeffizienz im Vordergrund, wodurch die Menschen und letztlich auch die Umwelt profitieren. In dieser Arbeit wird der Leser in die Thematik Gebäudeautomation eingeführt und ein Bezug zum Thema IoT hergestellt. Der Begriff Gebäudeautomation wird definiert und die geschichtliche Entwicklung bis hin zu modernen GAS erläutert. Ein großes Problem heutiger GAS ist die fehlende Interoperabilität zwischen Systemen unterschiedlicher Hersteller und Standards. Die vielen Standards haben sich im Laufe der Zeit entwickelt, da bisher das Hauptaugenmerk nicht auf die Kommunikation mit verschiedenen Systemen sondern eher auf Wirtschaftlichkeit und Leistungsfähigkeit gelegt wurde und die Systeme deshalb für spezielle Anwendungsgebiete optimiert wurden [2]. In dieser Arbeit wird ein möglicher Ansatz vorgestellt, der dieses Problem über einen BACnet/WS Server löst und somit die IT-Integration mit externen Systemen ermöglicht. Es wird deshalb das Objektmodell von BACnet/WS [3] vorgestellt und auf grundlegende Konzepte, wie Daten in BACnet/WS strukturiert sind, genauer eingegangen. Dabei wird der grundsätzliche Aufbau der Daten in Form eines Metamodells erläutert. Ein GGM dient dazu, um GAS möglichst allgemein beschreiben zu können, ohne dabei zu detailliert auf spezifische Technologien einzugehen. Aus diesem Grund wird eine mögliche Variante eines generischen Modells vorgestellt, um den Lösungsansatz möglichst technologieunabhängig beschreiben zu können. Des weiteren wird ein Ansatz vorgestellt, wie Instanzen des GGM in Instanzen des BACnet/WS Objektmodells überführt werden können und die einzelnen Schritte des Mappings werden anhand zahlreicher UML-Diagramme erklärt. Im Rahmen dieser Arbeit wurde auch eine BACnet/WS Referenzimplementierung erstellt, um einen möglichen Ansatz für die Integration eines GAS zu präsentieren. Die dafür eingesetzten Technologien und Tools werden vorgestellt und diskutiert. Da Webservices in BACnet/WS auf dem REST-Paradigma basieren wird darauf etwas genauer eingegangen, um die dahinterliegenden Konzepte

zu verdeutlichen. Damit der Leser einen tieferen Einblick in die Implementierung erhält werden die Entwicklungsumgebung und die einzelnen Module und deren Zweck beschrieben. Auch die Architektur und die wichtigsten Teile des Quellcodes werden genauer erklärt und anhand von Grafiken visuell dargestellt. Im Rahmen einer Fallstudie wird die Interaktion eines Web-Clients mit einem KNX-Beispielnetzwerk über den BACnet/WS Server anhand von drei ausgewählten Anwendungsfällen demonstriert.

5.2 Vergleich mit anderen Arbeiten

In diesem Abschnitt werden neben der vorliegenden Arbeit auch andere Arbeiten vorgestellt, die sich mit ähnlichen Themengebieten auseinandersetzen.

Jianbo Bai et al. heben in ihrer Arbeit [25] die enorme Wichtigkeit von Webservices für die Integration von GAS hervor. Sie erwähnen, dass Webservices drei unterschiedliche Probleme bei der Integration von GAS lösen können. Das erste Problem stellt die Integration zwischen GAS und existierenden Enterprise-Anwendungen dar. Die Integration von GAS, die auf unterschiedlichen Protokollen basieren, wird als zweites Problem erwähnt. Als drittes Problem wird die Integration von Subsystemen wie z.B. Klimatechnik-Steuerung, Beleuchtungs-Steuerung oder Feuer-Alarmierung angesprochen, die auf verschiedenen Protokollen basieren. Webservices haben laut den Autoren deshalb so eine hohe Bedeutung, da diese Technologien mittlerweile sehr weit verbreitet sind und dadurch eine plattformunabhängige Kommunikation zwischen verschiedenen Systemen realisiert werden kann. Da Webservices auch in Enterprise-Anwendungen sehr weit verbreitet sind, ist eine nahtlose Anbindung von GAS möglich, wodurch das erste Integrationsproblem gelöst wird. Webservices ermöglichen auch die Integration verschiedener Protokolle ohne zusätzliche Gateway Einrichtungen zu benötigen, wodurch auch das zweite und dritte Problem durch den Einsatz von Webservices gelöst werden kann.

