600 Jahre Universität Rostock

Traditio et Innovatio

Masterarbeit zum Thema

Automatisierte Topologieerkennung in Feldbussen

Studiengang:	Informatik
Vorgelegt von:	Richard Dabels
Matrikelnummer:	214206647
Bearbeitungszeitraum:	1. November 2019 – 1. Mai 2020
Betreuer:	Dr.-Ing. Thomas Mundt
Erstgutachter:	Prof. Dr. rer. nat. Clemens H. Cap
Zweitgutachter:	Prof. Dr. rer. nat. habil. Andreas Heuer

Lehrstuhl für Informations- und Kommunikationsdienste
Institut für Informatik
Fakultät für Informatik und Elektrotechnik
Universität Rostock

Gedruckt auf säurefreiem und elementar-chlorfrei gebleichtem Papier.

Stand:: 1. Mai 2020

Inhalt

Kurzzusammenfassung

In komplexen Gebäudeautomatisierungssystemen werden Feldbusse eingesetzt, um Geräte wie Sensoren und Aktoren miteinander zu verbinden. Die Natur der Gebäudeplanung und -nutzung sorgt für eine schnell voranschreitende Dokumentenerosion der Planungsunterlagen und auch die Systeme selbst sind des Öfteren durch mangelnde Sicherheitsvorkehrungen geprägt.

Die adäquate Absicherung von Feldbussen erfordert Wissen über die verschiedenen Netzwerkansichten – die *logische Topologie*, die *physische Topologie* und die *funktionale Topologie*. Im Rahmen dieser Arbeit wird ein Modell erarbeitet, welches die Darstellung der genannten Topologien unterstützt. Dabei wird die *logische Topologie* im Detail behandelt und ein Prototyp für dessen Rekonstruktion mit Hilfe passiver Scans für KNX-Netzwerke implementiert.

Abstract

Field busses are used in complex building automation systems to connect devices like sensors and actors. The nature of building planning and its usage causes a rapid advancing document erosion and even the systems themselves are often designed without security in mind.

The adequate protection of field busses requires knowledge of the different network views – the *logical topology*, the *physical topology* and the *functional topology*. As part of this work, a model that supports the representation of each topology is developed. The logical topology is examined in detail and a prototype for its automatic recognition using passive scanning is implemented for KNX networks.

Kapitel 1

Motivation

In diesem Kapitel

Unsere vernetzte Welt befindet sich im stetigen Wandel. Informationen sind so einfach zugänglich, wie noch nie zuvor in der Geschichte der Menschheit. Der überwiegende Großteil von uns besitzt Computer im Taschenformat, welche über Leistungen verfügen, für die vor wenigen Jahrzehnten noch raumfüllende Systeme nötig waren. Informationsaustausch ist für uns mittlerweile selbstverständlich und Technologie erleichtert uns an jeder Stelle den Alltag. Mehr und mehr hören wir von Schlagwörtern wie *Internet of Things* (IoT) und es scheint, als würden alle Geräte „smart“ werden.

Smart Home, beziehungsweise *Gebäudeautomatisierungssysteme* (GAS), ist einer der Bereiche, in denen aktuell viel Entwicklung betrieben wird. Dabei ist die Idee von intelligenten Haushalten nicht neu. Bereits Anfang der 1880er Jahre wurden einfache Thermostate eingesetzt, um die Temperatur in Boiler-Räumen zu kontrollieren [Kat19].

Selbstverständlich haben sich die Ansprüche unserer Gesellschaft heutzutage weit über automatische Temperaturregelung hinaus entwickelt. Wir erwarten weiterhin, dass die Beleuchtung von selbst erkennt, sobald wir den Raum betreten, dass unsere Heizung lernt, wann wir zu Hause sind und wann wir schlafen. Wenn wir aufstehen, soll der Kaffee bereits kochen und beim Heimkommen möchten wir, dass die Tür sich von selbst öffnet oder biometrisch entriegelt werden kann.

Zumindest können wir uns solch eine Zukunft vorstellen. Dass dies bereits möglich ist, sehen wir in einigen anderen Branchen. Zur effizienten Stromversorgung werden von den Kraftwerkbetreibern sogenannte *Smart Grids* benutzt – „Netzwerke, welche die Aktionen aller durch sie verbundenen Nutzer einbeziehen, um Strom nachhaltig, ökonomisch und sicher bereitstellen zu können“ [Our+20]. Sie stellen eine bidirektionale Verbindung zwischen Verbrauchern und Erzeugern her, um die Stromerzeugung zu jedem Moment an dessen Verbrauch anzupassen. Besonders angesichts des steigenden Bedarfs an erneuerbaren Energien ist eine intelligentere Stromversorgung und -Speicherung schlichtweg eine Notwendigkeit [WL18].

Auch in der Automobilindustrie ist die Automatisierung seit Langem eingezogen. Fahrzeuge verfügen heutzutage über eine Reihe von Komfort- und Sicherheitsfunktionen – die meisten wurden dabei innerhalb der letzten 20 Jahre entwickelt. Mittlerweile gehören automatische Fensterheber und Scheibenwischer, Klimaanlage, Ver- und Entriegelung per Funk, Navigationssysteme mit Multimedia-Funktionen sowie Türen, die sich von selbst öffnen und während der Fahrt verschließen sowie vieles mehr zum Industriestandard [Asc14; Knxb]. Und obwohl bereits eine beachtliche Menge an Features genannt ist, wurde die neueste und relevanteste Entwicklung – das *autonome Fahren* – hier noch gar nicht bemerkt.

Diese Entwicklungen zeigen uns vor allem, dass die Automatisierung über großes Potential verfügt. Allein die Energieeffizienz von GAS wäre deren Verwendung wert – in der Europäischen Union sind Gebäude für rund 40% des allgemeinen Energieverbrauchs verantwortlich [Cou10]. In gewerblichen Gebäuden ist es beispielsweise möglich, Energieeinsparungen von bis zu 30% zu erreichen [Kat19].

Ein sich erweiternder Ausbau der IoT-Infrastruktur bringt jedoch auch Gefahren mit sich. Viele Unternehmen kämpfen um ihre Position in dem neu erschlossenen Markt der Automatisierung und nicht selten wird der Aspekt der Sicherheit an dieser

Stelle zweitrangig behandelt. Resultate dieser Praxis sind vielfältig: Unsichere IoT-Geräte, welche im Verbund als Botnets DDoS-Attacken durchführen [Kol+17; Don+18; Sym18], Herzschrittmacher, deren Frequenzen und Stromverbrauch manipuliert werden können [Lar17], Babymonitore mit unverschlüsselten Ad-Hoc-Netzwerken [Tho16], Sicherheitskameras mit frei im Internet abrufbaren Videoübertragungen [Adh13] und sogar Fahrzeuge, welche, mit entsprechender Vorbereitung, vom Scheibenwischer bis zur Lenkung von Angreifern kontrollierbar sind [Dro15]. Mit Malware wie Stuxnet [Gay10; Kus13] und Industroyer [Sch17; AC17] stellten sich Angriffe auf Computersysteme zusätzlich als politische Waffe mit verheerendem Schadenspotential heraus.

1.1 Dokumentenerosion

Die Feldbusse, mit welchen die in GAS eingesetzten Geräte verbunden werden, zeichnen sich durch ihre relative Einfachheit aus. In klassischen Elektroinstallationen werden Geräte per Sternverkabelung an einer zentralen Verteilung verdrahtet. Was in kleineren Gebäuden noch praktisch ist, entwickelt sich bei zunehmender Komplexität jedoch schnell zum administrativen Albtraum. Hier ist es sinnvoller, alle Geräte über ein geteiltes Bussystem miteinander zu verbinden. An einen solchen *Feldbus* können verschiedene Geräte angeschlossen werden und frei miteinander kommunizieren [Knxb].

Moderne GAS entwickeln sich jedoch trotz einfacher Bustechnik mehr und mehr zu komplexen Netzwerken, welche nur mittels vollständiger Dokumentation nachvollziehbar sind. Die *Dokumentenerosion* ist das grundlegende Problem der vorliegenden Arbeit. Damit ist gemeint, dass die Dokumentation komplexer Netzwerke einem raschen Alterungsprozess unterliegt. Nach Wartungsarbeiten, Neukonfigurationen, Erweiterungen etc. des Netzwerkes wird die ursprünglich erstellte Dokumentation oft nicht aktualisiert. Die folgenden Aspekte des Netzwerkes sind dadurch betroffen:

- **physische Topologie**: Die Verkabelung als solche wurde geändert. Der Verlauf dieser im Gebäude ist durch die Dokumentation nicht mehr zurückverfolgbar.

- **logische Topologie**: Die logischen Netzwerksegmente und die Einteilung der Geräte in ihnen wurden neu eingeteilt. Dies erschwert den Umgang mit den vorhandenen Geräten.
- **funktionale Topologie**: Diese Topologie beschreibt Relationen zwischen Geräten. Hier können Probleme entstehen, wenn den installierten Geräten neue Aufgaben zugewiesen werden. Die Beziehung zwischen Geräten ist folglich durch die Gebäudedokumentation nicht mehr nachvollziehbar.

1.2 Netzwerksicherheit

Vor allem jedoch trägt eine unvollständige Dokumentation zu einer beeinträchtigten Netzwerksicherheit bei. Mit dem steigendem Alter des Netzwerkes öffnet sich zunehmend die Schere zwischen dem realen System und dessen Planungsunterlagen. In diesem Fall sind die für eine Sicherheitsanalyse nötigen Informationen nicht mehr vorhanden, was Angriffsvektoren auf den Feldbus öffnet.

Für dessen Analyse müssen topologische Informationen über ein GAS vorliegen, welche in der Praxis in bestimmten Netzwerkmodellen gespeichert werden. Durch die Nutzung von Filtern können dann Segmente des Netzwerks betrachtet und sicherheitskritische Knoten in diesem ermittelt werden. Diese Informationen können dann zur Prävention von Angriffen genutzt werden.

Somit stellen alternde Planungsunterlagen nicht nur in der Wartung und Erweiterung eines Systems, sondern auch in dessen Sicherheit ein Problem dar. Hier wäre es wünschenswert, diese Dokumentation mit einem automatischen Verfahren zu validieren.

1.3 Folgen der Dokumentenerosion

Es gehört zur Verantwortung eines Netzwerkadministrators, die Topologie seines Netzwerkes zu kennen. Ist dies nicht gegeben, entstehen offensichtliche Nachteile: Ist

der physische Aufbau unbekannt, so werden Wartungsarbeiten und die Fehlersuche erschwert. Sollte der Kabelverlauf in den Wänden nicht bekannt sein, müssen diese im schlimmsten Fall unnötig geöffnet werden. Die so entstehenden Unkosten untergraben alle Vorteile, welche durch flexible Bussysteme überhaupt erst entstehen sollten. Die angedachte, einfache Wartung des GAS entartet zum praktischen Herumprobieren.

Auch die Sicherheit eines Feldbusses kann durch fehlende Topologiekenntnisse in Frage gestellt werden. In Feldbussen ist es oft wichtig, dass Netzschnittstellen ausreichend geschützt sind. In Protokollen wie KNX basiert die Sicherheit des Systems einzig und allein auf einer Separation der Netzwerke – innerhalb des Feldbusses existieren keine Sicherheitsvorkehrungen und auch Passwörter, welche zur Gerätekonfiguration nötig sind, werden im Klartext übertragen [JKK14; GKK16]. Ein Angreifer mit Zugriff auf das Netzwerk erlangt rasch viel Kontrolle und kann ungehindert Telegramme mitschneiden, manipulieren oder selbst erzeugen um sich als Kommunikationspartner auszugeben. Sicherheit wird in KNX dadurch gewährleistet, dass physischer Zugang zum Feldbus unterbunden wird. Schnittstellen müssen unbedingt in zugangsbeschränkten Bereichen des Gebäudes eingerichtet werden [Knxc]. Falls die Topologie des Netzwerkes unbekannt ist, können solche Zugriffspunkte undokumentiert sein, was die Sicherheit des gesamten Systems erheblich in Gefahr bringt. Erweiterungen wie *KNX Data Secure* und *KNX IP Secure* existieren um KNX-GAS zu schützen [Knxc], es ist jedoch zu erwarten, dass diese nicht in jeder Installation genutzt werden.

1.4 Weiterer Aufbau der Arbeit

Im Verlauf dieser Arbeit soll ein Modell erarbeitet werden, in welchem sich verschiedenste GAS darstellen lassen. Dieses soll in weiterführenden Arbeiten zur Sicherheitsanalyse von Feldbussen genutzt werden. Es folgt eine kurze Erläuterung des Aufbaus der vorliegenden Arbeit.

Zunächst werden verschiedene GAS-Standards und die von ihnen genutzten Medien betrachtet. Dies ist nötig, um Modell zur Darstellung von Feldbussen zu finden, welches kompatibel zu möglichst vielen dieser Systeme ist. Auch werden vergleichbare

Bemühungen zur Topologieerkennung im Internet aufgezeigt. Dies dient hauptsächlich zum Belegen der Relevanz des Themas, aber auch um effiziente Algorithmen in diesem Gebiet zu betrachten.

Auch wird eine Übersicht über bereits existierende Modellierungssprachen und -Frameworks gegeben. Diese werden im Hinblick auf ihre Verwendung zur Darstellung von Feldbussen untersucht. Das letztendlich genutzte Format, *AutomationML*, findet dabei besondere Ansprache. In diesem können die *logische Topologie*, die *physische Topologie* und die *funktionale Topologie* dargestellt werden.

Aufgrund des zeitlichen Rahmens der vorliegenden Arbeit wird lediglich die Erzeugung der *logischen* Topologie umgesetzt. Für diese werden nach AutomationML eine Vielzahl an Netzwerk-Elementen definiert. Auch werden Anforderungen und Qualitätskriterien für das Modell aufgezeigt. Vor- und Nachteile mehrerer möglicher Datenquellen werden detailliert behandelt.

Die hier konzipierte Topologieerkennung wird am Beispiel des GAS-Standards *KNX* vorgenommen. Es wird gezeigt, wie dort genutzte Nachrichtenformate zur Informationsgewinnung genutzt werden können. Zusätzlich wird auf die Unterschiede zwischen passivem und aktivem Scannen eingegangen.

Auch wird ein Prototyp für die logische Topologieerkennung implementiert. Dessen Aufbau und Funktionsweise wird in dieser Arbeit aufgezeigt. Dieser soll zusätzlich die Möglichkeit besitzen, gültige AML-Dateien zu erzeugen.

Kapitel 2

Systeme und Topologien

In diesem Kapitel

In diesem Kapitel werden verschiedene Grundlagen für die vorliegende Arbeit vorgestellt. Der erste Abschnitt wird sich mit der formalen Erklärung und Vorteilen der Nutzung von GAS beschäftigen. Im Anschluss werden verschiedene Kommunikationsmedien betrachtet, welche für die Verwendung von GAS in Betracht gezogen werden können. Eine Auswahl von am Markt verfügbaren GAS wird darauffolgend vorgestellt und deren topologische Eigenschaften erläutert. Zuletzt wird auf vergleichbare Bemühungen der Topologieerkennung im Internet aufmerksam gemacht.

2.1 Gebäudeautomatisierung

Mehr und mehr kommen GAS in gewerblichen, aber auch privaten Einrichtungen, zum Einsatz. Ein GAS ist dabei ein Netzwerk aus Sensoren und Aktoren, welches

verschiedenste Automatisierungsfunktionen in Haushalten übernimmt. Es stehen nicht nur Gründe wie Komfort im Vordergrund: Auch Sicherheit und besonders Energieeinsparungen sind wichtige Aspekte. Es wurde gezeigt, dass in der EU Gebäude für rund 40% des Energieverbrauchs verantwortlich sind und durch die Nutzung von GAS in gewerblichen Gebäuden Energieeinsparungen von bis zu 30% möglich sind [Cou10; Kat19]. Des Weiteren können durch automatisierte Lüftungs- und Heizkontrolle die Produktivität der Arbeitskräfte gesteigert sowie andere Arbeiten effektiv unterstützt werden [MHH09].

Im Grunde steht der hohen Funktionalität eines GAS nur der hohe Einstiegspreis entgegen. Über die gesamte zu erwartende Laufzeit eines neu errichteten Gebäudes lässt sich jedoch trotzdem sagen, dass die Nutzung dieser Systeme nicht nur ökologisch, sondern auch ökonomisch sinnvoll ist [Kas+05].

2.1.1 Feldbusse

Als Grundlage für ein GAS können in vielen Systemen verschiedenste Medien (siehe Abschnitt 2.2) genutzt werden. Diese Netzwerke und die auf ihnen gesprochenen Protokolle werden als Feldbusse bezeichnet. In der Prozess- und Automatisierungstechnik werden sie bereits seit geraumer Zeit genutzt, um Sensoren, Aktoren und vielen weiteren für die Ausführung des Systems benötigte Geräte miteinander zu verbinden.

Feldbusse spielen für diese Arbeit eine zentrale Rolle. Kapitel 4 widmet sich der Konzipierung eines Modells, durch welches sich zahlreiche Eigenschaften dieser Netzwerke darstellen lassen und das insbesondere der Sicherheitsanalyse eines GAS dienen soll.

2.1.2 Automatisierungspyramide

GAS können in einem drei-Schichten-Modell dargestellt werden, welches auch als *Automatisierungspyramide* (Abb. 2.1) bezeichnet wird. Jede dieser Schichten – die Feldbusebene, die Automationsebene und die Leitebene – realisiert dabei einen Teilaspekt

einer GAS-Lösung. In der Praxis werden jedoch selten alle drei Ebenen von einem GAS allein abgedeckt, weshalb die Spezifikation der Schnittstellen zwischen diesen enorm wichtig ist [MHH16].

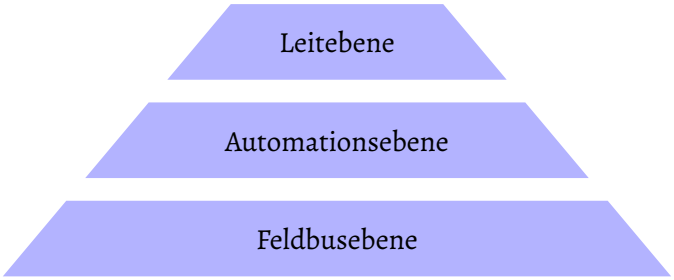


Abbildung 2.1: Die Automatisierungspyramide nach [MHH16; Asc14]

Die *Feldbusebene* stellt die Grundlage für jede Art von GAS dar. Sie bezeichnet den Bus selbst und damit die eigentliche Verkabelung, das verwendete Medium und die Vernetzung der im System verfügbaren Geräte. In dieser findet die Kommunikation unter den Busteilnehmern statt. Aus diesem Grund lassen sich viele Funktionen der Gebäudeautomatisierung bereits nur auf Grundlage dieser Ebene realisieren: Die Steuerung von Beleuchtung, Temperatur und Belüftung sind Beispiele dafür [MHH16].

Der tatsächlich automatische Teil der GAS wird in der *Automationsebene* implementiert. Hier werden Regeln für das Verhalten des Systems definiert. So kann hier die Logik für die Zeit-, die Anwesenheits- und die Zustandsteuerung festgelegt werden. Auf diese Weise lassen sich Prozesse für bestimmte Zustände festlegen. Dazu gehören beispielsweise die Beleuchtung in der Dunkelheit oder bei Anwesenheit, die Regelung der Lufttemperatur und -Feuchtigkeit und das Smart Metering [MHH16]

Zu der *Leitebene* gehören Funktionen, welche dem Management des GAS zugeordnet werden. Somit werden von dieser I/O-Funktionalitäten wie Bedienung, Visualisierung und das Anzeigen von Störmeldungen realisiert [MHH16].

2.2 Medien

Ein GAS kann auf vielen verschiedenen Medien realisiert werden. Diese haben aufgrund ihrer jeweiligen Eigenschaften auch einen großen Einfluss auf die Topologie des

Feldbusses. Geräte in kabellosen Systemen kommunizieren im Vergleich zu denen in kabelgebundenen Systemen gänzlich anders. Während Erstere meist keiner starren Struktur folgen, ist die Position jedes Gerätes in den Zweiteren festgelegt und kann spontan nicht verändert werden. Auch unter kabelgebundenen Systemen existieren Unterschiede. So kann dieses eine dedizierte Installation für ein GAS sein, oder auf die unterliegende Elektroinstallation zurückgreifen.

Welche Medien bei der Verwendung von GAS zum Einsatz kommen, hängt oft davon ab, in welchem Zustand sich der Bau befindet: Handelt es sich um einen Neubau oder eine Renovierung, so kann für einen verhältnismäßig kleinen Aufpreis ein drahtgebundenes Netzwerk installiert werden. Soll ein bestehendes Gebäude ohne große Unkosten um ein GAS erweitert werden, so bieten sich kabellose Lösungen oder die Nutzung vorhandener Systeme wie Ethernet oder das Stromnetz an. Funkbussysteme nutzen spezielle Protokolle, um Geräte kabellos und somit kostengünstig zu verbinden [Asc14].

Die Topologie eines GAS ist stark abhängig von dem gewählten Medium und ein Netzwerkmodel muss diese weitgehend abdecken. Aus diesem Grund werden folgend die Eigenschaften möglicher Medien zusammengefasst.

2.2.1 Kabelgebundene Systeme

In dedizierten (kabelgebundenen) Systemen werden Sensoren und Aktoren unabhängig von bestehenden Netzwerken miteinander verbunden. Da viele Hersteller an vielen, sehr unterschiedlichen Systemen arbeiten, unterscheiden sich die angebotenen Lösungen sehr. In der Regel finden aber in solchen Systemen vieradrige Kabel ihren Einsatz, wobei auf zwei Adern Daten übertragen werden und die restlichen, abhängig vom System, für die Stromversorgung zuständig sind [Asc14]. In KNX beispielsweise werden Busteilnehmer per Steckklemme zu dem Netz hinzugefügt und entfernt, ohne dass die Busleitung unterbrochen werden muss. Kollisionen werden durch das Verfahren *Carrier Sense Multiple Access/Collision Avoidance* (CSMA/CA) behandelt, bei dem versucht wird, Paketkollisionen durch vorheriges Abhören der Leitung zu vermeiden [Knxb].

Dedizierte Systeme verfügen über den Vorteil, dass sie ihr Medium nicht mit anderen Anwendungen teilen müssen. Das kann hilfreich sein, da so die vollständige Bandbreite des Kanals für das GAS genutzt werden kann und Wartungen parallel stattfinden können. In Hinsicht auf bestimmte Protokolle kommt jedoch ein weiterer Vorteil auf: Die Separation der Netzwerke. Protokolle wie KNX bauen ihre Sicherheit auf dem Konzept auf, dass ein Angreifer niemals Zugriff zu dem Netzwerk erlangen darf. Das ist nötig da die Kommunikation unverschlüsselt und unautorisiert geschieht. Dies wird erst durch ein dediziertes System ermöglicht [Knxc].

2.2.2 Kabellose Systeme

Kabellose GAS bieten vor allem einen enormen Kosten- und Installationsvorteil. Diese werden entweder durch eine feste Stromversorgung oder mit Batterien in Reichweite ihrer Infrastruktur betrieben und zeichnen sich durch ihre hohe Flexibilität aus. Für die Übertragung wird oft das ISM-Band genutzt, welches die Frequenzen 433 MHz und 868 MHz umfasst [Asc14]. Der Nachteil dieser Systeme ist, dass sich bei vielen Netzteilnehmern die Wahrscheinlichkeit von Kollisionen während der Kommunikation stark erhöht. Dies hat die Mehrfachsendung von Nachrichten und damit eine höhere Netzlast mit steigenden Latenzen zur Folge [Knxb].

2.2.3 Powerline-basierte Systeme

Eine weitere kostengünstige Alternative zu kabellosen Systemen sind Powerline-basierte Verfahren. Hier wird auf die vorhandene 230-V-Leitung zurückgegriffen, um Geräte über die Stromversorgung zu vernetzen. Als namentliche Vertreter sind GAS wie X10 und digitalStrom als reine Powerline-Systeme zu nennen. Des Weiteren verfügen Systeme wie KNX über entsprechende Erweiterungen [Asc14].

Bei unverschlüsselten Protokollen wie KNX ist jedoch auf die Sicherheit des Netzwerkes zu achten: Da Stromnetze typischerweise nicht oder nur ungenügend voneinander abgeschirmt sind, ist es möglich, dass Signale noch außerhalb des eigenen Haushalts erkannt werden [PDR08]. Wird das Signal jedoch an der Stromquelle genügend abgeschirmt, so können Powerline-Systeme nicht verlässlich zwischen Gebäuden genutzt werden [SHS17].

2.2.4 IP-basierte Systeme

Neben der Elektroinstallation ist auch Ethernet in fast allen Haushalten weit verbreitet. Obwohl keine GAS rein auf Basis von LAN/WLAN existieren, können Systeme wie KNX, LCN, LON sich das verfügbare Netzwerk mit speziellen Komponenten zunutze machen. Weitere Vorteile sind, dass jegliche vorhandene Hardware meist bereits über Ethernet kommunizieren kann. Bezüglich des Stromverbrauchs sind WLAN-Systeme aufgrund des höheren Netzverkehrs anspruchsvoller als rein kabellose GAS [Asc14].

2.3 Standards

Auf den Grundlagen der in 2.2 gezeigten Medien entwickeln viele Hersteller eine große Zahl an konkreten GAS-Lösungen. Diese speziellen Umsetzungen unterscheiden sich weiterhin topologisch zueinander. Für die Entwicklung eines Models, welches verschiedenste GAS umfasst, ist es aus diesem Grund nötig, eine Übersicht über die am Markt verfügbaren Systeme zu erarbeiten.

Der Bereich der GAS umfasst eine große Anzahl an Standards und Protokollen. Über 80 verschiedene Systeme werden auf dem Markt mittlerweile angeboten – alle mit eigenen Vor- und Nachteilen sowie vorgesehenen Einsatzgebieten. Für Bauherren und Installateure ist diese Zahl unüberschaubar und Interoperabilität ist längst nicht zwischen allen gegeben [Asc14]. Im Folgenden wird eine Auswahl an relevanten GAS vorgestellt.

2.3.1 KNX

KNX ist ein Standard [Knxa], welcher aus der Verschmelzung dreier führender GAS hervorgegangen ist: EIB, EHS und BatiBus [SHS17]. KNX ist nicht an ein einziges Medium gebunden, sondern bietet die Möglichkeit über eine eigene Twisted-Pair-Leitung (TP), Funk (RF), Ethernet (IP) oder per Powerline (PL) zu kommunizieren. Oft wird auf die Eigenschaften des Vorgängers EIB zurückgegriffen. So sind die Spezifikationen für

TP und PL Erbschaften des älteren Standards, IP und PL jedoch Erweiterungen [Knxd]. In gewissen Maßen ist auch eine Abwärtskompatibilität zu älteren EIB-Geräte gegeben [Asc14].

Ein KNX-Netzwerk wird hierarchisch in Arten von Segmente eingeteilt. Die kleinste vorkommende ist die *Linie*. In einer solchen können bis zu 64 Teilnehmer miteinander verbunden werden. Mit Hilfe von *Linienverstärkern* können auch bis zu 256 Geräte pro Linie genutzt werden. Eine zweite Möglichkeit, die maximale Anzahl der Teilnehmer zu erhöhen, ergibt sich durch das Hinzufügen weiterer Linien. Bis zu 16 Linien können dabei mit Hilfe von *Linienkopplern* miteinander zu einem *Bereich* verbunden werden. Dies geschieht auf der sogenannten *Hauptlinie*, welche auch eine vollwertige, eigenständige Linie darstellt. Des Weiteren können auch bis zu 15 Bereiche durch *Bereichskoppler* zu einem Gesamtsystem verbunden werden. Bereiche wiederum werden auf einer Linie miteinander verbunden, welche als *Backbone* bezeichnet wird [Knxd; Knxb]. Im maximalen Aufbau ist es möglich bis zu 58.623 Geräte in einem KNX-Netzwerk zu verbinden.

Anhand deren Position im Netzwerk wird jedem Gerät eine eindeutige, physische Adresse zugeordnet. Befindet sich der Teilnehmer auf der vierten Linie des neunten Bereichs, so wird ihm eine Adresse zwischen 9.4.1 und 9.4.255 zugewiesen. Die erste Adresse der Linie, 9.4.0, ist für den Linienkoppler reserviert. Auch Bereichskopplern wird jeweils die erste Adresse ihres Bereichs zugewiesen – in diesem Beispiel also 9.0.0 – und die Backbone-Linie wird immer mit 0.0.0 adressiert. Ringtopologien sind in KNX grundsätzlich nicht erlaubt [Knxd].

Zur Konfiguration eines KNX-Netzwerkes wurde von der KNX-Association die Software ETS (Engineering Tool Software, Beispiel in Abb. 2.2) für das Betriebssystem Windows veröffentlicht. Per USB- oder IP-Schnittstelle kann diese auf den KNX-Bus zugreifen und Anwendungsprogramme auf die Geräte aufspielen, welche online durch ihre Hersteller angeboten werden [Knxb]. Des Weiteren ist es mit der ETS-Software möglich, komplette KNX-Netzwerke als Projekte aus dem System zu extrahieren, um dieses beispielsweise offline zu designen und später auf das reale Netzwerk zu überführen [SHS17].

In der KNX-Kommunikation wird das Nachrichtenformat *Common External Message Interfaces (cEMI)* genutzt. Aus diesem lassen sich viele Informationen über die beteiligten

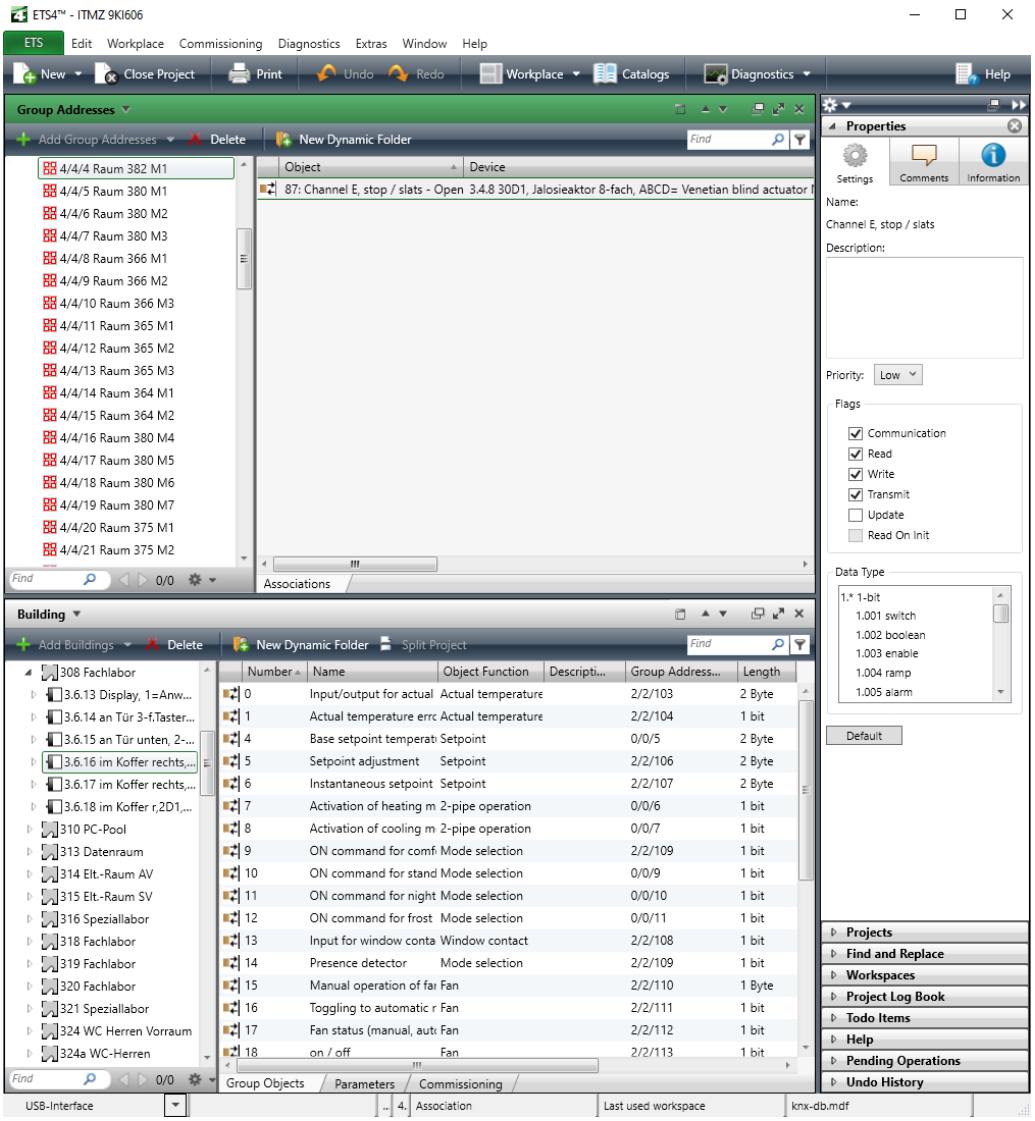


Abbildung 2.2: ETS-Projekt im Editor

Geräte in Erfahrung bringen. Das Format cEMI soll jedoch erst in Abschnitt 5.1 näher erläutert werden, sobald dessen detaillierte Beschreibung für das konkrete Konzept erforderlich ist.

2.3.2 LonWorks

In Europa ist LON (Local Operating Network) Hauptkonkurrent von KNX. Dieses GAS wurde von Echolon entwickelt und kann als zentralisiertes oder dezentralisiertes

System eingerichtet werden [MHH09]. Das dem System zu Grunde liegende LonTalk-Protokoll ist offen und ermöglicht so verschiedensten Herstellern die Entwicklung neuer Geräte für das GAS. Auch ist LON auf einer Vielzahl von Medien nutzbar. Dazu gehören TP, RF, PL, COAX-Kabel, Glasfaser und auch Infrarot. Durch die Vielzahl an unterstützten Medien und das offene Protokoll ist LON besonders flexibel einsetzbar [Asc14].

Die Topologie eines LON-Netzwerkes ist frei wählbar: Es kann als Stern-, Ring-, Baum- oder Linienstruktur implementiert werden. Identifiziert werden Geräte durch eine eindeutige Identifikationsnummer in 48 Bit, welche bei der Herstellung des frei programmierbaren Neuron-Chips in jeder Komponente festgelegt wird. Das Netzwerk kann auch in zwei Domains unterteilt werden – jede mit bis zu 255 Linien à 127 Knoten. So kann das Gesamtsystem auf eine Größe von bis zu 32.385 Knoten heranwachsen [Asc14].

Im Zentrum des LonWork-Systems sitzt der Neuron-Chip, welcher in jedem Gerät installiert ist. Auf diesem finden sämtliche Berechnungen und die Kommunikation mit anderen Geräten statt. Obwohl das zur Ausführung benötigte Programm oft bereits auf den Chips vorinstalliert ist, lassen sich diese auch über das LonTalk-Protokoll programmieren [MHH09]. Dank der offenen Architektur des LonWork-Systems existieren für LON, entgegen beispielsweise KNX mit ETS, eine Vielzahl an Programmiertools für Endanwender [Asc14].

2.3.3 ZigBee

ZigBee ist, entgegen der bisher vorgestellten GAS, ein kabelloser Standard der ZigBee Alliance und ordnet sich selbst als kostengünstig und stromsparend im Markt ein. Um die batteriebetriebenen Geräte effizient betreiben zu können, wird eine maximale Datenrate von 250 kb/s genutzt, welche jedoch in den USA und Europa auf jeweils 40 kb/s und 20 kb/s gedrosselt wird. Im Standard werden drei verschiedene Bänder für die drahtlose Kommunikation definiert: 2.4 GHz (global), 915 MHz (USA und Australien) und 868 MHz (Europa) [Saj+06]. Seit der 2012 erschienen Revision des ZigBee-Standards ist es auch möglich ein Netzwerk per ZigBee IP an das Internet anzubinden [Fra+13].

Verfügbare Geräte unterscheiden sich in zwei Aspekten: Zum einen wird zwischen zwei verschiedenen physikalischen Rollen – den sogenannten *Full Function Devices* und *Reduced Function Devices* – unterschieden: Bei diesen handelt es sich jeweils um Geräte, welche entweder mit relativ hohen Systemresourcen ausgestattet wurden, oder batteriesparende Geräte, welche weniger Rechenkapazitäten besitzen und nicht selten den Großteil der Zeit im Standby-Modus verbringen. Zum anderen existieren logische Rollen, zu denen Zigbee-*Koordinatoren*, *-Router* und *-Endgeräte* gehören. Koordinatoren und Router sind aufgrund ihrer hohen Leistungsanforderungen immer Full Function Devices. Pro Netzwerk existiert genau ein Koordinator, welcher auch für dessen Initialisierung zuständig ist. Andere Knoten können sich beliebig zu diesem verbinden. Router sind lediglich für die Informationsweiterleitung sowie zur Vergabe der 16 Bit großen Geräteadresse zuständig. Auch zu ihnen können sich Knoten beliebig verbinden. Endgeräte können entweder Full Function Devices oder Reduced Function Devices sein. Sie können nur mit Routern oder dem Koordinator kommunizieren [Cra04; Saj+06]. Durch die theoretisch sehr hohe Vernetzung der Geräte entsteht in der Regel eine stark vermaschte Topologie.

2.3.4 Z-Wave

Ein weiteres kabelloses GAS ist das von Zensys entwickelte Z-Wave. Es verfolgt ähnliche Ziele wie ZigBee: Eine einfache Einrichtung und kostengünstige Vernetzung dank kabelloser Technologien sowie stromsparende Endgeräte. Durch entsprechende Gateways wird es ermöglicht, das GAS an das Internet anzuschließen und so zu kontrollieren [YMK16]. Zur Übertragung wird in Europa und Amerika jeweils auf den Frequenzen 868.42 MHz und 908.42 MHz gesendet. Die Datenraten orientieren sich hierbei an 40 kb/s [FG13].

Z-Wave bietet zwei verschiedene Arten von Geräten: *Controller* und *Slaves*. Die letzteren müssen keine Routing- und Topologie-Informationen besitzen und führen lediglich Befehle aus, welche sie von ihren Controllern gesendet bekommen. Controller existieren als *primäre Controller* und *sekundäre Controller*. Primäre Controller nehmen dabei eine ähnliche Funktion wie die Koordinatoren in einem ZigBee-Netzwerk ein: Sie speichern Routing-Informationen, wissen, welche Geräte miteinander kommunizieren

können und administrieren die 8 Bit Geräte-IDs sowie die 32 Bit Netzwerk-ID, welche genutzt werden, um Z-Wave-Netzwerke voneinander zu trennen [FG13; YMK16]. Mit einem reduzierten Funktionsumfang verfügt der sekundäre Controller lediglich über eine eigene Routing-Tabelle, welche auch vom primären Controller abgefragt werden kann [YMK16]. Ein weiterer Unterschied zu ZigBee besteht darin, dass nicht nur die Controller Telegramme weiterleiten dürfen. In Z-Wave kann jeder nicht-batteriebetriebene Knoten als Repeater arbeiten [FG13]. Hier entsteht – wie bei ZigBee – eine stark vermaschte Netzwerktoplogie.

2.3.5 Gateways

Im Bereich der GAS wird bei Geräten, welche Verbindungen zwischen verschiedenen Protokollen und Medien herstellen, von Gateways gesprochen. Wie bereits in Abschnitt 2.2 und 2.3 gezeigt, existiert eine Vielzahl an verschiedenen GAS-Standards sowie Kommunikationsmedien. Implementiert ein System keinen offenen Kommunikationsstandard, so wird ein spezialisiertes Gerät benötigt, welches die Verbindung zwischen verschiedenen Netzwerken und Medien herstellt (siehe Abbildung 2.3). Diese Geräte werden Gateways genannt [Asc14].

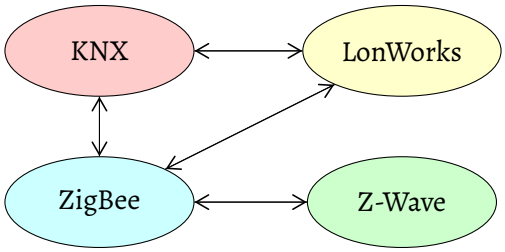


Abbildung 2.3: Verbindungen zwischen GAS durch Gateways [Int; LTD; WS08; Kel]

Aus der Betrachtung der Systemsicherheit sind Gateways als besonders kritische Geräte einzuordnen. Angriffe auf diese Netzknoten haben zur Folge, dass möglicherweise alle verbundenen Teilsysteme kompromittiert werden [Gra14].

2.4 Topologieerkennung im Internet

Dieser Abschnitt zeigt frühe Bemühungen zur Topologieerkennung, welche im Bereich des Internets durchgeführt wurden, da dieses Aufgrund der ähnlichen Technologie zu

Feldbussen vergleichbar ist. Betrachtet werden hier die möglichen Gründe für die Ermittlung von Netzwerkinformationen, jedoch auch welche Strategien genutzt wurden, um topologische Eigenschaften aus solchen zu extrahieren und welche Probleme damit verbunden sind. Auch wird gezeigt, wie Algorithmen in diesem Bereich verbessert wurden, um eine höhere Effizienz und eine kleinere Netzlast zu erzeugen. Besonders die Verkleinerung des Fußabdrucks der Topologieerkennung ist für diese Arbeit wichtig, um zu zeigen, wie ein Angreifer seinen Zugriff auf ein Netzwerk verschleiern kann. Die verwendeten Modelle selbst finden in diesem Abschnitt jedoch keine Ansprache, da diese in der verwendeten Literatur kaum Erwähnung fanden.