Im Rahmen einer Arbeit von Kastner et al. [26] wurde ein Prototyp entwickelt, der die Integration eines KNX-Netzwerks über einen BACnet/WS Server ermöglicht. Wie in der Referenzimplementierung wurde der Prototyp für Java entwickelt und kommuniziert über die Java-Bibliothek Calimero mit dem KNX-Netzwerk. Im Gegensatz zur Referenzimplementierung basiert der Prototyp jedoch auf SOAP, da der BACnet/WS Standard aus dem Jahr 2004 noch SOAP als Grundlage für die Webservice-Schnittstelle vorgesehen hatte.

Eine weitere interessante Arbeit ist das *IoTSyS*-Projekt [27], welches am Institut für Rechnergestützte Automation der Technischen Universität Wien erstellt wurde. Es handelt sich dabei um eine Middleware, die die Integration von GAS ins IoT ermöglichen soll. Die IT-Integration erfolgt über einen oBIX-Gateway-Server der den Zugriff auf das GAS ermöglicht. Für die Feld- und Automationsebene stellt die Middleware bereits Konnektoren für KNX, BACnet oder Wireless M-Bus zur Verfügung. Der KNX-Konnektor basiert hier ebenfalls auf Calimero. Die IoTSyS-Middleware ist aber nicht auf die vorgegebenen Technologien beschränkt, sondern ermöglicht die Implementierung weiterer Konnektoren und gerätespezifischer Schnittstellen, die einfach in die Middleware integriert werden können.

Von Jung et al. [28] wird eine andere Herangehensweise präsentiert. In der Arbeit wird ein IPv6 basierter Ansatz zur Integration von einzelnen Komponenten eines GAS ins IoT vorgestellt. Die Kernidee dieser Arbeit ist die Möglichkeit, jede Komponente eines GAS über eine eigene IPv6 Adresse aus dem Internet ansprechen zu können. Dabei erfolgt die Kommunikation über einen oBIX-Webservice, der über die IPv6 Adresse der Komponente aufgerufen werden kann. Die Grundlage für die Darstellung der Netzwerk- oder Komponentenstruktur bilden sogenannte *Contracts*, die über das oBIX Objektmodell definiert werden und über die Webservice-Schnittstelle zugänglich sind. Die Contracts sind als Schnittstellen zu verstehen, die beschreiben, wie mit den einzelnen Komponenten kommuniziert werden kann. Das BACnet/WS Pendant zu den oBIX-Contracts wäre der Definition Context, der in unserer Arbeit für einen ähnlichen Zweck verwendet wurde. In der Arbeit von Jung et. al wird ein s.g. *IPv6 multi-protocol gateway stack* beschrieben, der einerseits auf den Komponenten selbst oder in einem Gateway installiert werden kann. An oberster Stelle des Stacks steht der oBIX-Webservice mit den definierten Contracts für die Kommunikation. Damit die Webservice-Kommunikation auch auf leistungsschwachen Geräten möglich ist, werden Protokolle definiert, die für solche Geräte optimiert sind. Für den Nachrichtenaustausch wird wahlweise HTTP oder CoAP eingesetzt. CoAP stellt dieselben Funktionen wie HTTP zur Verfügung, basiert aber auf dem *User Datagram Protocol* (UDP) und ermöglicht dadurch den effizienteren Austausch von Nachrichten, was vor allem auf Geräten mit niedriger Leistung ein großer Vorteil ist. Als Datenformat wird an Stelle von XML das EXI Format verwendet, welches es ermöglicht, XML-Inhalte effizienter austauschen zu können. Um ein IoT basierend auf dem IPv6 multi-protocol gateway stack zu realisieren, müssten sämtliche Komponenten um diesen Netzwerkstack erweitert werden. Da dies aufgrund der hohen Anzahl an bereits verfügbaren Komponenten nicht möglich ist, wird in der Arbeit auch eine Lösung zur Integration dieser Komponenten angeboten. Die Anbindung erfolgt dabei über einen zentralen Gateway, der für jede Komponente den erwähnten Netzwerkstack inklusive dem oBIX-Webservice zur Verfügung stellt. Der Webservice kann dann über die IPv6-Adresse der Komponente angesprochen werden, d.h., die IPv6-Fähigkeit dieser Geräte wird vom Gateway immitiert.