Bereits in den frühen 2000er Jahren war die Topologieerkennung von Netzwerken ein relevantes Thema. Zu diesem Zeitpunkt bildeten sich mehrere Initiativen, um entsprechende Informationen über das noch wachsende Internet zu ermitteln. Die Gründe dafür waren vielfältig, jedoch meist im informellen Bereich angesetzt. In [Huf+02] ist von Interesse – ähnlich wie für die Zielstellung dieser Arbeit – ob die erwartete Topologie einer generierten Topologie entspricht. Des Weiteren waren die hier angestellten Messungen relevant für das Load Balancing des frühen Internets. Auch die Pfadlängen der Anfragen wurden im Rahmen dieser Arbeit untersucht. [MBB00] beschreibt eines der ersten Verfahren zur Erkennung der Topologie des Internets. Die hier ermittelten Daten wurden hauptsächlich von der High-Performance-Connection-Community genutzt, um ein Verständnis der verfügbaren Netzwerke zu gewinnen. Es ist aber auch im Interesse der Internet Service Provider (ISPs), Forschung in diesem Bereich anzustellen. So haben bereits 2001 viele IPSs an ähnlichen Projekten teilgenommen [Geo+01]. Verschiedene Initiativen werden hier kurz vorgestellt.

2.4.1 Projekte des NLANR

Das *National Laboratory for Network Research* (NLANR) hat in den späten 90er bis frühen 2000er Jahren an verschiedenen Projekten gearbeitet, welche eine Vielzahl an Informationen über das damalige Internet zusammengetragen haben. Im Fokus lagen dabei jeweils die Interessen der High-Performance-Connection-Community, welche an strukturellen Informationen interessiert war [MBB00].

Eines dieser Projekte war die *Network Analysis Infrastructure* (NAI). Erforderliche

Daten wurden entweder passiv oder aktiv gesammelt. Passive Verfahren sahen vor, an strategischen Punkten des Netzwerkes den Verkehr zu analysieren. Am besten bieten sich hierfür Verbindungen zwischen verschiedenen Netzsegmenten an. Aktive Messungen beinhalten das Senden eigener Proben und die anschließende Auswertung des sich daraus ergebenden Verhaltens. Typischerweise gehören hierzu Informationen wie Latenz und Packet Loss. Weiterhin bot das Projekt jedoch auch Möglichkeiten Kontrollinformationen auszuwerten. Hier wurden Routing-Tabellen und Netzwerk-Management-Informationen gesammelt [MBB00].

Cichlid war ein weiteres Projekt der NLANR, welches der Visualisierung der gewonnenen Informationen diente. Der Client wurde als ein oder mehrere Datenservern implementiert und leitet deren Informationen zur visuellen Darstellung weiter [MBB00].

OCXmon war lediglich für passives Monitoring gedacht. Im Jahre 2000 existierten 11 verschiedene OCXmon-Monitore zum Sammeln, wobei bis zu 25 weitere bereits geplant waren. Die gesammelten Informationen beinhalteten IP-Quell- und Ziel-Adresse sowie das verwendete Protokoll und Portnummern. Die generierten Informationen wurden hauptsächlich zur Analyse des Datenflusses und Generierung von Statistiken eingesetzt. Jeder der Monitore generierte täglich 10 MB bis 1 GB Daten. Die damals zur Aufzeichnung verwendeten Geräte konnten so Datenverkehrsinformationen in einem zeitlichen Rahmen von etwa einem Monat speichern [MBB00].

Das letzte hier vorgestellte Projekt ist das *Active Measurement Projekt* (AMP), welches, wie der Name bereits verrät, für aktive Messungen vorgesehen war. Hier wurde im Jahre 2000 eine Gesamtanzahl von 100 Monitoren verwendet, wobei ein monatlicher Zuwachs von etwa 10 Geräten zu verzeichnen war. Die gesammelten Informationen beinhalteten die Round-Trip-Time, die Paketverlustrate und den Paketdurchsatz. Die vorhandenen Monitore waren konfiguriert, jede Minute ICMP-Pakete untereinander zu versenden und auch deren Antwortzeiten zu speichern. Des Weiteren wurden im Abstand von 10 Minuten traceroutes zwischen den Monitoren und zu verschiedenen Webservern ausgeführt [MBB00].

2.4.2 Test Traffic Measurements

Das Projekt *Test Traffic Measurements* (vorher: *Test Traffic Project*), wurde durch die Unterstützung der ISPs von der RIPE NCC ins Leben gerufen, da diese ein geschäftliches Interesse an der Performance-Auswertung ihrer Netzwerke besaßen. Hierfür wurden in den Provider-Netzwerken in der Nähe der Border-Router (die Router, welche verschiedene Netzsegmente miteinander verbinden) Geräte zur Messung installiert. Diese wurden als Black-Boxes konzipiert, um die Manipulation durch Netzbetreiber zu vermeiden [Geo+01].

Auch hier wurden Performance-Informationen in verschiedenen Metriken durch die installierten Geräte gesammelt. Von besonderem Interesse war unter anderem der Aspekt der One-Way-Messungen: Hier wurden Fälle betrachtet, nach denen der Netzverkehr asymmetrisch verläuft, sprich Pakete auf Hin- und Rückweg von Quelle zu Ziel verschieden geroutet werden. Zum Schutz der Privatsphäre der Kunden wurde eigener Netzverkehr erzeugt [Geo+01].

2.4.3 skitter

Auch das *Center for Applied Internet Data Analysis* (CAIDA) begann Anfang des Jahrtausends mit Messungen der Struktur des Internets. Hier wurde ein Tool namens *skitter* entwickelt, welches ähnlich wie vorhergehende Ansätze an verschiedenen strategischen Positionen als Monitor implementiert wurde. Ziele des Projektes waren der Vergleich ermittelter Topologien zu den dokumentierten Netzwerken und die Analyse des Netzverkehrs zum Zwecke des Load Balancing [Huf+02].

Insgesamt existierten im Jahre 2002 18 Skitter-Monitore, welchen gruppenweise verschiedene Aufgaben zugeteilt wurden: Sieben Monitore wurden zur Untersuchung von DNS-Clients genutzt. Hierzu wurden DNS-Clients beobachtet, welche Anfragen zu deren Root-Servern stellten und so eine Liste mit 58.000 Adressen zusammengestellt. Fünf Monitore waren für verschiedenste IP-Adressen zuständig. Hierfür wurde aus verschiedenen Quellen eine Liste von 661.000 Einträgen angefertigt, wobei für jedes /24-Segment nur ein Repräsentant gewählt wurde. Ein weiterer Monitor nutzte eine verkleinerte Version dieser Liste, um höhere Abfrageraten zu ermöglichen. Die letzten

fünf Monitore waren für verschiedenste Webserver zuständig. Hier wurde eine Liste von 15.000 Einträgen erstellt [Huf+02]. Die Zahl der Monitore erhöhte sich bis ins Jahr 2005 auf 24 [DFC05].

Auch skitter nutzte ein einfaches Verfahren, gestützt auf ICMP-Traceroutes und ansteigender Time-To-Live (TTL). Anfragen an entsprechende Ziele fanden in sogenannten Cycles statt – die Zeit welche ein skitter-Monitor benötigt, um alle Adressen seiner Liste abzufragen. Einfluss auf die Dauer eines solchen Cycles hatten verschiedene Faktoren: Die Länge des Pfades bis zum angefragten Ziel spielt eine erhebliche Rolle, jedoch auch die Anzahl der dazwischenliegenden Hops, welche nicht auf Anfragen antworten. Weiterhin ist die Netzgeschwindigkeit ausschlaggebend, aber auch die Größte der verwendeten Liste von Hosts [Huf+02].

Das Projekt skitter wurde 2008 beendet. In dessen Folge wurde die *Archipelago (Ark) Measurement Infrastructure* ins Leben gerufen [Cai].

2.4.4 Betrachtungen der Effizienz

In [DFC05] wurden verschiedene Verfahren der Netzwerkanalyse betrachtet und auf deren Effizienz untersucht. Dabei stellte es sich heraus, dass viele der oben genannten Projekte unter Problemen in Bezug auf deren Skalierbarkeit litten. Probleme ergaben sich dabei nicht clientseitig – die Projekte SETI@home, NETI@home und DIMES nutzen sogar minimal intrusive Systeme, welche teils in Screen Savern implementiert wurden – sondern serverseitig, da nicht zwischen Messungen von vielen Monitoren und böswilligen DDoS-Attacken unterschieden werden konnte. Verschiedene Projekte versuchten die auffallenden Probleme unterschiedlich zu lösen: skitter besaß beispielsweise nur eine begrenzte Anzahl an Monitoren, währen DIMES nur eine geringe Anfragerate zugelassen hat.

Noch dazu zeigte sich, dass viele der Verfahren hochgradig ineffizient arbeiteten. Für eine Analyse der Ergebnisse von skitter wurden von dessen insgesamt 971.080 Zielen eine Auswahl von 50.000 untersucht. Hierbei stellte sich heraus, dass durchschnittlich lediglich 10,4 % der Anfragen neue, unbekannte Interfaces entdeckten. Der Rest der Anfragen teilte sich auf 2,0 % fehlerhafte Adressen (oder solche ohne Antwort)

und ganze 87,6 % redundante Anfragen auf. Der überwiegende Anteil der Messungen entdeckte also keine neuen Informationen [DFC05]. Um diesem Umstand Abhilfe zu verschaffen, wurden mehrere Ansätze vorgeschlagen.

Doubletree

Doubletree ist ein Algorithmus, welcher redundante Abfragen in *traceroute*-ähnlichen Messungen verringert. Alle verfolgten Pfade können hier als einer von zwei Bäumen betrachtet werden: Ein Baum hat seine Wurzel im Ziel der Anfrage und seine Blätter in der Menge aller Clients, welche die Wege zur Wurzel verfolgen. Ähnlich kann die Wurzel des Baumes im Client liegen und die Blätter aus all den Hosts bestehen, welche als Ziele in der jeweiligen Liste des Clients angegeben sind [DFC05]. Mit diesem Verfahren lässt sich der Fußabdruck eines aktiven Scans verkleinern, was auch wichtig für die unerkannte Informationsgewinnung in Feldbussen ist.

Die Anfragen können vor- und rückwärts vom Client zum Ziel geschehen. Im sogenannten *backward-probing* wird mit einer sinkenden TTL die Route zum Host von diesem aus zurückverfolgt. Die besuchten Interfaces werden in einem *local stop set* gespeichert. Wird in einer Anfrage ein bekanntes Interface entdeckt, so muss der restliche Weg zum Client nicht weiter verfolgt werden [DFC05].

Zu sehen ist dieses Verfahren in Abbildung 2.4. Hier wird gezeigt, wie bekannte Wege (rot markiert) von anderen Traces nicht weiterverfolgt werden.

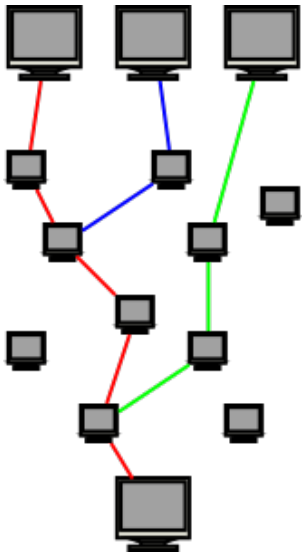


Abbildung 2.4: Ablauf des Probing

Ähnlich geschieht das *forward-probing*. Hier wird zwischen allen Monitoren das sogenannte *global stop set* erarbeitet, in welches die Clients Paare von Interface und Zieladresse eintragen. Treffen sich zwei Pfade auf dem Weg zur gleichen Zieladresse, so wird der restliche Pfad nicht weiter abgelaufen. Die Synchronisierung des *global stop sets* unter den Monitoren stellt die größte Herausforderung des Verfahrens dar [DFC05].

Bloom-Filter

Das Problem der Verteilung der *global stop sets* des Doubletree-Algorithmus wird heuristisch von Bloom-Filtern gelöst. Hier werden nicht komplette Mengen von Pfadinformationen zwischen den Monitoren geteilt, sondern lediglich ein Vektor fester Länge. Dieser ist in seiner initialen Form leer, besteht also durchgängig aus Nullen. In diesem wird durch die Platzierung von Einsen kodiert, ob ein Knoten in einem der Sets enthalten ist. Hierfür wird der Knoten durch die Verwendung einer oder mehrerer Hashfunktionen in eine Position des Vektors übersetzt und diese Position anschließend auf Eins gesetzt. Die Länge des Vektors beeinflusst die Wahrscheinlichkeit einer Kollision und damit die Ausdruckskraft des Filters [Dom+16].

Der Nachteil dieses Verfahrens ist, dass False Positives durchaus auftreten können. Wurde ein Interface jedoch besucht, kann dies durch die Verwendung eines Bloom-Filters garantiert nachgewiesen werden. Der enorme Vorteil dieses Verfahrens ist, dass die Größe des Sets bei noch guter Aussagekraft ca. um das 17-fache reduziert werden kann [DFC05].

Capping and Clustering

Die letzte betrachtete Möglichkeit um die Netzlast zu reduzieren, ist das sogenannte *Capping and Clustering*. Hier soll insbesondere für die ungewollte Wahrnehmung als DDoS-Attacke Abhilfe geschaffen werden. Hierbei wird ein einfaches Verfahren genutzt: Verschiedene Ziele werden zu Gruppen geclustert und es wird nur einer bestimmten Anzahl an Monitoren erlaubt, diese anzufragen. Auch so wird eine Verringerung der Netzlast erreicht [DFC05].

Kapitel 3

Modellierung von Netzwerken

In diesem Kapitel

In Kapitel 2 wurden vor allem die verschiedenen auf dem Markt verfügbaren Automatisierungssysteme mit ihren jeweiligen Eigenschaften aufgezeigt. Dieser Überblick ist nötig, um ein Modell für die Darstellung von Netzwerken zu finden, welches mit möglichst vielen GAS kompatibel ist. Des Weiteren muss dieses Modell die verschiedenen in 1.1 erwähnten Ansichten eines Netzwerks darstellen können: die logische Topologie, die physische Topologie und die funktionale Topologie.

In diesem Kapitel werden verschiedene Beschreibungsmöglichkeiten für Netzwerke behandelt, um einen Kandidaten für die Modellierung von Feldbussen zu finden. Besondere Beachtung findet dabei das offene Datenformat *AutomationML* (AML), welches sich aufgrund dessen vielseitigen Beschreibungsmöglichkeiten für die Darstellung der einzelnen Topologien eignet und deswegen im weiteren Verlauf der Arbeit verwendet wird.

3.1 Network Design Markup Language

Die Netzwerkbeschreibungssprache NDML, beziehungsweise NDML+, wurde von der TU Dresden im Rahmen des *Computer-Aided-Network-Design*-Projektes (CANDY) entwickelt, um die Planung von Ethernet-, WLAN- und WiMAX-Netzwerken zu vereinfachen. NDML+ erweitert NDML um weitere Planungsaspekte von Netzwerken wie Traffic-Mapping, Kosten und Analyse der Geometrie. Somit hebt sich NDML+ zu einer Reihe seinerzeit existierender Netzwerkbeschreibungssprachen ab [Lun+07].

Für die Grammatik von NDML+ wird die Erweiterte Backus-Naur-Form (EBNF) genutzt. Diese hat die Form $G(NDML+) = (T, V, P, S)$, wobei G die Grammatik, S das Startsymbol, V die Menge der Variablen, T die Menge aller Terminalsymbole und $P = \{V \times T\}$ die Menge aller Produktionsregeln ist. Die in S beinhalteten Symbole stellen dabei die verschiedenen Sichten dar, nach denen ein Projekt betrachtet werden kann (bspw. *Environment* für Bauplanung, *Cost* für Buchhaltung etc.). Nach [Lun+07] ist die Sprache wie folgt definiert:

- S := [ProjectInfo][Environment][Constraints][Traffic][Topology][Technology] [Test][Cost].
- ProjectInfo := Projects[ProjectDocumentation][ProjectContainers].
- Project := id ProjectName projectType manager customer buildingPlan.
- ProjectDocumentation := ProjectDocumentation.
- ProjectDocumentation := id docPath lastChange.
- ProjectContainers := ProjectContainer.
- ProjectDocumentation := vendor any.

In [LGS06] wird genauer auf die verwendeten Algorithmen des CANDY-Projektes eingegangen. Unter anderem werden Verfahren für die Modellierung der Wellenausbreitung in kabellosen Systemen betrachtet. Auch Betrachtungen zu der Modellierung der Planungskosten finden hier Anspruch.

NDML+ bietet eine Vielzahl an Beschreibungsmöglichkeiten zur Darstellung von Ethernet- und WLAN-Netzwerken. Ob eine Nutzung im Bereich der Feldbusse und GAS

dank dieser Vorarbeit möglich wäre, müsste genauer betrachtet werden. In dieser Arbeit wurde sich jedoch gegen die Nutzung von NDML+ entschieden. Dies hat hauptsächlich folgende Gründe:

- **Planungsorientierung**: Obwohl die hier betrachteten Sprachen allesamt an der Planung von Netzwerken orientiert sind, legt NDML+ einen besonderen Fokus auf Möglichkeiten zur Modellierung von Aspekten, welche den Rahmen dieser Arbeit weit übersteigen. So ist die Modellierung der Ausbreitung von Funknetzen im Reverse Engineering – welches in Kapitel 4 betrieben wird – schwer nachvollziehbar. Auch Aspekte wie die Kostenberechnung sind in der weiteren Nutzung dieser Arbeit als überflüssig zu betrachten.
- **Dokumentation**: Im Rahmen der Recherche von NDML+ wurde eine – besonders zu AutomationML – vergleichsweise spärliche Dokumentation gefunden. Dies erschwert die Arbeit mit dem Framework.
- **Alter und Verbreitung**: NDML+ [Lun+07] wurde im Jahr 2007 als Erweiterung von NDML [LGS06] spezifiziert und fand seitdem nur eine verhältnismäßig kleine Verbreitung. AML hingegen verbindet bereits genutzte, offene Industriestandards.

Zusammenfassend lässt sich sagen, dass die Sprache NDML+ für den vorliegenden Einsatzzweck nicht optimal ist. Darüber hinaus stellt sie jedoch einen interessanten Vorschlag mit guter Effizienz im Vergleich mit Netzwerksimulation [Vas+] und für die Planung von Netzwerken dar, auch wenn sie in der Praxis weniger Verwendung, als potentiell möglich, findet.

3.2 NevML

Eine weitere Möglichkeit zur Darstellung von Netzwerktopologien ist *NevML*. Dieses verfolgt ein ähnliches Ziel wie AML: Viele Modellierungssysteme nutzen keine offenen Standards zur Speicherung ihrer Informationen. Dies führt dazu, dass die dort erstellten Topologien nicht in fremde Systeme exportiert werden können [Wan10].

In NevML werden Geräte, Verbindungen und deren jeweilige grafische Darstellung getrennt voneinander modelliert. Die verschiedenen Objekte haben nach [Wan10] die folgenden Eigenschaften:

- **Device**: *id* (eindeutige ID), *name* (Bezeichner), *type* (Typ), *ip* (IP Adresse), *desc* (Beschreibung), *x* und *y* (Koordinaten für die Darstellung auf der Topologiekarte)
- **Connection**: *id*, *bandwidth* (Bandbreite der Verbindung), *source* (Quelle der Verbindung), *target* (Ziel der Verbindung), *desc*
- **Device Appearance**: *type* (ID für ein (grafisches) Symbol), *image* (Datei oder URL), *width* und *height* (Maße des Symbols)
- **Connection Appearance**: *bandwidth* (ID für eine Verbindung), *width* (Breite der Verbindung im Editor)

NevML bietet durch das vorgestellte Modell die Möglichkeit, die logische Topologie eines Netzwerkes detailreich abzubilden. Auch grafische Abbildungen werden ermöglicht. Trotz dessen wird gegen das Framework entschieden. Folgend werden Gründe aufgezeigt, weshalb NevML im Verlauf dieser Arbeit nicht zum Einsatz kommt:

- **Fokus**: NevML beschränkt sich weitgehend auf die Darstellung der logischen Topologie. Für diese Arbeit jedoch soll ein Modell genutzt werden, das alle drei in 1.1 beschriebenen Topologien umfasst, damit das entwickelte Konzept um diese erweiterbar ist. Dies ist ohne Weiteres mit Hilfe von NevML nicht möglich. Auch die Modellierung der individuellen Verbindungen zwischen logischen Geräten kann unter Umständen in Feldbussen nicht genau rekonstruiert werden. In NevML ist diese Modellierung jedoch nötig. Auch der Aspekt der grafischen Darstellung ist für diese Arbeit nicht von Relevanz.
- **Dokumentation und Verbreitung**: Die Dokumentation und Verbreitung von NevML sind vergleichbar zu NDML (siehe Abschnitt 3.1). Dies stellt sich auch bei NevML als Problem heraus.

Aus den oben genannten Gründen wird NevML hier nicht als Beschreibungsmittel genutzt. Ähnlich wie in Abschnitt 3.1 ist jedoch zu erwarten, dass die grafische Modellierung von logischen Netzwerktopologien für Zwecke außerhalb dieser Arbeit besser geeignet ist.

3.3 Network Markup Language

Die *Network Markup Language* (NML) stellt ein sehr umfangreiches Framework dar. Ähnlich wie bei den anderen vorgestellten Modellierungssprachen wird die Syntax von NML in XSD formuliert. Ein großer Unterschied zu den anderen Kandidaten ist jedoch, dass für die Modellierung der Netzwerke ein Datenbankschema verwendet wird. Verschiedenste Netzwerkobjekte werden in [Ham+13] definiert. In Abbildung 3.1 ist deren vollständiges Klassendiagramm zu sehen.

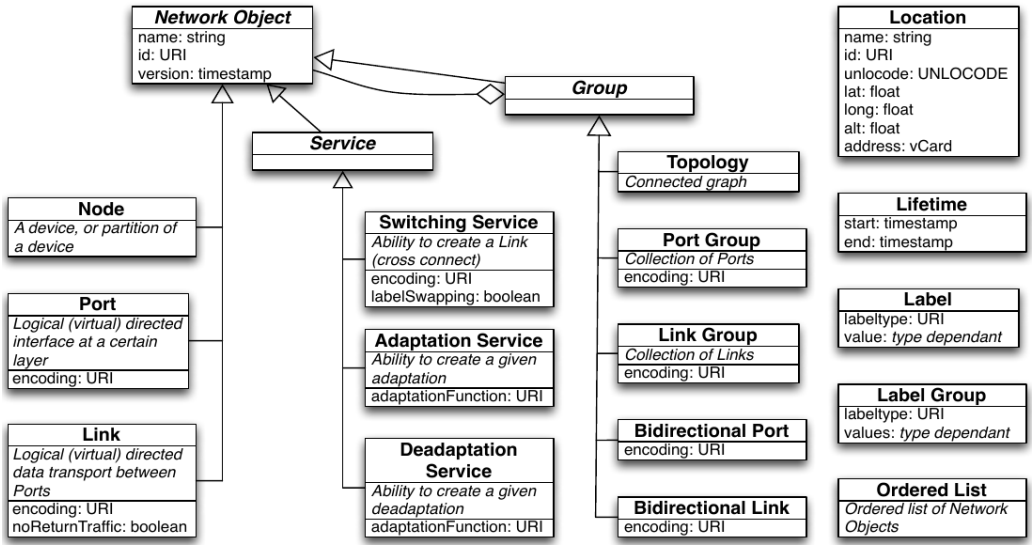


Abbildung 3.1: NML-Klassendiagramm aus [Ham+13]

Warum NML für diese Arbeit nicht genutzt wurde, lässt sich weitgehend mit **Fokus** erklären. Der wichtigste Grund ist, dass auch hier, ähnlich wie bei NevML (siehe Abschnitt 3.2), ausschließlich die logische Netzwerktopologie dargestellt wird. Zwar können auch Standorte mittels GPS-Koordinaten beschrieben werden, diese sind jedoch für Sicherheitsanalysen nicht aussagekräftig genug. An dieser Stelle kann jedoch erwähnt werden, dass NML eine umfangreiche Dokumentation besitzt.

3.4 AutomationML

Wie bereits in Abschnitt 1.1 erwähnt, sind drei Aspekte eines Netzwerks von der Dokumentenerosion betroffen: Die physische, die logische und die funktionale Topologie.

Diese beschreiben jeweils, wo Geräte installiert und wie sie miteinander verkabelt sind, die logische Einteilung dieser Geräte in Segmente, und die Relationen zwischen Geräten. Zur Speicherung dieser Informationen eignet sich das offene Datenformat *AutomationML* (AML), da dieses für jede dieser drei Ansichten Darstellungsmöglichkeiten definiert.

In den Abschnitten 2.2 und 2.3 wurden bereits einige Unterschiede in den verschiedenen verfügbaren GAS aufgezeigt. Besonders aus der Nutzung proprietärer Tools zur Konfiguration der Systeme entsteht das Problem, dass sich Netzkonfigurationen nicht leicht auf andere Standards überführen oder mit ihnen vergleichen lassen.

Ähnliche Probleme finden sich auch in anderen Branchen wieder. Hier hat sich herausgestellt, dass es für die Industrie sinnvoll ist, übergreifende Standards zur Beschreibung von ähnlichen Informationen zu entwickeln. So auch in der Anlagenplanung: Viele der darin beteiligten Disziplinen verfügten über eigene Tools und Verfahren. Dazu gehören Anlagenbau, Elektrotechnik, Verfahrenstechnik, Prozessleittechnik, HMI-Entwicklung, PLC-Programmierung etc. [Aut18]. Wieder und wieder stellt sich der Datenaustausch zwischen diesen Disziplinen als ein Flaschenhals heraus. Um diese sogenannte „Papierschnittstelle“ aus der Planung zu entfernen und durch ein offenes und erweiterbares Dateiformat zu ersetzten, wurde AML entwickelt [Dra+10]. Dies ist ein auf XML-Schema basierendes Format, welches verschiedenste Industriestandards in sich verbindet. Dazu gehören *CAEX* für die Abbildung von Anlagenstrukturen, *COLLADA* für die 3D-Modellierung und *PLCopen XML* für die Abbildung von Logik, Prozessen und Verhalten [SL15].

Im Folgenden sollen die einzelnen Standards erläutert werden und es wird gezeigt, wie AML, ein Format welches für den Datenaustausch von Anlagenplanungen konzipiert wurde, auch im Bereich der Gebäudeautomatisierung zur Modellierung von Feldbussen und insbesondere zur Darstellung von logischen, physischen und funktionalen Topologien verwendet werden kann.

3.4.1 Logische Topologie: CAEX

In Planungsprozessen von Anlagen herrscht eine hohe Heterogenität in Bezug auf die verwendeten Engineering-Werkzeuge. Einen standardisierten Werkzeugkatalog

für die gesamte Branche zu entwickeln, ist aufgrund der hohen Komplexität der Anwendungsfälle eine sehr schwierige Aufgabe. Um trotzdem einen Datenaustausch zu ermöglichen, wurde *CAEX* als ein unabhängiges Format zur Speicherung planungsrelevanter Daten entwickelt.

CAEX (Computer Aided Engineering Exchange) ist ein offenes Datenformat, basierend auf XML. Es wird genutzt, um die Struktur von Automatisierungssystemen und Anlagen zu modellieren [Ber+16]. Zudem bildet es die Grundlage von AML. In CAEX werden konkrete Inhalte in hierarchischer Form modelliert und Konzepte wie Objekt, Attribut, Schnittstellen, Hierarchien, Referenzen und Klassen definiert. Über CAEX werden auch die anderen in AutomationML verwendeten Standards – *PLCopen XML* und *COLLADA* – eingebunden. Des Weiteren werden vier Arten von Bibliotheken, welche den Vorteil der Wiederverwendbarkeit bieten und als Objektkatalog dienen, definiert [Dra+10; Aut18]. Es folgt eine Übersicht über die für diese Arbeit relevantesten Definitionen in CAEX.

InternalElement

Mit Hilfe des <`InternalElement`> werden konkrete oder logische Elemente in einer Anlage definiert. Diese Elemente zeichnen sich durch weitere Eigenschaften wie Attribute, Interfaces oder Verbindungen zu anderen Elementen aus. <`InternalElement`>-Objekte werden ineinander verschachtelt, um komplexe Strukturen zu erschaffen [Aut18].

ExternalInterface

Das <`ExternalInterface`> wird genutzt, um Schnittstellen für Relationen zwischen Objekten des Typs <`InternalElement`> zu erzeugen. Auch können so projektexterne Dateien eingebunden werden – dazu zählen die Formate COLLADA und PLCopen XML. Für diese werden jeweils die Klassen `COLLADA-`, `PLCopenXMLInterface` und `ExternalDataReference` bereitgestellt. Eingebunden werden Dateien über das Attribut `refURI` [Aut18].

InternalLink

<InternalLink> beschreibt die Verknüpfung von zwei oder mehreren Instanzen des Typs <ExternalInterface>. Sie sind damit die konkreten Abbildungen von Relationen und Verbindungen zwischen Objekten in CAEX [Aut18].

InstanceHierarchy

Die Instanzhierarchie ist ein CAEX-Objekt, unter dem eine konkrete Anlage mit Hilfe der oben beschriebenen Klassen modelliert wird [Aut18]. In Abbildung 3.3 kann (oben links) gesehen werden, wie ein Netzwerk in einer solchen Hierarchie modelliert wird.

RoleClass

Die <RoleClass> dient zur Definition von abstrakten Klassen zur allgemeineren Beschreibung eines <InternalElement>. Dies ist nötig, da die Bezeichner der Elemente in der Instanzhierarchie nicht notwendigerweise ihre Funktion erklären und dies zu Unverständlichkeiten führen kann. Aus diesem Grund kann Dank der <RoleClass> jedem Element ein allgemeiner Bezeichner zugeordnet werden [Aut18].

RoleClassLib

Unter der <RoleClassLib> werden die verschiedenen <RoleClass> definiert und in einer Bibliothek zusammengefasst [Aut18]. Ein Beispiel für eine solche Bibliothek ist in Abbildung 3.3 (unten links) gezeigt.

InterfaceClassLib

Unter der <InterfaceClassLib> werden die verschiedenen Objekte der <InterfaceClass> definiert und zusammengefasst. Die drei durch [Aut18] vordefinierten Klassen von <ExternalInterface> sind in Abbildung 3.2 gezeigt.

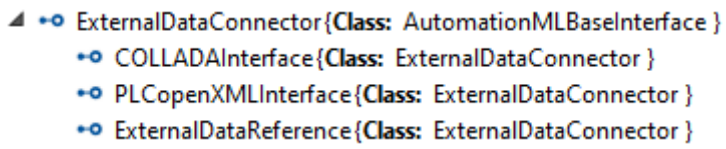


Abbildung 3.2: Ext. Daten in `AutomationMLBaseInterfaceLib` des AML-Standards

AttributeTypeLib

Unter der <`AttributeTypeLib`> werden die verschiedenen Attribute für Klassen im XML-Schema definiert und zusammengefasst. Diese können dann <`InternalElement`> und <`RoleClass`> zugeordnet werden [Aut18]. Zu sehen ist diese in 3.3 (unten rechts).

Group

Mit Hilfe der AML-<`Group`>-Objekte lassen sich `InternaElements` zu Gruppen zusammenfassen. Dies ist nützlich, um Strukturinformationen und Instanzinformationen getrennt voneinander darzustellen. So lassen sich beispielsweise Sichten für verschiedene Einsatzbereiche definieren. Die Objekte, welche hierarchisch unter einer Gruppe zusammengefasst werden, sind gespiegelte Instanzen von in der Instanzhierarchie vorkommenden `InternaElements`. Die Zuweisung eines gespiegelten Objekts zu seiner konkreten Instanz wird mittels des Attributes `RefBaseSystemUnitPath` hergestellt, welches dann auf die ID des konkreten Objektes verweist [Aut18].

Am Beispiel: Der AutomationML-Editor

Die im Standard von AML enthaltenen Funktionen und Klassen werden durch eine vierteilige Dokumentation ausführlich beschrieben. Diese beschreiben die Architektur des Frameworks [Aut18], die verschiedenen bereitgestellten Klassenbibliotheken [Aut14] sowie die Darstellung von Kinematik, Geometrie [Aut17a] und Logik [Aut17b].

Des Weiteren existiert ein offizieller AML-Editor, welcher die Funktionen des Formats beherrscht und eine Übersicht über Projekte gibt. Er eignet sich auch, um einfache Anlagen selbst zu entwerfen. In Abbildung 3.3 wird ein solches Projekt gezeigt.

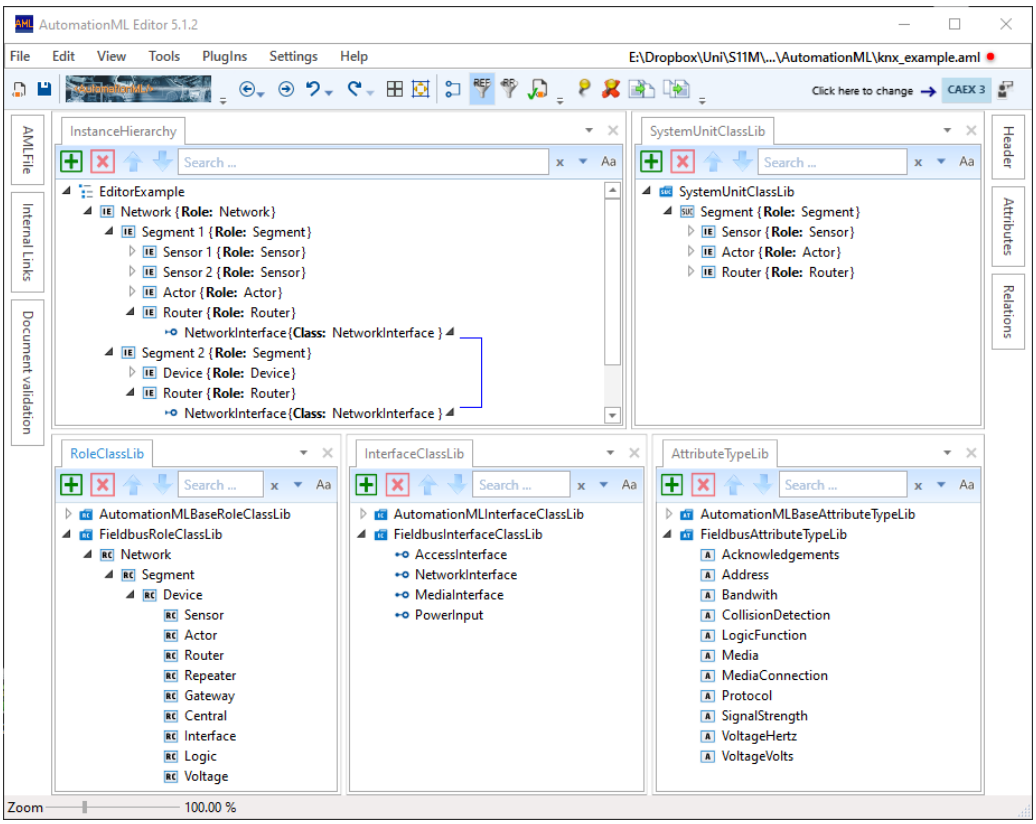


Abbildung 3.3: Beispielhafter Aufbau eines Feldbusses im AML-Editor

Hierbei handelt es sich um einen Feldbus, welcher in der Instanzhierarchie (oben links) modelliert wurde. Die für dieses Modell genutzten Geräte wurden in der Rollenbibliothek (unten links) vordefiniert und können im Editor per Drag-and-Drop in die Instanzhierarchie kopiert werden. Alle genutzten Interfaces werden in ihrer entsprechenden Bibliothek (unten Mitte) aufgelistet und auch Attribute für die Klassen wurden (unten rechts) festgelegt. Des Weiteren kann in der Instanzhierarchie gesehen werden, wie eine Verbindung zwischen Segmenten hergestellt werden kann. Hierfür werden Interfaces miteinander verknüpft, was im Editor durch die blaue Linie dargestellt wird.

Das Beispiel in Abbildung 3.3 zeigt, wie dank AML mit einfachen Mitteln die logische Topologie eines Netzwerkes modelliert werden kann. Auf die genaue Definition der Klassen wird in Kapitel 4 genauer eingegangen.

3.4.2 Geometrie und Kinematik: COLLADA

In der 3D-Modellierung wurden für lange Zeit keine offenen Formate genutzt. Dies lag unter anderem daran, dass die verwendeten Entwicklerwerkzeuge sich stark voneinander unterschieden und für die Distribution von Modellen in Spielen oder anderen Medien meist optimierte, binäre Formate verwendet wurden. Oft ließen sich Assets nur sehr schwer in fremde Projekte einbinden und viele Tools verfügten nicht über die Möglichkeit, Modelle zu exportieren. Der Datenaustausch stellt auch in diesem Fall ein großes Problem dar [Col].

COLLADA (COLLAborative Design Activity) dient als offenes und standardisiertes Format zum Austausch von 3D-Inhalten [Dra+10]. Die Hauptziele von COLLADA sind nach [Col] die folgenden:

- Bereitstellung eines gut spezifizierten, XML-basierten, kostenlosen, offenen Formats, um unabhängig von binären Darstellungen zu sein.
- Schaffung einer gemeinsamen Sprache, um COLLADA Assets in existierenden Tool-Chains zu integrieren.
- Adoption von so vielen Nutzern wie möglich.
- Bereitstellung eines einfachen Integrationsmechanismus, sodass alle Informationen durch COLLADA bereitgestellt werden können.
- Schaffung einer gemeinsamen Basis für den Austausch von 3D-Informationen zwischen Anwendungen.
- Katalyse von digitalen Inhalten für verschiedene Branchen.

Diese Zielsetzungen machen COLLADA zu einem ausgezeichneten Kandidaten für den Einsatz in verschiedenen, weiterführenden Tools. Das zeigt sich auch darin, dass das Format bereits in einer Vielzahl von Anwendungen integriert wurde. Als Beispiele für namentliche Vertreter sind hier Autodesk, Google Earth, Maple, macOS Preview, NASA WorldWind und SketchUp zu nennen. Durch die weite Verbreitung wäre COLLADA in jedem Fall ein geeigneter Kandidat für die Speicherung von 3D-Informationen, unter anderem für die Darstellung der *physischen Toplogie* eines Feldbusses.

Ein COLLADA-Dokument folgt dabei dem Aufbau wie in Listing 3.1 gezeigt. Wichtig sind die Wurzel des Dokumentes <COLLADA>, sowie die sich darunter befindenden Elemente <assets>, <library_geometries>, <library_visual_scenes>, <library_materials>, <scene> und <extra> [Col; Dra+10].

```
<?xml version="1.0" encoding="utf-8"?>
<COLLADA xmlns=
  "http://www.collada.org/2008/03/COLLADASchema"
  version="1.5.0">
  <asset>
    <created/>
    <modified/>
  </asset>
  <library_geometries/>
  <library_visual_scenes/>
  <scene />
</COLLADA>
```

Listing 3.1: Aufbau einer COLLADA-Datei nach [Col]

Wie in Kapitel 1.4 bereits beschrieben, wird die Umsetzung der *physischen Topologie* eines Feldbusses in dieser Arbeit nicht genauer betrachtet. Da COLLADA jedoch ein möglicher Kandidat für weiterführende Arbeiten ist, soll die grundlegende Struktur des Formats in Folge erläutert werden.

assets

Das Element <assets> dient dazu, Metainformationen über ein COLLADA-Objekt zu sammeln. In diesem können wiederum verschiedenste Elemente verschachtelt werden. Zu diesen gehören einschließlich: das Element <contributor>, in welchem die Ersteller des Elternelements beschrieben werden, <coverage>, welches die Platzierung des Elements im *realen* Raum beschreibt, <title>, welches den Titel des Elternelements definiert sowie <unit>, mit dem die genutzten Einheiten beschrieben werden [Col].

```
<COLLADA>
  <asset>
    <created>2005-06-27T21:00:00Z</created>
    <keywords>COLLADA interchange</keywords>
    <modified>2005-06-27T21:00:00Z</modified>
    <unit name="nautical_league" meter="5556.0" />
    <up_axis>Z_UP</up_axis>
  </asset>
</COLLADA>
```

Listing 3.2: Aufbau des Elements <asset> in COLLADA nach [Col]

library_geometries

Hier werden Elemente des Typs <geometry> platziert. Wie der Name bereits verrät, dient dieses Element dazu, Informationen über die Geometrie des beschriebenen Objektes zu speichern. Dazu gehören Formen wie B-Spline, Bézier, Mesh, NURBS und Patch. Es werden nur polygonale Geometriebeschreibungen von COLLADA unterstützt. Des Weiteren können auch Elemente des Typs <asset> und <extra> hier Platz finden [Col].

```
<library_geometries>
  <geometry name="cube" id="cube123">
    <mesh>
      <source id="box-Pos"/>
      <vertices id="box-Vtx">
        <input semantic="POSITION" source="#box-Pos"/>
      </vertices>
    </mesh>
  </geometry>
</library_geometries>
```

Listing 3.3: Aufbau des Elements <library_geometries> in COLLADA nach [Col]

library_visual_scenes

Hier werden Elemente des Typs <visual_scene> in einheitliche Bibliotheken geordnet. Diese enthalten die Informationen, gespeichert in <node>-Elementen, welche

alle darzustellenden grafischen Informationen beschreiben [Col].

```
<library_visual_scenes>
  <visual_scene id="vis_sce1">
    ...
  </visual_scene>
  <visual_scene id="vis_sce2">
    ...
  </visual_scene>
</library_visual_scenes>
```

Listing 3.4: Aufbau des Elements <library_visual_scenes> in COLLADA nach [Col]

library_materials

In <library_materials> werden Objekte des Typs <material> gesammelt. Diese enthalten Formeln und deren Parameter, um spezielle Effekte grafischer Objekte darzustellen. Dies können beispielsweise Effekte wie Blurs, Blooms, Farbfilter oder Kameralinsen-Effekte sein [Col].

```
<library_materials>
  <material id="mat1">
    ...
  </material >
  <material id="mat2">
    ...
  </material>
</library_materials>
```

Listing 3.5: Aufbau des Elements <library_materials> in COLLADA nach [Col]

scene

Die durch die COLLADA-Datei bereitgestellte Szene wird in <scene> beschrieben. Hier werden visuelle und transformative Informationen definiert. Weiterhin können unter diesem die Elemente <instance_physics_scene>, <instance_visual_scene> und <instance_kinematics_scene> gespeichert werden [Col].

```
<COLLADA>
  <scene>
    <instance_visual_scene url="#world"/>
  </scene>
</COLLADA>
```

Listing 3.6: Aufbau des Elements <scene> in COLLADA nach [Col]

extra

<extra> kommt in den meisten von COLLADA spezifizierten Elementen als Kind-Knoten vor, um weitere konkrete Inhalte oder semantische Metainformationen festzulegen. Das einzige Element, welches in <extra> vorkommen muss, ist <technique>. In diesem werden Informationen für externe Plattformen und Programme definiert, welche die COLLADA-Datei später bearbeiten könnten [Col].

```
<geometry>
  <extra>
    <technique profile="Max" xmlns:max="some/max/schema">
      <param name="wow" sid="animated" type="string">
        a validated string parameter from the COLLADA schema.
      </param>
      <max:someElement>
        defined in the Max schema and validated.
      </max:someElement>
      <uhoh>
        something well-formed and legal, but that cant be
        validated because there is no schema for it!
      </uhoh>
    </technique>
  </extra>
<geometry>
```

Listing 3.7: Aufbau des Elements <extra> in COLLADA nach [Col]

Wie bereits erwähnt wird die Modellierung der *physischen Topologie* in dieser Arbeit nicht genauer betrachtet, da es sich bei COLLADA um einen zu umfangreichen Standard handelt. Die soeben vorgestellte Struktur des Formats muss demzufolge genügen, um zu zeigen, dass es sich hierbei um einen potenten Kandidaten für die Darstellung von geometrischen Informationen von Feldbussen handelt.