Wie man anhand der vorgestellten Arbeiten bereits erkennen kann, sind verschiedene Entwicklungen zu beobachten, die das Problem der Integration verschiedener GAS zu lösen versuchen. Es geht in den Arbeiten auch hervor, dass Webservices dabei eine sehr große Rolle spielen. Welche Technologie oder welcher Ansatz sich schlussendlich durchsetzen wird, ist jedoch zurzeit noch nicht absehbar.

5.3 Offene Punkte

In diesem Abschnitt werden Themenpunkte zusammengefasst, die durch die vorliegende Arbeit nicht oder nicht zur Gänze behandelt wurden. Das Ziel dieser Arbeit war die Erarbeitung eines Ansatzes um die Konzepte eines GGM in ein BACnet/WS Objektmodell überführen zu können. Ausgehend von diesem Objektmodell wurde eine Referenzimplementierung des BACnet/WS Standards vorgestellt, welche die Anbindung an ein

KNX-Netzwerk ermöglicht.

Das GGM wurde bewusst einfach gehalten, indem als Netzwerkkomponenten lediglich Datapoints und Devices verwendet wurden, obwohl GAS im Allgemeinen auch weitere Komponenten vorsehen. Das Modell müsste deshalb um weitere Aspekte ergänzt werden, um auch Konzepte anderer GAS integrieren zu können, damit die Abbildung verschiedener Systeme gewährleistet wird.

Als Basis für die Referenzimplementierung wurde im Rahmen dieser Arbeit ein Metamodell erstellt, welches das BACnet/WS Objektmodell widerspiegelt. Das Metamodell stellt jedoch lediglich einen Auszug aus dem Objektmodell von BACnet/WS dar und ist auf jene Elemente fokusiert, die für das Mapping zwischen dem GGM und dem BACnet/WS Objektmodell benötigt werden. Für eine vollständige Referenzimplementierung müsste das Metamodell um die noch fehlenden Elemente erweitert werden.

Die Referenzimplementierung ist keinesfalls vollständig, da nur Teile des BACnet/WS Standards umgesetzt wurden. Der Standard definiert einige REST-Schnittstellen und -Funktionen, welche die Abfrage und die Manipulation der Daten im Netzwerk ermöglichen. Es wurden jedoch nur einige dieser Schnittstellen und Funktionen implementiert. Sicherheitsaspekte, Concurrency Control und die Historisierung von Daten wurden aus Gründen der Einfachheit ebenfalls vorerst nicht berücksichtigt, sind aber für eine vollständige Referenzimplementierung von großer Bedeutung. Des Weiteren beschränkt sich der BACnet/WS Server nur auf die wichtigsten Datenpunkttypen von KNX und die Implementierung wurde nur mit einem KNX-Netzwerk im Rahmen der Fallstudie getestet. Deshalb müsste die Implementierung um weitere Standards ergänzt werden, um die Integration verschiedener Systeme zu ermöglichen.

Die vorliegende Arbeit ist als Basis zu sehen, in der ein möglicher Ansatz beschrieben wird, um GAS an externe Systeme oder auch andere GAS anbinden zu können. Die Ansätze dieser Arbeit können so weiterentwickelt werden, dass die Integration von GAS mit Komponenten verschiedenster Hersteller über BACnet/WS ermöglicht wird.

Abbildungsverzeichnis

Listings

KAPITEL 6

Abkürzungsverzeichnis

WS Webservice

DDC Direct Digital Control

IP Internet Protocol

IoT Internet of Things

GAS Gebäudeautomationssystem

IANA Internet Assigned Numbers Authority

GGM generisches Gebäudeautomationsmodell

HTTP Hypertext Transfer Protocol

CoAP Constrained Application Protocol

UDP User Datagram Protocol

EXI Efficient XML Interchange

ASHRAE American Society of Heating, Refrigerating and Air-Conditioning Engineers

REST Representational State Transfer

SOAP Simple Object Access Protocol

XML Extensible Markup Language

JSON JavaScript Object Notation

JSONP JSON with padding

NAT Network Address Translation

IPv6 IP version 6

IPv4 Internet Protokoll in der Version 4

URI Uniform Resource Identifier

JAX-RS Java API for RESTful Services

HTML Hypertext Markup Language

PDF Portable Document Format

WAR Web Application Archive

TGA technischen Gebäudeautomation

Literaturverzeichnis

[1] Lu Tan and Neng Wang. Future internet: The Internet of Things. In *3rd International Conference on Advanced Computer Theory and Engineering*, volume 5, pages V5–376–V5–380, August 2010.