3.4.3 Prozesse: PLCopen XML

Industrielle Anlagen verfügen natürlich nicht nur über einen einzigen physischen Zustand, sondern haben in der Regel ein komplexes Verhalten in dem viele Subsysteme miteinander interagieren. Im Sinne von AML wird unter anderem von mechanischen Bauteilen und physischen Prozessen, wie beispielsweise Förderbändern und anderen Aktoren, ausgegangen. Nach [Dra+10] wird in *PLCopen XML* zwischen drei verschiedenen Arten des modellierten Verhaltens unterschieden:

- **Sequencing** beschreibt das gesteuerte Verhalten, also das Verhalten, welches für die Ausführung von Aktionen erwünscht ist. Hierzu zählt beispielsweise die Funktion eines Förderbandes, welches einen Gegenstand oder ein Produkt befördert. Es beschreibt die zielgerichtete Ausführung von Aktionen zum Auslösen bestimmter Ereignisse.
- **Behaviour** ist das Verhalten eines Systems, welches ungesteuert passiert und von der zugrunde liegenden Physik ausgelöst wird. Dabei kann, abhängig von der betrachteten Anlage, oftmals das *Sequencing* eines Systems von einem anderen Subsystem als *Behaviour* betrachtet werden. So sehen beispielsweise Systeme, welche mit dem oben genannten Förderband interagieren, dessen Effekte nicht als *Sequencing*, sondern als *Behaviour*. Aktoren besitzen Beschreibungen für beide Arten von Verhalten.
- **Interlocking** beschreibt Aktionen, welche nur unter bestimmten Bedingungen ausgeführt werden und vom Systemzustand abhängig sind. Das *Interlocking* wird logisch ausgedrückt und unterbindet, beziehungsweise erlaubt, die Überführung von Systemzuständen unter Betrachtung der Zustände anderer Aktoren im System.

Jedes der oben vorgestellten Verhalten wird durch ein bestimmtes Beschreibungsmittel spezifiziert. Dies geschieht für jedes Objekt des modellierten Systems. PLCopen XML ist ein mächtiges Framework, welches viele Modellierungssprachen zusammenfasst. Deshalb wird vom AML-Konsortium empfohlen, die Implementierung dieser Formate über einen *Intermediate Modelling Layer* (IML) zu realisieren [Dra+10].

Bei den unterstützten Formaten handelt es sich um Gantt Charts, PERT Charts, Sequential Function Charts, Impulsdiagramme, Logiknetzwerke und State Charts, welche für unterschiedliche Verhalten und in unterschiedlichen Abschnitten der Anlagenentwicklung verwendet werden. Welches Verfahren in welchem Stadium der Anlagenplanung genutzt wird, ist in Tabelle 3.1 aufgezeigt.

Produkt-Entwurf	**Anlagen-Planung**	**Mechanik-Planung**	**Elektro-Planung**	**SPS-Progr.**	**Roboter-Progr.**	**HMI-Progr.**	**Inbetr.-nahme**
Gantt Charts	Gantt Charts, PERT Charts	Gantt Charts, PERT Charts, State Charts, SFC	Impuls-Diagr., Logik-Netzw., State Charts, SFC	Impuls-Diagr., Logik-Netzw., State Charts, SFC	Impuls-Diagr., Logik-Netzw., State Charts, SFC	Impuls-Diagr., Logik-Netzw., State Charts, SFC	State Charts, SFC

Tabelle 3.1: Modellierung verschiedener Abschnitte der Anlagenplanung nach [Dra+10]

Dabei ist zu beachten, dass nicht jedes der Beschreibungsmittel für Sequencing, Behaviour und Interlocking verwendet werden kann. Mit welchen Mitteln diese darzustellen sind, wird in Tabelle 3.2 als Steuerungsbeschreibung, Verhaltensbeschreibung und Verriegelungslogik aufgeführt.

Beschr. von Fertigungsproz.	**Beschr. von Anlagenst.**	**Beschr. von Verriegelungslog.**	**Steuer.beschr. der Anlage.**	**Verh.beschr. der Anlage**
Gantt Charts, PERT Charts	Impuls-Diagramme	Logiknetzwerke	Sequential Func. Charts	State Charts, Sequential Func. Charts

Tabelle 3.2: Modellierungssprachen einzelner Aspekte der Anlage nach [Dra+10]

Die oben gezeigten Mittel dienen dazu, das Verhalten einer Anlage zu beschreiben. Diese Beschreibungsmittel eignen sich am ehesten in AML dafür, die *funktionale Topologie* eines Feldbusses darzustellen. Um ein geeignete Verfahren zu finden, muss Tabelle 3.1 erneut betrachtet werden.

Hier wird der Planungsprozess einer Anlage beschrieben. In gewisser Hinsicht lässt sich die automatische Generierung der funktionalen Topologie als invers zur eigentlichen Zielsetzung von PLCopen XML interpretieren. In diesem *Reverse Engineering* wird eine komplettierte Anlage betrachtet und versucht, die Planungsdokumente nachzuvollziehen. Aus diesem Grund liegt der Fokus für die Wahl des geeigneten Verfahrens am hinteren Ende von Tabelle 3.1. Auch ist für die funktionale Topologie eines Netzwerks nur das Sequencing und das Behaviour – die Steuerungsbeschreibung und die Verhaltensbeschreibung, gezeigt in Tabelle 3.2 – der Anlage relevant. Werden diese Aspekte zusammen betrachtet, so kommen zur Modellierung der funktionalen Sicht nur Sequential Function Charts und State Charts als Kandidaten vor.

Auch die *funktionale Topologie* von Feldbussen wird im Rahmen dieser Arbeit nicht genauer behandelt, da es sich auch bei PLCopen XML um einen zu umfangreichen Standard handelt. Ob die vorgeschlagenen Mittel tatsächlich zu dessen Modellierung geeignet sind, sollte weiter untersucht werden.

3.5 Zusammenfassung

In Abschnitt 1.1 werden drei verschiedene Topologien erwähnt: Die *logische Topologie*, die *physische Topologie* und die *funktionale Topologie*. In diesem Kapitel wurde gezeigt, welche Beschreibungsmittel existieren, um Netzwerke darzustellen. Hier wurde AML vor allem deshalb gewählt, weil es eine ähnliche Unterteilung in Topologien vorgibt.

Es wurde gezeigt, wie die *logische Topologie* durch die Nutzung von CAEX abgebildet werden kann. In Folge wurde das Format COLLADA, welches im Bereich der 3D-Modellierung genutzt wird, kurz erklärt. Diese wird in dieser Arbeit genutzt, um die *physische Topologie* darzustellen. In dem vorhergehenden Abschnitt wurde PLXopen XML erklärt, welches zur Darstellung der *funktionalen Topologie* genutzt wird.

So wurde gezeigt, wie alle in Abschnitt 1.1 genannten Aspekte des Netzwerks durch AML abgedeckt werden können. Dies ist vor allem für weiterführende Arbeiten relevant. Das nächste Kapitel widmet sich der Formulierung eines konkreten Konzeptes, welches die Darstellung der *logischen Topologie* behandelt.

Kapitel 4

Logische Topologieerkennung

In diesem Kapitel

Feldbusse können aus drei verschiedenen Sichten betrachtet werden. Konkret sind damit die *logische Topologie*, die *physische Topologie* und die *funktionale Topologie* gemeint. Jede dieser Topologien beschreibt einen anderen Aspekt des Netzwerkes. Die logische Topologie hält die logischen Verbindungen zwischen Geräten des Netzwerks fest. In ihr ist gezeigt, wie einzelne Netzknoten miteinander verbunden und in welchen Segmenten sie angeordnet sind. In der physischen Topologie ist die Position aller Elemente des Netzwerkes im physischen Raum und in der funktionalen Topologie das Verhältnis der Geräte und ihre Aufgaben beschrieben.

Das in diesem Kapitel vorgestellte Konzept soll die Darstellung der logischen Topologie eines Netzwerkes ermöglichen. Hierfür wird das offene Datenformat *AutomationML* (AML) genutzt, welches in Kapitel 3.4 behandelt wurde. Die anderen genannten Topologien werden aufgrund des hohen Mehraufwandes der dafür in AML vorgesehenen

Standards nicht behandelt. Seinen Nutzen soll das erarbeitete Modell in der Analyse der Sicherheit von Feldbussen finden.

Wichtig ist, dass sich mit Hilfe des Konzeptes möglichst alle GAS beschreiben lassen. Hierfür wurde in Kapitel 2.3 eine Auswahl der am Markt verfügbaren Systeme vorgestellt. Auch die hauptsächlich von den GAS genutzten Medien wurden in Kapitel 2.2 aufgezeigt. Die wichtigsten Aspekte aus dieser Übersicht sind dabei die Nutzung eines kabelgebundenen oder kabellosen Mediums sowie die durch den jeweiligen Standard ermöglichten Netzwerktopologien.

Auch eine Auswahl der zur Topologiegenerierung nutzbaren Datenquellen finden in diesem Kapitel Ansprache. Besonders auf die Dokumentenerosion (siehe Abschnitt 1.1) muss hier geachtet werden. Gleichzeitig werden Qualitätskriterien definiert, mit denen die erzeugten Topologien bewertet werden können.

Für die Erprobung wird KNX als stellvertretendes GAS genutzt. Diese Wahl wurde aufgrund dessen hoher Verbreitung und vielseitiger Einsatzmöglichkeiten getroffen. Für die Modellierung wird das Datenformat AML genutzt, da es auch Möglichkeiten für die Darstellung der anderen in Abschnitt 1.1 genannten Topologien unterstützt.

4.1 Anforderungen

Die Anforderungen des Modells leiten sich aus dessen Nutzen zur Sicherheitsanalyse und aus der Notwendigkeit der Modellierung verschiedener GAS-Netzwerktopologien ab. Diese Topologien sind hauptsächlich von dem verwendetem Medium abhängig. Einheiten wie KNX-Bereiche und -Linien oder Funkreichweiten von kabellosen Repeatern anderer GAS werden hier verallgemeinert als *Segmente* dargestellt. Werden diese verschachtelt angeordnet, so lassen sich bereits dadurch komplexe Netzwerke aufbauen.

Einzelne Geräte der Netzwerke werden mit einfachen Klassen modelliert, welche sich durch ihre Attribute unterscheiden. Deren Definition erfolgt in Abschnitt 4.3.4 und nutzt ein aus der Dokumentation von AML bekanntes Format. Auch Gruppen müssen dargestellt werden können, wenn das betrachtete GAS diese unterstützt.

Es wird eine Reihe an Datenquellen für die Topologiegenerierung vorgeschlagen. Besonders werden aber die Aspekte des passiven und des aktiven Scanns als solche behandelt. Diese entnehmen ihre Informationen dann direkt dem Feldbus. Mögliche Eigenschaften der Geräte und Segmente, welche sich aus diesen ableiten lassen sollen, werden folgend genannt:

- Adressen
- Zugriffsmedium
- Signalstärke
- Protokoll
- Kanal
- Übertragungsgeschwindigkeit
- Acknowledgement
- Kollisionsbehandlung

Zu den Eigenschaften der einzelnen Geräte gehört die **Adresse**. Diese ist jedem im Netzwerk vorkommenden Gerät zugewiesen und dient auch als eindeutiger Bezeichner. Das **Zugriffsmedium** ist eines der in Abschnitt 2.2 genannten Medien und sollte vor allem bei Gateways mitgeführt werden. Ist dies kabellos, so sollte auch für jedes Gerät die **Signalstärke** zu dessen jeweiligem Repeater mit aufgenommen werden. Wenn segmentgebende Repeater betrachtet werden, muss jedoch auf das Mitführen der Signalstärke verzichtet werden, wenn diese keinen klaren Bezug zu *einem* Kommunikationspartner hat.

Segmente werden Eigenschaften zugeordnet, welche in der Regel für alle in ihnen enthaltenen Geräte gelten. Dazu gehört auf jeden Fall das in dem jeweiligen Segment gesprochene **Protokoll**. Segmente verschiedener Protokolle können durch zwei Instanzen der Klasse *Repeater* und die Verwendung des AML-<`InternalLink`> (siehe Abschnitt 3.4.1) verknüpft werden. Weitere Eigenschaften von Segmenten sind in Abhängigkeit ihres Protokolls die **Übertragungsgeschwindigkeit**, die Nutzung von **Acknowledgements** und die **Kollisionsbehandlung**, sowie bei kabellosen Segmenten der verwendete **Kanal**. Die Abgrenzung von Segmenten passiert zum Beispiel bei dem Wechsel des Mediums, des Protokolls, des Bereichs oder der Linie (siehe KNX in Abschnitt 2.3.1), der Domain (siehe LonWorks in Abschnitt 2.3.2) oder des Funkbereichs

eines kabellosen Repeaters.

Die genannten Eigenschaften sind wichtig, um das logische Verhalten eines Netzwerkes zu modellieren. Auf die Abbildung konkreter Verbindungen zwischen Geräten wird verzichtet, um die Kompatibilität zwischen kabelgebundenen und kabellosen GAS herzustellen.

4.1.1 Geräteklassen

Neben den in 4.1 genannten Eigenschaften lassen sich Geräte in bestimmte Klassen einteilen, welche in vielen verschiedenen GAS vorkommen. Viele dieser Geräteklassen sind in [Asc14] aufgezeigt und werden entsprechend in das Konzept übernommen:

- Sensoren
- Aktoren
- Router
- Repeater
- Gateway
- Logic Devices
- Zentrale
- Voltage

Sensoren und **Aktoren** sind in GAS meist die ausführenden Elemente und grundlegend für die Automatisierung zuständig. Bei Sensoren handelt es sich um jegliche Form von Geräten, welches eine Information aus seinem Umfeld aufnimmt: Seien es Helligkeit, Luftfeuchtigkeit, Wärme oder ein einfacher Tastenschalter. Es ist typisch für Sensoren, in einem funktionalen Verhältnis zu Aktoren zu stehen, welche auf die erfassten Umgebungsinformationen reagieren, sprich Aktionen wie Licht an- und ausschalten, Fenster öffnen, heizen und Türen öffnen, ausführen.

In IP-Netzwerken vermitteln **Router** Pakete zwischen verschiedenen Subnetzen. In Feldbussen finden sich, abhängig vom verwendeten GAS, ähnliche Konzepte wieder. Am Beispiel von KNX werden Telegramme zwischen Linien und Bereichen ausgetauscht.

Repeater sind häufig in kabellosen Systemen aufzufinden, da die Signalstärke der Geräte stark abhängig von deren Distanz zueinander sowie den Gebäudeeigenschaften ist. Beispiele dafür sind die Standards ZigBee und Z-Wave, welche in Kapitel 2.3 vorgestellt wurden. Auch in KNX treten verwandte Konzepte auf. Hier werden Linienverstärker benutzt, um die maximale Anzahl an Geräten auf einer Linie sowie dessen maximale Reichweite zu erhöhen. Diese Verstärker werden hier als Repeater klassifiziert.

Auch **Gateways** sind ein bekanntes Konzept aus der Netzwerktechnik. In GAS sind sie die Geräte, welche Verbindungen zwischen verschiedenen Protokollen und Medien herstellen. Allein in KNX wird eine Vielzahl an Medien unterstützt. Netzsegmente auf verschiedenen Medien wie KNX-TP, KNX-PL, KNX-RF und KNX-IP werden durch KNX-Gateways miteinander zu einem Netzwerk verbunden. Auch existieren zahlreiche Gateways, welche Verbindungen zu anderen Protokollen herstellen.

Die klassischen logischen Funktionen *AND*, *OR* und *XOR* können in GAS wie KNX durch spezielle Module implementiert werden. In der Regel ist der unterstütze Funktionsumfang jedoch deutlich höher. So werden unter anderem auch Filter, Inverter, Delays, Konverter und Zufallsgeneratoren durch solche Module realisiert. Trotz der letztendlich sehr umfangreichen Funktionalität werden hier diese Typen von Geräten unter der Klasse **Logic Devices** zusammengefasst.

In einigen GAS werden **Zentralen** zur Ausführung von Funktionen genutzt. Diese sich meist in zentraler Lage des Gebäudes befindlichen Systeme sind dann notwendig für jegliche Automatisierung. Sensoren und Aktoren kommunizieren in solchen Systemen mit der Zentrale, statt direkt miteinander. Oftmals finden sich solche Strukturen in rein Powerline-basierenden GAS.

In vielen GAS – besonders in Powerline-basierenden Systemen – existiert eine Vielzahl an Geräten, welche verschiedenste Aufgaben im Bereich der Elektrotechnik übernehmen. Dazu gehören Phasenkoppler, Bandsperren, Stromzähler und auch die Stromversorgung für den Feldbus an sich. In dieser Arbeit wird diese Klasse an

Geräten lediglich unter dem Bezeichner **Voltage** zusammengefasst, da eine individuelle Betrachtung einzelner Gerätetypen allein durch die spätere Realisierung des Konzeptes in KNX nicht zielführend wäre.

4.1.2 Geräte in kabelgebundenen Systemen

Bezüglich auf das von ihnen verwendete Medium lassen sich GAS in zwei verschiedene Gruppe einteilen: Die kabelgebundenen und die kabellosen Systeme. Diese besitzen charakteristische Unterschiede, besonders im Hinblick auf ihre Topologie. Unter der Betrachtung von KNX allein fällt es leicht, ein Netzwerk in Segmente einzuteilen.

In KNX existieren Geräte auf einer Linie. Diese können wiederum zu Bereichen zusammengefasst werden, indem sie durch Linienkoppler auf einer neuen Linie verbunden werden. Die Bereiche werden dann über Bereichskoppler auf der sogenannten Backbone-Linie zu dem Gesamtsystem verbunden. Die hierarchische Struktur ist in diesem Fall bereits vorgegeben und Bereiche sowie Linien und Backbone können ebenso durch anonyme Konzepte, die Segmente, zusammengefasst werden. Die meisten der in Abschnitt 4.1.1 genannten Geräteklassen lassen sich in KNX wiederfinden.

GAS wie LonWorks unterstützen eine Vielzahl an Topologien. Auch diese müssen durch ein einheitliches Konzept dargestellt werden können. Die erste Segmentierung findet bei der Nutzung verschiedener LonWorks-Domains statt. Dann findet eine Einteilung in bis zu 255 Linien statt – auch diese werden als Segmente betrachtet. Eine weitere, topologieabhängige Unterteilung in zusätzliche Segmente ist an dieser Stell nicht mehr von Nöten. Auch in LonWorks finden sich typischerweise Geräte wie Router, Gateways und Stromversorger.

Ein Beispiel für eine kabelgebundene GAS-Konfiguration ist in Abbildung 4.1 zu sehen. Hier kann es sich um ein KNX-Netzwerk handeln, in dem die blau und rot markierten Geräte jeweils Teil einer Linie sind, während die grünen zu dem verbindenden Bereich gehören. Da in KNX die Linienkoppler (Router) Teil ihrer jeweiligen Linie sind, werden sie auch dem entsprechenden Segment zugeordnet. Auch der übergreifende Bereich wird als Segment dargestellt.

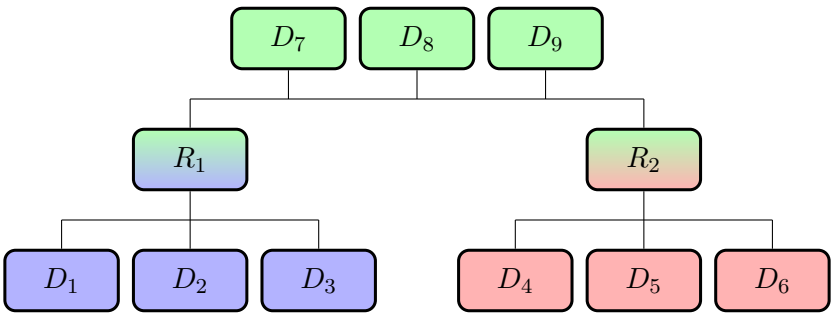


Abbildung 4.1: Beispiel für eine kabelgebundene GAS-Konfiguration

Anhand der hier betrachteten kabelgebundenen Beispiele lässt sich feststellen, dass die in Abschnitt 2.3 vorgestellten GAS bereits durch eine einfache, hierarchische Unterteilung in Segmente dargestellt werden können. Zu beachten ist auch, dass Geräte hierbei eindeutig in ihrer logischen Position bestimmbar sind.

4.1.3 Geräte in kabellosen Systemen

Kabellose Systeme sind hingegen komplizierter. Meist sind Geräte in diesen GAS aufgrund ihrer kabellosen Vernetzung in vermaschten Topologien angeordnet. Das bedeutet, dass keine klare Segmentierung des Netzwerkes wie bei den kabelgebundenen Systemen vorgenommen werden kann. Deshalb erfolgt hier die Einteilung in Segmente anhand der Zugehörigkeit zu Routern und Repeatern. Das bedeutet, dass alle Geräte, welche sich in der Sendereichweite eines Routers oder Repeaters befinden, damit auch in dessen Segment angeordnet sind. Ein Resultat dieses Vorgehens ist, dass die gleichen Geräte hier auch in mehreren Segmenten gleichzeitig existieren können.

Dieses Vorgehen ist für beide in Abschnitt 2.3 vorgestellten kabellosen GAS zulässig. In ZigBee werden die sogenannten Full Function Devices zur Weiterleitung von Telegrammen verwendet, da nur diese über die dafür nötigen Systemressourcen verfügen. Ähnlich verhält sich Z-Wave, bei welchem die Geräte in Controller und Slaves eingeteilt werden. Hier dürfen jedoch alle nicht-batteriebetriebenen Geräte unabhängig von ihrer Einteilung als Repeater dienen.

Paketweiterleitende Geräteklassen existieren in beiden Systemen und die Einteilung in Segmente erfolgt über diese. Dass Geräte demzufolge nicht mehr eindeutig

einem Segment zugeordnet werden, ist dabei nicht nur eine Konsequenz des vorgeschlagenen Modells, sondern auch der zugrunde liegenden Netzwerktechnik, mit welcher dies auch oft nicht möglich ist.

Das Beispiel in der Abbildung 4.2 zeigt das Problem der mehrfachen Einteilung in Segmente auf. Hier existieren drei Router, welche mit Geräten in ihren jeweiligen Einflussradien verbunden sind. Bei Betrachtung von R_1 und R_2 wird ersichtlich, dass die Segmente sich bei der oben beschriebenen Einteilung teilweise überdecken. So haben die jeweiligen Segmente $\{R_1, R_2, R_3, D_1, D_2, D_5\}$ und $\{R_1, R_2, D_1, D_2, D_3, D_4\}$ die gemeinsame Schnittmenge $\{R_1, R_2, D_1, D_2, \}$. Das Segment des Routers R_3 kommt fast ohne Überlappungen aus. Lediglich das Vorhandensein der Router selbst sorgt dafür, dass es sich Geräte mit anderen Segmenten teilt: $\{R_1, R_3, D_6, D_7, D_8\}$.

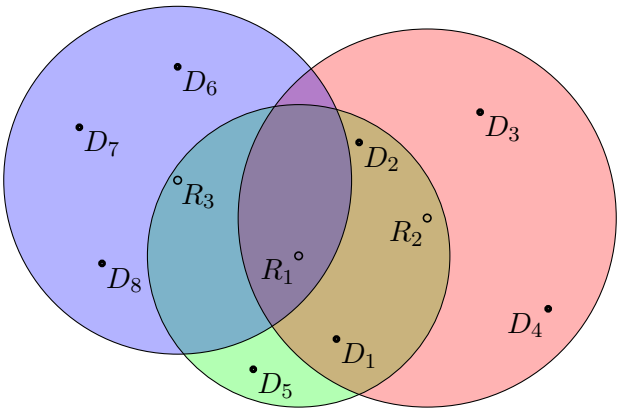


Abbildung 4.2: Beispiel für eine kabellose GAS-Konfiguration

Ein weiteres Problem, welches bei der Betrachtung der GAS in Abschnitt 2.3 auffällt, ist, dass Geräte nicht der eindeutigen Klassifizierung folgen, wie sie in Abschnitt 4.1.1 vorgeschrieben wurde. So können in kabellosen Systemen Geräte möglicherweise sowohl als Repeater als auch als Sensor oder Aktor dienen. Dieser Umstand wird so gelöst, dass die Klasse des Gerätes durch seine primäre Funktion bestimmt wird: Existiert ein Gerät, welches als Sensor und gleichzeitig als Repeater oder Router dient, so wird dies als Repeater oder Router dargestellt, da dieser möglicherweise sein Segment aufspannt.

4.1.4 Geräte in Gruppen

Das Konzept der Multicasts ist aus der Welt der IP-Netzwerke bereits bekannt. Durch solche werden 1:n Verbindungen zwischen Rechnern aufgebaut, um Nachrichten effizienter zu übermitteln. Gruppen in GAS haben eine ähnliche Funktion: Mit Hilfe dieser ist es möglich eine Menge von Geräten unter einer Adresse ansprechbar zu machen. Auch wenn Gruppen nicht von jedem GAS unterstützt werden, ist es sinnvoll, diese in einem Modell abzubilden, da sie viel über die Funktionalität des Systems verraten. In KNX beispielsweise ist es möglich, dass viel über Gruppen und nicht über individuelle Adressen kommuniziert wird. Werden Gruppen nicht dargestellt, so gehen an dieser Stelle viele Informationen über das Netzwerk verloren. AML verfügt bereits nativ über die Klasse *Group* (siehe Abschnitt 3.4.1), welche verschiedene Elemente miteinander assoziieren kann. Hier bietet es sich an, dieses Konzept auch für die Darstellung von Gruppen in GAS zu übernehmen.

Klar wird aufgrund der Eigenschaften von Gruppen, dass diese außerhalb der Einteilung des Netzwerks in Segmente dargestellt werden können. Gruppen sind als globale Eigenschaften eines GAS zu betrachten. Die konkrete Modellierung von Gruppen in diesem Konzept findet in Abschnitt 4.3.6 erneut Ansprache.

4.2 Qualitätskriterien

Zur konzeptuellen Betrachtung gehört auch die Formulierung von Qualitätskriterien, durch welche eine prototypische Implementierung bewertet werden kann. Informell ist die Qualität der Rekonstruktion einer Topologie durch ihre Nähe zum real vorliegenden System gegeben. Qualitätskriterien für das Verfahren werden anhand der folgenden Eigenschaften festgelegt:

- **Geräte**: Geräte in GAS können in Klassen, wie in Abschnitt 4.1.1 beschrieben, eingeteilt werden. Diese haben Eigenschaften wie Adressen und eine Klasse, welche deren Funktionsumfang beschreibt. Grundlegend ist von Interesse, ob Geräte gefunden werden und sie korrekt einer Klasse zugeordnet werden können, diese Zuordnung *zuverlässig* geschieht und deren weitere Eigenschaften vollständig erkannt werden können.

- **Topologie**: Die Topologie beschreibt die Netzwerkstruktur. Die logische Topologie beschreibt, welche Geräte miteinander verbunden und in welchen Netzwerksegmenten sie anzutreffen sind. Während vermaschte Topologien, wie sie in kabellosen Systemen anzutreffen sind, meist keine hierarchische Ordnung aufweisen, kann in KNX die hierarchische Ordnung eines Netzwerks durchaus betrachtet und bewertet werden.
- **Passiv-Aktiv-Verhältnis**: Mit diesem Verhältnis wird beschrieben, wie sich die Ergebnisse eines passiven Scans zu denen eines aktiven Scans vergleichen lassen. Im optimalen Fall wird dieses Verhältnis gegen 1 laufen. Dies würde bedeuten, dass sich durch einen aktiven Scan nicht mehr Informationen erhalten lassen, als durch einen passiven, nicht detektierbaren Scan. In der Realität ist jedoch zu erwarten, dass dieses Verhältnis, abhängig von dem betrachteten Netzwerk, weit unterhalb des optimalen Ergebnisses sein.
- **Ressourcenverbrauch**: Auch seitens der Berechnung der Topologierekonstruktion existieren Restriktionen. Die limitierenden Faktoren sind deren Laufzeit sowie die Größe der Logs auf dem Datenträger, welche zur Rekonstruktion in annehmbarer Qualität nötig sind.

Wichtig zu bemerken ist, dass die Qualität einer Rekonstruktion nur dann korrekt eingeschätzt werden kann, wenn ein vollständiger Plan des Feldbusses zum Vergleich vorliegt. Einzelne Qualitätskriterien, welche auf den oben beschriebenen Eigenschaften aufbauen, werden in den folgenden Abschnitten detaillierter beschrieben.

4.2.1 Anzahl der Geräte

Die Anzahl der erkannten Geräte ist eines der simpelsten Qualitätskriterien, welche sich aus der ursprünglichen Dokumentation und der Rekonstruktion ermitteln lassen. Das Verhältnis der Gesamtanzahl an erkannten zu den erwarteten Geräten $\frac{n_{rec}}{n_{doc}}$ ist gleichsam aber auch eines der aussagekräftigsten Mittel zur Beschreibung der Qualität einer Rekonstruktion. Bereits durch Betrachtung dieses Kriteriums können beispielsweise die Ergebnisse aktiver zu denen passiver Scanns verglichen werden. Eine hohe Anzahl an erkannten Geräten verrät unter Umständen auch viel über die vorliegende Topologie. So ist in einem KNX-Netzwerk in der Adressierung der Geräte bereits viel semantisches Wissen über dessen Topologie vorhanden.

4.2.2 Eindeutig erkannte Geräte

Wie bereits in 4.1.1 beschrieben, werden Geräte anhand ihrer Funktion in bestimmte Geräteklassen eingeteilt. Kann über die Telegramme ein Gerät nicht eindeutig zugeordnet werden, so soll diesem die Standardklasse *Device* zugeordnet werden. Sobald aus der Analyse hervorgeht, um welche Art von Gerät es sich genau handelt, muss diese Klasse einer der oben genannten Kategorien zugeordnet werden. Auf diese Weise kann bereits viel semantisches Wissen dargestellt werden.

Aus diesem Vorgehen ist ersichtlich, dass jedes Telegramm nach den an der Kommunikation beteiligten Adressen untersucht wird. Dies schließt bereits bekannte Geräte ein. Bekannte Geräte befinden sich in einem konstanten Zustand der Neuinterpretation und Informationen werden, wenn dies möglich ist, ergänzt. Ein Problem ergibt sich jedoch, wenn ein Gerät, welches bereits einer der oben genannten, speziellen Klasse zugeordnet wurde, nach der Analyse eines anderen Telegramms nun auch zu einer anderen Klassen gehören würde.

Die Häufigkeit solcher False-Positives ist ein weiteres Qualitätsmerkmal, welches die Zuverlässigkeit der unterliegenden Methodik zur Entscheidung der Klassenzugehörigkeit beschreibt. Es ist ein möglichst hoher Wert für das Verhältnis von True-Positives zu False-Positives $\frac{tp}{fp}$ anzustreben.

4.2.3 Vollständigkeit der Geräteeigenschaften

Die in Abschnitt 4.1 beschriebenen Geräteeigenschaften können auch zur Bewertung der Qualität herangezogen werden. Dafür wird jedes einzelne, gefundene Gerät auf die Vollständigkeit der ermittelten Eigenschaften überprüft. Erst nach einem vollständig ausgeführten Scan wird die globale Vollständigkeit nach $\sum_{i=0}^{N_{dev}-1} \frac{p_i}{P_i N_{dev}}$ bestimmt, wobei N_{dev} die Anzahl der gefundenen Geräte, p_i die Anzahl der gefundenen Eigenschaften des i-ten Geräts undP_i die erwartete Anzahl von Eigenschaften des i-ten Gerätes beschreibt.

4.2.4 Topologie

Die Topologie der vorgestellten GAS unterscheidet sich zum Teil stark. Während in kabellosen Systemen relative Einheit dadurch herrscht, dass in der Regel keine Restriktionen diesbezüglich vorgegeben sind, sieht dies in der verkabelten Welt anders aus. Systeme wie KNX schreiben eine bestimmte Netzwerkstruktur vor, welche in der Rekonstruktion entsprechend überprüft werden kann.

Die Rekonstruktion der Segmente ist in jedem Fall eindeutig – sei es die Unterteilung eines kabelgebundenen Systems in Linien oder die Zugehörigkeit eines kabellosen Gerätes zu einem Router oder Repeater. In funkbasierten Verfahren ist es jedoch möglich, dass die tatsächlich vorliegende Netzstruktur verknüpfter ist, als vorerst vorgesehen.

Ein Vergleich zwischen der originalen und der rekonstruierten Netzwerktopologie ist möglich. Hierfür müssen jedoch beide in die gleiche Struktur umgewandelt werden.

4.2.5 Passiv-Aktiv-Verhältnis

Eine der relevantesten Informationen ist, wie vollständig sich ein Netzwerk durch passives Scannen im Vergleich zu aktivem Scannen rekonstruieren lässt. Dies wäre in der Praxis besonders wichtig um einzuschätzen, wie hoch der Informationsgewinn eines passiven Angriffs auf ein Netzwerk wäre. Für die Beurteilung des Passiv-Aktiv-Verhältnisses wird auf das in 4.2.1 beschriebene Kriterium zurückgegriffen. Dieser Wert muss zuerst aus einer passiven Rekonstruktion ermittelt werden. Danach kann versucht werden, dieses Modell durch aktives Scannen zu vervollständigen. Der Wert $\frac{n_p}{n_a}$, wobei n_p die durch passive und n_a die durch aktive Scans ermittelte Anzahl an Geräten ist, beschreibt dann den relativen Informationsgewinn.

Dargestellt ist der Unterschied zwischen den passiv und aktiv erkennbaren Geräten in Abbildung 4.3. Hier wird von Gerät D_O aus ein Netzwerk beobachtet. Die gelb markierten Knoten sind diejenigen, welche sich durch einen passiven Scan beobachten lassen. Nachdem der passive Scan beendet wird, können aktiv weitere Geräte gesucht werden.

Diese sind hier rot dargestellt. Das Passiv-Aktiv-Verhältnis drückt den Unterschied zwischen den beiden Ergebnissen aus. Zu beachten ist, dass auch die gelb markierten Knoten aktiv hätten ermittelt werden können.

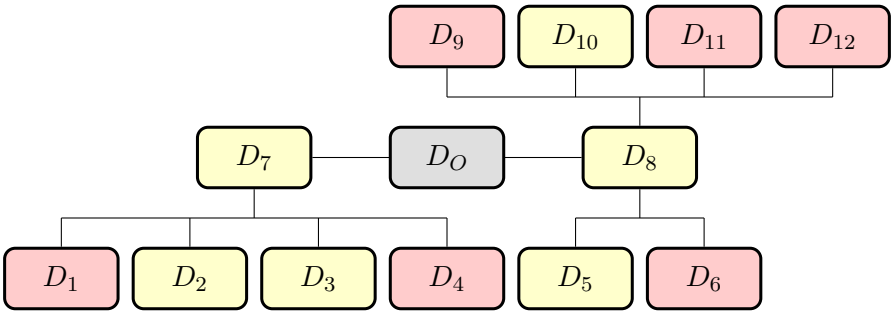


Abbildung 4.3: Passiv (gelb) und aktiv (rot) erkennbare Geräte; beobachtet von D_O (grau)

4.2.6 Benötigte Zeit

Zu den physischen Kriterien gehört die benötigte Zeit zur Rekonstruktion des Netzwerks. Diese kann anhand von zwei Verfahren zur Beurteilung der Qualität des Verfahrens genutzt werden. Einerseits ist es möglich eine Zeitvorgabe für die Laufzeit des Programms vorzugeben. Nach Ablauf dieser Zeit wird die Qualität des generierten Netzwerks anhand einer der vorhergehend beschriebenen Qualitätskriterien bewertet. Die Relation der vorgegebenen Zeit zu der daraus hervorgehenden Qualität des Ergebnisses spricht für oder gegen die praktische Anwendbarkeit des Verfahrens.

Weiterhin ist es möglich, eine bestimmte Qualität für ein Ergebnis vorzugeben. Das Verfahren wird daraufhin ausgeführt, bis die gewünschte Qualität erreicht ist. Offensichtlich ist es möglich, dass das Verfahren kein zufriedenstellendes Ergebnis erzeugen kann und somit nicht abbricht. Für dieses Fall muss die verfügbare Zeit beschränkt und das Ergebnis entsprechend gekennzeichnet werden.

4.2.7 Größe der Log-Dateien

Ähnlich wie die Kriterien bezüglich der Zeit kann auch die Größe der genutzten Traffic-Logs ein Qualitätskriterium für das Verfahren sein. Zum einen kann die

physische Größe der Datei auf dem Datenträger betrachtet werden. Ein Problem mit dieser Ansicht ist jedoch, dass diese durch Formatierung und Komprimierung beachtlichen Schwankungen unterliegen kann und damit viel an ihrer Aussagekraft verliert. Dieses Problem kann durch eine vorgegebene Formatierung umgangen werden.

Zum anderen kann die Anzahl der Einträge betrachtet werden. Diese ist von der Anzahl der aufgezeichneten Telegramme abhängig. So wird die Qualität des Ergebnisses direkt in ein Verhältnis zu den dazu benötigten Informationen gesetzt. Auch kann durch die Verwendung von Logs die zeitliche Komponente simuliert werden, wenn Timestamps entsprechend bei der Aufzeichnung mitgetragen wurden.

4.3 Konzipierung mit AutomationML

Bereits in Abschnitt 3.4 wurde festgestellt, dass sich durch das offene Datenformat AutomationML alle drei in Abschnitt 1.1 gezeigten Ansichten eine Netzwerkes darstellen lassen. Diese waren die *physische Topologie*, die *logische Topologie* und die *funktionale Topologie*. AML unterstützt die Formate COLLADA, *CAEX* und *PLCopen XML*, welche genutzt werden können, um die jeweiligen Ansichten eines Feldbusses darzustellen.

In Folge wird auf die Nutzung dieser Formate genauer eingegangen, die Darstellung der logischen Topologie jedoch am detailliertesten behandelt. Obwohl die anderen genannten Ansichten nicht als Teil der vorliegenden Arbeit angesehen werden, wird kurz aufgezeigt, welche Möglichkeiten sich durch die Nutzung dieser ergeben.

4.3.1 Logische Topologie

Bereits mit Hilfe der logischen Topologie lässt sich die Sicherheit eines Feldbusses einschätzen. Mit dieser kann dargestellt werden, inwiefern Geräte von einem bestimmten Punkt im Netzwerk aus erreichbar und somit auch angreifbar sind. Des Weiteren lassen sich hier viele Informationen über Geräte und das Netzwerk selbst darstellen, wie bereits in Abschnitt 4.1 aufgezeigt wurde. Durch eine ausführliche Analyse der logischen Topologie eines Netzwerks kann beispielsweise ermittelt werden, welche Bereiche

des Netzwerks besonders geschützt werden müssen [Gol18]. So kann in der logischen Topologie dargestellt werden, ob ein Gerät eine besonders sicherheitskritische Funktion erfüllt und damit besonderem Schutz unterliegen muss. Im Folgenden wird vor allem auf die logische Darstellung von kabellosen und kabelgebundenen Netzwerken, wie sie bereits in Abschnitt 2.3 gezeigt wurden, eingegangen.