[2] M. Jung, C. Reinisch, and W. Kastner. Integrating Building Automation Systems and IPv6 in the Internet of Things. In *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 683–688, July 2012.

[3] ASHRAE. A Data Communication Protocol for Building Automation and Control Networks (ANSI/ASHRAE Proposed Addendum am to ANSI/ASHRAE Standard 135-2012).

[4] A.C.W. Wong and A.T.P. So. Building automation in the 21st century. In *Fourth International Conference on Advances in Power System Control, Operation and Management. APSCOM-97. (Conf. Publ. No. 450)*, volume 2, pages 819–824 vol.2, November 1997.

[5] D. Trinchero, R. Stefanelli, D. Brunazzi, A. Casalegno, M. Durando, and A. Galardini. Integration of smart house sensors into a fully networked (web) environment. In *Sensors, 2011 IEEE*, pages 1624–1627, October 2011.

[6] T. Sauter, S. Soucek, W. Kastner, and D. Dietrich. The Evolution of Factory and Building Automation. *Industrial Electronics Magazine, IEEE*, 5(3):35–48, September 2011.

[7] W. Kastner, M. Kofler, M. Jung, G. Gridling, and J. Weidinger. Building automation systems integration into the Internet of Things the IoT6 approach, its realization and validation. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–9, September 2014.

[8] KNX Association. `http://www.knx.org/knx-en/`, Accessed: April 2015.

[9] ASHRAE. A Data Communication Protocol for Building Automation and Control Networks (ANSI/ASHRAE Standard 135-2012).

[10] ZigBee Alliance. ZigBee Specification, September 2012.

[11] OASIS, 2006. oBIX 1.0 Committee Specification 01, December 2006.

[12] M. Neugschwandtner, G. Neugschwandtner, and W. Kastner. Web Services in Building Automation: Mapping KNX to oBIX. In *5th IEEE International Conference on Industrial Informatics*, volume 1, pages 87–92, June 2007.

[13] Geoff Huston. The End of IPv4, Part 2. *The ISP Column*, pages 2012–08, 2012.

[14] C. Zaccone, Y. T'Joens, and B. Sales. Address reuse in the Internet, adjourning or suspending the adoption of IP next generation? In *Networks, 2000. (ICON 2000). Proceedings. IEEE International Conference on*, pages 462–468, 2000.

[15] P. Diogo, L.P. Reis, and N. Vasco Lopes. Internet of Things: A system's architecture proposal. In *9th Iberian Conference on Information Systems and Technologies*, pages 1–6, June 2014.

[16] M. Jakl. REST: Representational State Transfer, University of Technology Vienna. 2008.

[17] Oracle Corporation. Java API for RESTful Services (JAX-RS). `https://jax-rs-spec.java.net`, Accessed: March 2015.

[18] Oracle Corporation. Jersey - RESTful Web Services in Java. `https://jersey.java.net`, Accessed: March 2015.

[19] The Apache Software Foundation. Apache Tomcat. `http://tomcat.apache.org`, Accessed: March 2015.

[20] Pivotal Software. Spring. `https://spring.io`, Accessed: March 2015.

[21] The Apache Software Foundation. Apache Log4j 2. `http://logging.apache.org/log4j/2.x/`, Accessed: March 2015.

[22] Automation Systems Group. Calimero: KNX network access for Java, March 2015.

[23] The Eclipse Foundation. Eclipse. `https://www.eclipse.org`, Accessed: March 2015.

[24] The Apache Software Foundation. Apache Maven. `http://maven.apache.org`, Accessed: March 2015.

[25] Jianbo Bai, Hong Xiao, Xianghua Yang, and Guofang Zhang. Study on integration technologies of building automation systems based on web services. In *International Colloquium on Computing, Communication, Control, and Management*, volume 4, pages 262–266, August 2009.

[26] W. Kastner and S. Szucsich. Accessing KNX networks via BACnet/WS. In *IEEE International Symposium on Industrial Electronics*, pages 1315–1320, June 2011.

[27] Automation Systems Group - Vienna University of Technology. IoTSyS - an integration middleware for the Internet of Things. https://code.google.com/p/iotsys/, Accessed: March 2015.

[28] M. Jung, J. Weidinger, W. Kastner, and A. Olivieri. Building Automation and Smart Cities: An Integration Approach Based on a Service-Oriented Architecture. In *27th International Conference on Advanced Information Networking and Applications Workshops*, pages 1361–1367, March 2013.