KNX zum Beispiel weist eine stark hierarchische Struktur auf. Angefangen bei der Backbone-Linie gliedert sich das Netzwerk in Bereiche und Linien auf. Die Entfernung zwischen Bereichen und ihrer Platzierung in der KNX-Hierarchie kann dabei durch die Adressierung der Netzteilnehmer und die Betrachtung der Hop-Distanzen der Pakete eingeschätzt werden. Bereiche und Linien werden hier immer als Segmente dargestellt. Segmente, die Linien beschreiben, werden dabei in denjenigen Segmenten platziert, welche die Bereiche beschreiben. Des Weiteren werden alle Bereiche beim Vorhandensein einer Backbone-Linien, in einem weiteren Segment platziert, welches diese Linie repräsentiert. Dieses Verfahren soll in Kapitel 5 genauer beschrieben werden.

Weniger explizit ist die Topologie eines LonWorks-Netzwerks. Hier können Geräte beliebig vernetzt werden und es werden weniger Auflagen zur Struktur vorgegeben. Doch es ist auch hier ein Aufbau erkennbar, welcher sich durch eine hierarchische Segmentierung repräsentieren lässt: So existiert an oberster Stelle eine Unterteilung in verschieden LonWorks-Domains. Darauf folgt eine Zuweisung der Geräte zu Linien. Die unterliegende Topologie kann von Linien- bis Sternform zwar eine beliebige Form annehmen, eine Klassifizierung in hierarchisch angeordneten Segmenten ist jedoch trotzdem sinnvoll und aussagekräftig zur Analyse eines solchen Netzwerks.

Die vermaschten Netzstrukturen der kabellosen Standards unterliegen meist keiner so strikten Hierarchie. Einige GAS bieten die Möglichkeit, unterschiedliche Netzwerke des gleichen Typs über Gateways miteinander zu verbinden. In diesem Fall würde an dieser Stelle dann die erste Segmentierung vorgenommen werden. Generell werden Netzsegmente in kabellosen Systemen jedoch horizontaler angeordnet: Jeder Repeater und jeder Router spannt ein Segment auf, in welchem alle mit diesem verbundenen Geräte enthalten sind. Ein Resultat dieses Vorgehens ist, dass Geräte nicht mehr wie in kabelgebundenen GAS eindeutig in ein Segment zuzuordnen sind. Wie in Abschnitt 4.1.3 bereits angesprochen, ist dieser Umstand nicht alleine durch die Modellierung eines kabellosen Systems verursacht. Verbindungen zwischen Segmenten müssen in kabellosen Systemen in AML durch die Verbindung von Interfaces mit

<InternalLink> repräsentiert werden, da die logische Distanz zwischen Netzsegmenten aus dem Modell ansonsten nicht mehr ableitbar wäre.

Wie bereits des Öfteren erwähnt, werden die Segmente des Modells ineinander verschachtelt. Das oberste Element einer Instanzhierarchie soll hier jedoch immer das Element *Network* sein. In diesem können nicht nur verschiedene Segmente, sondern auch Gruppen platziert werden. Die Darstellung von Gruppen wird im Detail in Abschnitt 4.3.6 behandelt. Alle für das Modell definierten Geräteklassen, sowie die Klassen *Network* und *Segment* und die Definition der dort genutzten Attribute, können in Anhang A.1 und A.2 gefunden werden. In Abbildung 4.4 ist ein vereinfachtes Klassendiagramm der definierten Netzwerkklassen zu sehen.

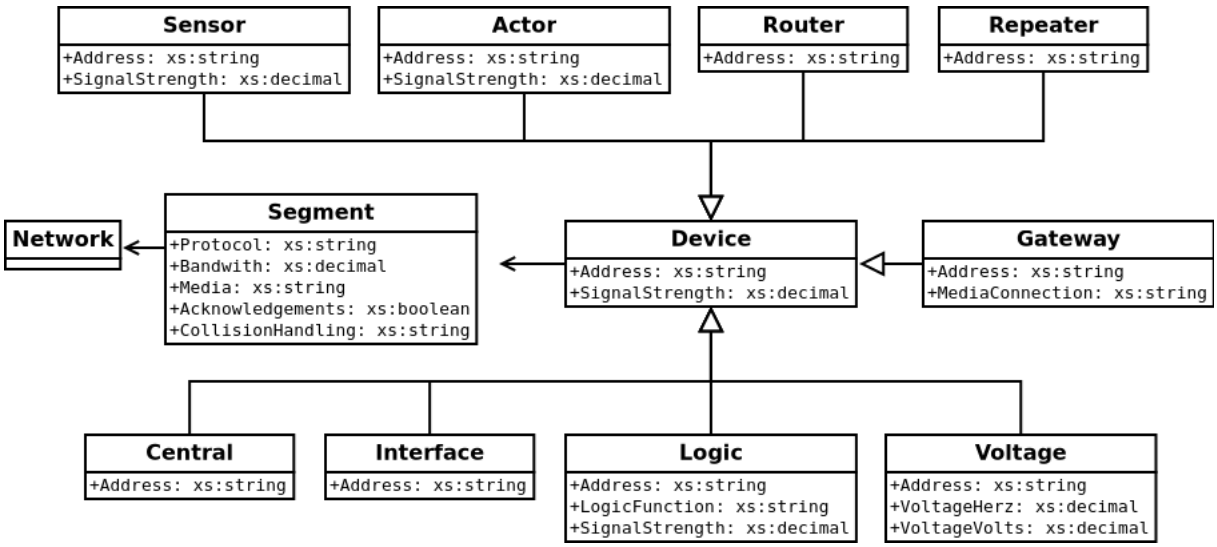


Abbildung 4.4: Vereinfachtes Klassendiagramm des Konzeptes (Details in Anhang A.1)

4.3.2 Physische Topologie

Auch die physische Topologie ist wichtig für die Einschätzung der Sicherheit eines Feldbusses. Diese beschreibt den Verlauf der Kabel und die Platzierung der Geräte. In AML wird für die Darstellung der physischen Eigenschaften einer Anlage das offene Datenformat COLLADA, beschrieben in Abschnitt 3.4.2, genutzt. COLLADA ist eine der

verbreitetsten Beschreibungssprachen von 3D-Inhalten und ein vielseitiges Werkzeug. Mit Hilfe dieses Formats ist es nicht nur möglich den Feldbus selbst, sondern auch weitere physische Eigenschaften des Gebäudes darzustellen. So ist vorstellbar, dass aus einer Analyse der physischen Topologie erkenntlich wird, mit welchen Anstrengungen es möglich ist, auf Geräte oder Kabel des GAS zuzugreifen – ob freier Zugriff darauf besteht, oder unter Umständen sogar Wände dafür aufgerissen werden müssen.

Auch weitere Einsatzmöglichkeiten durch die Nutzung von COLLADA sind denkbar: Mit Hilfe der physischen Informationen ist es möglich, in KNX-Netzwerken zu prüfen, ob aufgrund der Länge einer Linie bereits Linienkoppler installiert werden müssen. Für kabellose GAS kann durch die Position der Repeater und Router sowie die Eigenschaften von festen Objekten deren Funkreichweite und die Erreichbarkeit untereinander berechnet werden.

AML beschreibt auch, wie COLLADA-Objekte an Objekte der logischen Topologie gebunden werden. Hierfür wird eine von AutomationML vordefinierte Interfaceklasse namens `COLLADAInterface` genutzt. Eine genaue Betrachtung der Möglichkeiten für die Modellierung der physischen Topologie findet in dieser Arbeit keine Ansprache.

4.3.3 Funktionale Topologie: Reverse Engineering

In der funktionalen Topologie eines Feldbusses wird dargestellt, in welchen Beziehungen Geräte zueinander stehen. So wird beispielsweise abgebildet, welche Sensoren in welchen zeitlichen Abständen mit welchen Aktoren kommunizieren. Aus einer sicherheitstechnischen Betrachtung kann so ermittelt werden, welche Informationen wann verfügbar sind, welchen Effekt eine Unterdrückung der Kommunikation hätte, oder wie das Vorhandensein bestimmter Geräte vorgetäuscht werden könnte. AML unterstützt durch PLCopenXML, beschrieben in Abschnitt 3.4.3, eine Vielzahl an Darstellungsmöglichkeiten. Des Weiteren bietet es jedoch eine Vorschrift zur Anlagenplanung, welche durch ihre inverse Ausführung auch einen Plan aus einer bestehenden Anlage zurückgewonnen werden könnte.

In erster Linie dient AML in dieser Arbeit dafür, ein geeignetes Datenformat für die Darstellung der Feldbusse in physischer, funktionaler und besonders der logischen

Topologie bereitzustellen. Aus den Tabellen 3.1 und 3.2 geht jedoch ein weiterer wichtiger Aspekt hervor: Die dort gezeigten Abbildungen nach [Dra+08] beschreiben den gesamten Planungsprozess eines Produktionssystems und welche Beschreibungssprachen in einzelnen Entwicklungsschritten verwendet werden. Da im Fall dieser Arbeit jedoch kein System geplant, sondern Informationen aus einem existierenden System extrahiert werden sollen, müssen diese Schritte folglich wie in Tabelle 4.1 von hinten betrachtet werden.

Inbetr.-nahme	**HMI-Progr.**	**Roboter-Progr.**	**SPS-Progr.**	**Elektro-Planung**	**Mechanik-Planung**	**Anlagen-Planung**	**Produkt-Entwurf**
State Charts, SFC	Impuls-Diagr., Logik-Netzw., State Charts, SFC	Impuls-Diagr., Logik-Netzw., State Charts, SFC	Impuls-Diagr., Logik-Netzw., State Charts, SFC	Impuls-Diagr., Logik-Netzw., State Charts, SFC	Gantt Charts, PERT Charts, State Charts, SFC	Gantt Charts, PERT Charts	Gantt Charts

Tabelle 4.1: Schritte des Reverse Engineering einer Anlagenplanung nach Abb. 3.1

Tabelle 4.1 zeigt im Wesentlichen die gleichen Informationen wie Tabelle 3.1, ist jedoch in ihrer Ausführungsreihenfolge invertiert. Das Reverse Engineering einer fertigen Anlage beginnt hier bei dessen Inbetriebnahme und versucht daraufhin die Folgeschritte so detailreich wie möglich nachzuvollziehen. Dies beschränkt die Wahl der geeigneten Beschreibungssprachen auf *State Charts* sowie *Sequential Function Charts*. Bei Betrachtung von Tabelle 3.2 wird auch ersichtlich, dass die für GAS relevanten Aspekte die Steuerungsbeschreibung und die Verhaltensbeschreibung der Anlage sind. Auch hier werden *State Charts* sowie *Sequential Function Charts* genutzt, um das Verhalten zu beschreiben.

Durch die Nutzung von PLCopen XML unterstützt AML eine Vielzahl an Beschreibungsmitteln um eine funktionale Topologie darzustellen. Die Planung einer Anlage wird in [Dra+08] klar dargestellt und sieht bestimmte Sprachen zur Darstellung einzelner Teilaspekte vor. Wenn AML auch zum Reverse Engineering genutzt wird, ist es sinnvoll, die klar vorgegebenen Beschreibungsschritte zurückzuverfolgen. Aus diesem Grund sollte bei der Beschreibung der funktionalen Topologie in Betracht gezogen werden, *State Charts* oder *Sequential Function Charts* zu nutzen. Dies wird in der vorliegenden Arbeit jedoch nicht im Detail behandelt.

4.3.4 Gerätedefinition

Der Standard von AML definiert eine Vielzahl von Geräteklassen, mit welchen eine Anlage geplant werden kann. Diese vordefinierten Klassen genügen jedoch nicht, um Feldbusse und GAS adäquat abzubilden. Aus diesem Grund müssen eigene Klassen, Interfaces und Attribute erstellt werden, welche sich für die Modellierung von GAS besser eignen.

In Anhang A.1 werden alle die Klassen gezeigt, welche im Rahmen dieser Arbeit erzeugt wurden und zur Modellierung von Feldbussen dienen. Bei der Wahl der Eigenschaften wurde sich an den Auflistungen in Abschnitt 4.1 und 4.1.1 orientiert. Die Darstellung der Klassen selbst folgt dem Beispiel des offiziellen AML-Standards [Aut18; Aut14; Aut17a; Aut17b].

4.3.5 Attributdefinitionen in AutomationML

Auch konkrete Attributklassen werden in AML definiert. Auf diese beziehen sich die in Abschnitt 4.3.4 gezeigten Geräteklassen. Deren konkrete Definition ist in Anhang A.2 gezeigt. Auch diese ist in ihrer Form stark an die Vorgaben von [Aut18] angelehnt worden.

4.3.6 Gruppen

Vorhergehend wurden in AML Geräteklassen und deren Attribute definiert. Was bisher jedoch kaum Ansprache fand, ist die Darstellung von Gruppenadressen. Für diese wird ein in [Aut18] definiertes AML-Objekt namens *Group* genutzt, welches bereits in früheren Kapiteln, wie etwa in Abschnitt 3.4.1, erwähnt wurde.

In AML werden konkrete Anlageninformationen in der sogenannten Instanzhierarchie modelliert. In dieser werden CAEX-Elemente, vorwiegend die sogenannten `InternalElements`, zur Darstellung von Objekten hierarchisch untereinander angeordnet, um so komplexe Systeme zu modellieren. Für einige Einsatzszenarien ist es

jedoch sinnvoll, verschiedene Objekte in einen semantischen Zusammenhang zueinander zu stellen. Das kann dies beispielsweise bedeuten, dass alle Glühbirnen-Objekte eines Gebäudes für die einfachere Wartbarkeit miteinander assoziiert werden. Für das hier vorgestellte Konzept werden Gruppen genutzt, um Geräte eines Multicasts zusammen darstellen zu können.

In AML-Gruppen dürfen keine Instanzinformationen vorkommen. Dies bedeutet, dass der hierarchische Aufbau einer Anlage dort nicht beschrieben werden darf. Auch kommen in einer AML-Gruppe keine neuen Objekte vor, sondern lediglich gespiegelte Objekte von in der Instanzhierarchie vorkommenden `InternalElements`. Diese gespiegelten Objekte verweisen dabei mittels eines Attributes namens `RefBaseSystemUnitPath` auf ihre konkrete Instanz.

Für dieses Konzept sollen neue AML-Gruppen definiert werden, sobald eine Gruppenadresse bekannt ist. Das Attribut `Name` einer solchen Gruppe soll die Gruppenadresse sein und ihre Mitglieder werden unter dem Namen der gespiegelten Instanz in ihr gespeichert. Nach Möglichkeit sollen Gruppen direkt unter dem Element `Network`, definiert in Anhang A.3, des konkreten Modells dargestellt werden. Ein Beispiel für eine Gruppe in CAEX-Format ist am Endes des Anhangs A.3 zu sehen.

4.4 Datenquellen

Nach der Definition des Modells muss gezeigt werden, wie dieses gefüllt werden kann. Die vorgestellten Datenquellen weichen in ihrer Qualität und Vollständigkeit stark voneinander ab, sind jedoch nötig um, eine Netzwerktopologie zu rekonstruieren.

Der Zustand eines betrachteten Systems muss möglichst vollständig bekannt sein, um eine Rekonstruktion nach den Kriterien in Abschnitt 4.2 bewerten zu können. Erst sobald festgestellt wird, dass der rekonstruierende Algorithmus Ergebnisse hinreichender Qualität erzeugt, kann die Rekonstruktion der Topologie selbst zuverlässig zur Bewertung genutzt werden. Auch ist es wichtig sicherzustellen, dass die existierende Topologie zum Zeitpunkt der Erzeugung der Rekonstruktion äquivalent zur Vergleichstopologie ist, da nur gleiche Ausgangstopologien nach den Kriterien in Abschnitt 4.2 miteinander verglichen werden können.

Die Erzeugung dieser Ausgangstopologie ist enorm wichtig. Aus diesem Grund werden Verfahren benötigt, um eine ursprüngliche, dokumententreue Topologie zu erstellen. Hierfür werden im Folgenden einige Möglichkeiten vorgestellt.

4.4.1 Menschliches Wissen

Wo Systeme im Einsatz sind, existieren meist auch Menschen mit Wissen über diese. So ist eine triviale Quelle für Anlageninformationen die Befragung der am Netzwerk arbeitenden Administratoren. Dieses Wissen besteht in keiner konkreten Form und müsste auf jeden Fall in eine solche übersetzt werden.

Vollständigkeit

Hier kann hier meist nicht nachvollzogen werden, wie aktuell oder vollständig die vorliegenden Informationen sind. Es ist zu erwarten, dass menschliches Wissen fragiler ist, als die vorerst beschriebene Dokumentenerosion. So können beispielsweise Details über das vorliegende Netzwerk nicht nur vergessen werden, sondern gänzlich falsch wiedergegeben werden. Werden große Netzwerke mit vielen tausend Geräten betrachtet, so ist nicht zu erwarten, dass durch eine reine Befragung der zuständigen Personen eine Topologie rekonstruierbar ist.

4.4.2 Log-Dateien

Um verschiedene Zustände eines GAS einfach festzuhalten ist es sinnvoll, in zeitlichen Abständen den Traffic eines Feldbusses aufzuzeichnen. So kann – ohne jegliche Planungsdokumente in AML zu übersetzen – der Zustand eines Gebäudes implizit festgehalten werden. Je umfangreicher die Aufzeichnung ausfällt, desto aussagekräftiger ist die Momentaufnahme. Bei sehr langen Aufnahmen wird unter Umständen aber weniger der *Moment*, als das sich verändernde GAS festgehalten.

Diese aufgezeichneten Logs können dann zu jedem Zeitpunkt von dem hier vorgestellten Konzept, oder einer möglichen Weiterentwicklung dessen, zu rekonstruierten Topologien umgewandelt werden. Bei ausreichender Qualität der Rekonstruktion hat dies den Effekt, dass die händische Erzeugung einer Topologie im Format von AML – wie vorgeschlagen in Abschnitt 4.4.3 – nicht vorgenommen werden muss. Der für diese Technik benötigte Aufwand ist um ein Vielfaches niedriger als bisher vorgeschlagene Verfahren und macht die Verwendung von Log-Dateien zur Dokumentation von Feldbustopologien in erster Instanz zu der effektivsten möglichen Datenquelle.

Vollständigkeit

In Bezug auf die Vollständigkeit der dargestellten Informationen haben Log-Dateien ein enormes Potential. Diese können jederzeit erzeugt werden, was in jedem Fall für deren Aktualität spricht. Weiterhin ist zu unterscheiden, ob diese durch aktives oder durch passives Scannen erzeugt wurden. Abhängig von der Konfiguration des Netzwerks und der Dauer der Aufnahme werden bei passivem Scannen weniger Informationen als im aktiven Scannen gewonnen. Einige Geräte sind unter Umständen so konfiguriert, dass sie kaum oder gar keine Datenpakete versenden. Diese sind dann durch einen passiven Scan nicht ermittelbar.

Durch aktives Scannen können zusätzlich auch Geräte angesprochen werden, welche dem Verfahren bis dato unbekannt sind. In GAS wie KNX ist der Adressbereich der Geräte bekannt und noch klein genug um durch Probing weitere Informationen zu gewinnen. Bei der Verwendung des Hop-Zählers 7 werden KNX-Telegramme von Geräten nicht vermindert, was für weitreichende Scans genutzt werden kann. Solche Möglichkeiten sind jedoch nicht zwangsweise in allen GAS verfügbar.

Ein Problem von langen Aufnahmen ist, dass über den Zeitraum der Aufnahme die Topologie des GAS geändert werden kann. Dies würde zu Artefakten wie doppelt auftretenden oder neu hinzugefügten Geräten führen. Das Zeitfenster der Aufnahme ist also auch im Hinblick auf mögliche Modifikationen des GAS zu wählen.

4.4.3 Planungsunterlagen

Jedes Gebäude wird auf Grundlage von Planungsdokumenten erbaut. Diese Planungsunterlagen sind eine wichtige Informationsquelle, um die Ergebnisse des vorgestellten Verfahrens zu ermitteln.

Eine solche Dokumentation kann auf mehrere Arten genutzt werden. Zum einen ist es bei kleineren Netzwerken möglich, die Auswertung einer von dem vorgestellten Konzept erzeugte Topologie händisch mit den Planungsunterlagen zu vergleichen. Hierfür würde bereits eines der einfacheren, in Abschnitt 4.2 erwähnten Kriterien, wie der Vergleich anhand der Anzahl der gefundenen Geräte, genügen. Für einen ausführlichen Vergleich sollte jedoch die Möglichkeit herangezogen werden, die ursprüngliche Dokumentation in ein zu diesem Konzept kompatibles Format zu übersetzen, damit auch kompliziertere Untersuchungen maschinell ausgeführt werden können.

Zu beachten ist aber auch, dass solche Planungsunterlagen in der Regel sehr komplex und für Laien mitunter unverständlich sind. Ein Beispiel für solche Planungsunterlagen ist in Anhang A.4 zu sehen. Die händische Übersetzung eines solchen Plans ist für komplexe Anlagen unter Umständen keine Option.

Wenn diese Informationen jedoch in maschinenlesbarer Form existieren, dann können Planungsunterlagen zumindest einfach ausgewertet werden. In Listing 4.1 zum Beispiel ist ein Ausschnitt aus einer Datei mit Konfigurationsdaten abgebildet. Das hier genutzte JSON-Format macht dessen Auswertung relativ einfach.

```
{
  ...
  "devices": {
    "P-0421-0_DI-5": {
      "address": 0,
      "name": "",
      "location": "P-0421-0_BP-2"
    },
    "P-0421-0_DI-3": {
      "address": 1,
      "name": "",
      "send": [
        "2306",
        "2305"
      ],
      "location": "P-0421-0_BP-2"
    },
    ...
  },
  ...
}
```

Listing 4.1: Ausschnitt aus Planungsdaten

Vollständigkeit

In Bezug auf die Vollständigkeit der Dokumentation kann sich nur sagen lassen, dass diese von der in Abschnitt 1.1 beschriebenen Dokumentenerosion abhängig ist – alternde Dokumente haben also eine Tendenz zur Unvollständigkeit. Die gewonnen Informationen sind somit meist nicht immer verlässlich.

4.4.4 Konfigurationstools

Viele GAS bieten Programmierwerkzeuge für deren Konfiguration an. Im Falle von KNX ist dies ETS – die *Engineering Tool Software*, entwickelt von der KNX Association (Bsp. in Abb. 2.2). Diese wird hauptsächlich von Administratoren genutzt, um KNX-Netzwerke zu entwerfen. Jedoch ist es damit auch möglich, den aktuellen Zustand eines solchen Netzwerks abzufragen. Wird diese Möglichkeit genutzt, so kann im Falle von KNX eine vollständige Konfiguration erzeugt werden, welche dann in das kompatible AML-Format übersetzt werden muss. Diese würde sich am Besten für die Beurteilung des vorgeschlagenen Verfahrens eignen.

Ein Problem ist jedoch, dass das von ETS genutzte Datenformat, genau wie die Software selbst, nicht offen ist. Dies unterscheidet es von Kandidaten wie LonWorks, welche ein offenes System bereitstellen. Deshalb muss entweder mittels Reverse Engineering die Informationsstruktur aus ETS gewonnen werden oder die Topologie händisch aus der GUI extrahiert werden.

Vollständigkeit

Bei der Nutzung von Konfigurationstools kann damit gerechnet werden, dass die aufgezeichneten Daten im Vergleich zu den anderen Datenquellen am vollständigsten sind. Das Problem ist jedoch, dass nicht jedes GAS zwangsweise Konfigurationstools bereitstellt, welche auch über die Möglichkeit der vollständigen Abfrage verfügen. In jedem Fall eignet sich der Einsatz von ETS für KNX jedoch am besten, um das vorgestellte Verfahren mittels der in Abschnitt 4.2 vorgeschlagenen Kriterien zu bewerten.

Kapitel 5

Topologieerkennung am Beispiel von KNX

In diesem Kapitel

Wie bereits erwähnt, soll die automatische Topologieerkennung am speziellen Einsatzszenario von KNX erprobt werden. KNX, beschrieben in Abschnitt 2.3.1, ist ein kabelgebundener Standard, welcher seine Geräte in eine hierarchische Ordnung von Bereichen und Linien unterteilt. Anhand dieser klaren Unterteilung lässt sich das Netzwerk praktisch in Segmente einteilen. Dieses Vorgehen wird in diesem Kapitel beispielhaft aufgezeigt.

In Abschnitt 4.4 wurden viele mögliche Datenquellen für die Abbildung in AML aufgezeigt. Das aktive sowie das passive Scannen wird jedoch als der relevanteste Beitrag aufgefasst und entsprechend im Detail erklärt. Hierfür wird im Folgenden gezeigt, wie das passive und aktive Scannen eines KNX-Netzwerkes vonstatten gehen kann. Im Vorfeld wird kurz eine Erläuterung zu relevanten Aspekten des Telegrammformats

Common External Message Interfaces (cEMI) gegeben, da mit Hilfe von diesem auf dem für den Prototypen genutzten KNX-IP-Interface kommuniziert wird. Im Anschluss wird gezeigt, wie sich mit Hilfe des Scannens Informationen aus einem solchen Netzwerk gewinnen lassen können.

5.1 cEMI – Common External Message Interface

cEMI wird als Nachrichtenformat in KNX-Netzwerken genutzt. Aus diesem Grund ist es wichtig, dieses ausreichend zu verstehen, da nur so die an einer Kommunikation beteiligten Geräte korrekt erkannt und ihrer Aufgabe nach einer Klasse zugeordnet werden können. Zudem sind die über dieses Format übermittelten Informationen die einzigen, welche sich durch reines Scannen beobachten lassen. Der Aufbau von cEMI wird in Tabelle 5.1 untersucht.

#	Bytes	Bedeutung
1	**29**	**Message Code:** 29 = L_DATA.ind – Data Request
2	**00**	**Add. Info Length:** 0 = keine zusätzlichen Informationen.
3	**BC**	**Ctrl 1:** Kontrollfeld 1; Erklärung folgt
4	**E0**	**Ctrl 2:** Kontrollfeld 2; Erklärung folgt
5	**36**	**Src. Address:** Nibble 1 für Bereich (3), Nibble 2 für Linie (6),
6	**19**	Byte 2 für Gerät (25); Adresse = 3.6.25
7	**12**	**Dst. Address:** Nibble 1 für Bereich (1), Nibble 2 für Linie (2),
8	**81**	Byte 2 für Gerät (129); ist nach Ctrl 2 Gruppe = 1/2/129
9	**00**	**Data Length:** Länge der Nutzdaten nach Byte 11 = 0 Byte
10	**00**	**ADPU:** TPCI (Bit 1 bis 6), APCI (Bit 7 und 8)
11	**81**	Fortsetzung APCI

Tabelle 5.1: Aufbau des cEMI-Formats nach [DEH]

Der **Message Code** kann eine von drei Formen annehmen, welche den Typ des Datenpaketes beschreibt. Im Feld **Additional Info Length** kann angegeben werden, ob nach Byte 2 zusätzliche Informationen folgen. Eine große semantische Bedeutung besitzen die **Kontrollfelder 1 und 2** in Byte 3 und 4. Hier werden in den einzelnen Bits folgende Informationen kodiert [DEH]:

- **Bit 1:** Angabe des Typs des Frames – erweitert oder nicht.
- **Bit 2:** *Reserviert*.
- **Bit 3:** Gibt an, ob das Telegramm im Falle eines Fehlers erneut gesendet wird.
- **Bit 4:** Angabe, ob Broadcast, oder nicht.
- **Bit 5-6:** Priorität des Telegramms: System (0), normal (1), dringend (2), gering (3).
- **Bit 7:** Aufforderung, Acknowledgements zu senden.
- **Bit 8:** Kein Fehler (0, Confirm) oder error (1, L_Data.con).
- **Bit 9:** Gibt an, ob die Zieladresse eine individuelle, oder eine Gruppenadresse ist.
- **Bit 10-12:** Hop Count (7x0 wird nie verringert).
- **Bit 13-16:** Format des erweiterten Frames; sonst 0

In Byte 5 bis 8 werden jeweils die **Quelladresse** und die **Zieladresse** angegeben, wobei die Zieladresse möglicherweise eine Gruppenadresse ist. **Data Length** gibt an, wie viele Bytes an Nutzdaten nach dem ADPU-Teil des Frames folgen. **ADPU** ist ein weiterer komplexer Teil des Telegramms. Dieser wird aufgeteilt in *TPCI* (Bit 1-6) und *APCI* (Bit 6-16). Im Bereich des TPCI kann angegeben werden, ob es sich um ein nummeriertes oder nicht-nummeriertes Paket handelt, sowie eine Sequenznummer von 0 bis 15 kodiert werden. Auch wird hier festgelegt, ob es sich um Kontrolldaten handelt [sol]. Der Bereich APCI bietet dank seiner Größe die Möglichkeit der ausführlicheren Beschreibung des Telegramms. Hier steht unter anderem, ob Werte gesendet oder angefordert werden, physische Adressen gesetzt oder angefordert werden, Speicherinhalte angefordert oder gesendet werden, sowie Zugriffsrechte verwaltet werden. Weiterhin existiert eine Vielzahl an weiteren Befehlen, welche hier jedoch keine Ansprache finden werden.

Bei der Betrachtung des cEMI-Formates wird jedoch klar, dass es eine Vielzahl an Informationen über die beteiligten Kommunikationspartner verrät. Neben Quell- und Zieladresse, welche selbst bereits Auskunft über die Existenz von mindestens einem Gerät geben, wird im Bereich des TPCI und auch in den möglichen Nutzdaten ab Byte 11 viel über die Klasse der in der Kommunikation beteiligten Geräte verraten. Ein Ansatz zur Klassifizierung der Kommunikationsgeräte muss folglich mit einer Vielzahl an statistischen Informationen über die Kommunikation arbeiten. Im einfachsten Fall kann der Traffic zwischen bereits bekannten Knoten aufgezeichnet werden und mit dem zwischen unbekannten Teilnehmern verglichen werden, um diese korrekt ihrer Funktion zuzuordnen.

5.2 Passives Scannen

Passives Scannen ist das meist nicht-detektierbare Abhören des Netzwerkverkehrs. Hier werden an einer oder mehreren Stellen des Netzwerkes mit Hilfe von entsprechenden Interfaces Datenpakete mitgeschnitten und zur späteren Auswertung gespeichert oder aber direkt ausgewertet. Das passive Scannen ist hier die wichtigste behandelte Funktion. Das liegt vor allem daran, dass es vielseitig einsetzbar ist. Einige Vorteile des passiven Scannens sind:

- Die Nutzung zur **Konservierung eines Netzwerkzustands**: Diese Möglichkeit wurde bereits in Abschnitt 4.4.2 angesprochen. Wo regulär nur vollständige Systemzustände anhand von Planungsunterlagen erhalten werden, ist es mit Hilfe der automatischen Rekonstruktion von Typologien auch implizit möglich, Netzwerkzustände durch die Verwendung von Traffic-Logs zu erzeugen. Ein solches Verfahren ist bei gesicherter Qualität der Rekonstruktion im hohen Maße flexibel, besitzt jedoch auch die in Abschnitt 4.4.2 aufgezählten Nachteile.
- Die einfache **Erweiterbarkeit**: Gerade in Bezug auf die Implementierung des Konzeptes ist es wichtig, schnell am praktischen Einsatzszenario testen zu können. An dieser Stelle muss ein fertiggestellter, passiver Scanner nur um eine Komponente ergänzt werden, welche die weiterhin zu prüfenden Geräte eigenständig ermittelt und Telegramme versendet. Die Gerät-erkennende Logik wird durch den passiven Scanner bereitgestellt und die aktive Komponente ist somit eine Erweiterung dafür.
- Die **Detektierbarkeit**: Gerade aus Betrachtungen der Netzwerksicherheit ist es wichtig zu wissen, welche Informationen durch einen Netzwerkteilnehmer in Erfahrung gebracht werden können, *ohne* dass dieser durch sein direktes Eingreifen aufspürbar ist. Passives Scannen ist in den meisten Fällen durch eine reine Analyse des Netzwerkverkehr *nicht* detektierbar und auch aus diesem Grund von weiterem Interesse für die Implementierung.

In Tabelle 5.2 wird ein möglicher Ausschnitt aus einer Log-Datei betrachtet, welche durch passives Scannen erzeugt wurde. Für die folgenden Beispiele gilt auch, dass die Adresse des aufzeichnenden Interfaces `3.5.47` ist.

Timestamp	**cEMI**
…	…
1572609600	29 00 BC 60 35 03 35 52 00 00 81
1575201600	29 00 BC E0 35 B2 35 78 00 00 81
1577880000	29 00 BC E0 35 C9 35 78 00 00 81
…	…

Tabelle 5.2: Ein Beispiel für aufgezeichnete Datenpakete; Farbgebung nach Tabelle 5.1

In Tabelle 5.2 werden die cEMI-Nachrichten von drei Einträgen betrachtet. Die Quell- und Ziel-Adressfelder der Telegramme geben bereits Aufschluss über Geräte. So sind die Quellen 3503, 35B2 und 35C9 beobachtet worden. Diese lassen sich in die Geräteadressen `3.5.3`, `3.5.178` und `3.5.201` übersetzen. Zu den Zieladressen gehören 3552 und 3578, wobei die der letzten Telegramme durch Kontrollfeld 2 (E0) als Gruppenadresse deklariert wurde. So ergeben sich folgende Adressen: `3.5.82` und `3/5/120`.

Die fünf Adressen, welche sich durch die passive Aufnahme, dargestellt in Tabelle 5.2, haben beobachten lassen, sind `3.5.3`, `3.5.82`, `3.5.178`, `3.5.201` sowie die Gruppenadresse `3/5/120`. Dazu kommt die des Beobachters, `3.5.47`. Alle vorkommenden Adressen haben nach der Beschreibung in Abschnitt 2.3.1 die Eigenschaft, dass sie sich im gleichen Bereich und in der gleichen Linie befinden. Aus diesem Grund werden alle Geräte nach der Beschreibung in Abschnitt 4.3.1 in ein Segment eingeordnet. Gruppen werden, wie in Abschnitt 4.3.6 beschrieben unter dem Wurzelelement *Network* angeordnet. Die Teilnehmer der Gruppe `3/5/120` sind für dieses Beispiel als Platzhalter angebracht, da diese aus den Telegrammen in 5.2 nicht hervorgehen. Zu sehen ist das aus diesen Informationen resultierende Modell in Abbildung 5.1.

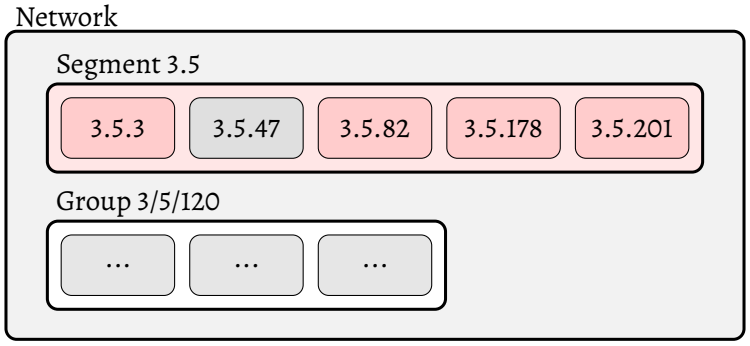


Abbildung 5.1: Beispiel der Segmentierung einer KNX-Linie

Als nächstes wird der Fall betrachtet, in dem Adressen außerhalb des eigenen Segments beobachtet werden. Hierfür sollen drei weitere Geräte aus Segment `3.7` aufgezeichnet werden, welche mit `3.7.12`, `3.7.33` und `3.7.78` adressiert werden. Die neue Segmentaufteilung ist in Abbildung 5.2 abgebildet.

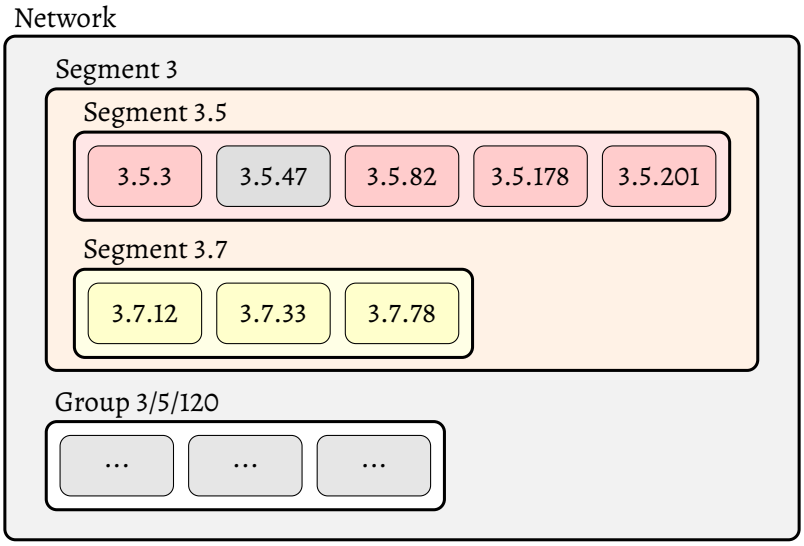


Abbildung 5.2: Beispiel der Segmentierung eines KNX-Bereichs

In Abbildung 5.2 wurde nun nicht nur ein neues Segment, sondern es wurden zwei neue Segmente hinzugefügt. Segment `3.5` und `3.7` stehen dabei für die KNX-Linien `3.5` und `3.7` sowie Segment 3 für Bereich 3 steht. Diese Einteilung ist hier möglich, da die Hierarchie eines KNX-Netzwerkes mit Linien und Bereichen bekannt ist. Wie das Modell nach dem Hinzufügen von weiteren Adressen aufgebaut ist, ist in Abbildung 5.3 gezeigt.

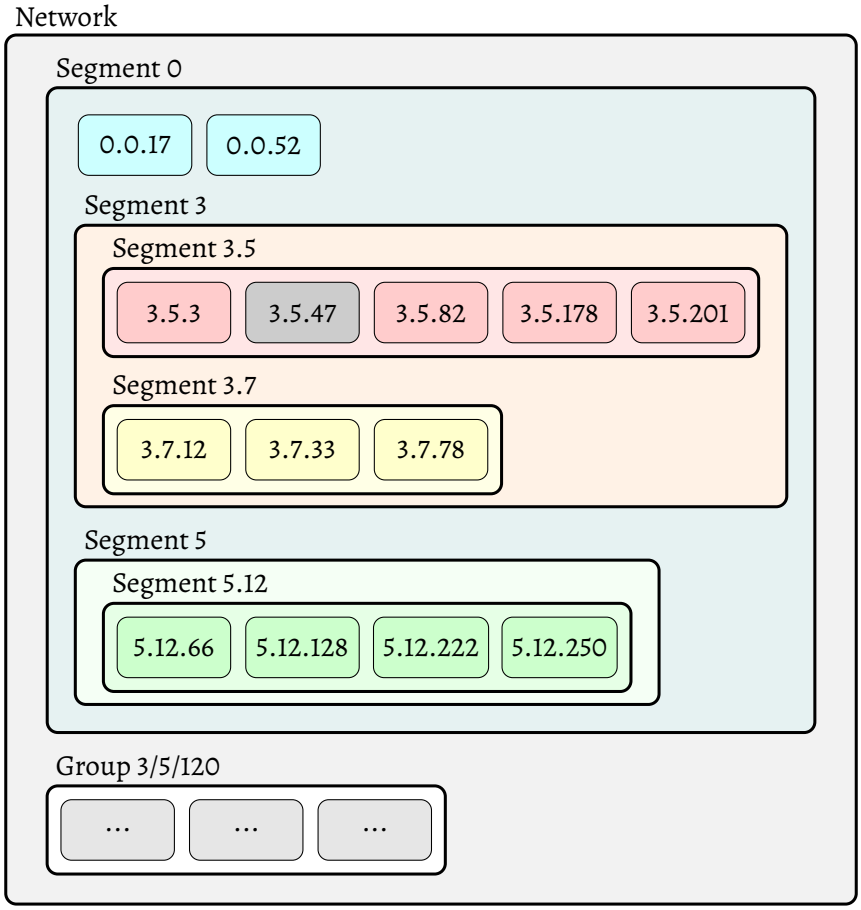


Abbildung 5.3: Beispiel der Segmentierung zweier KNX-Bereiche

Abbildung 5.3 zeigt, wie gleich zwei neue Segmente dem Modell hinzugefügt wurden: Die Backbone-Linie `0` und Linie `5.12` – wobei zweitere implizit Bereich `5` mit einschließt. Weiterhin werden, wie in Abschnitt 2.3.1 beschrieben, alle Bereiche durch die Backbone-Linie verknüpft. Diese wird natürlich auch als Segment im Modell dargestellt.

Diese Abbildung soll den maximalen Ausbau des Beispielnetzwerks darstellen und gibt einen Überblick über den Aufbau des unterliegenden CAEX-Dokumentes. Die dazugehörige Datei ist in Anhang A.3 zu finden. Der dort gezeigte CAEX-Code wurde zur besseren Übersicht um oft wiederkehrende Deklarierungen gekürzt und analog zu Abbildung 5.3 farbig gekennzeichnet.

5.3 Aktives Scannen

Wie bereits in Abschnitt 5.2 erwähnt, wäre durch die Implementierung eines passiven Scanners bereits die Grundlage für einen aktiven Scanner geschaffen. Dessen Aufgabe würde außerdem daraus bestehen, Kandidaten für das aktive Probing zu ermitteln und dieses durchzuführen. Hierzu würde der Adressbereich des jeweiligen GAS ohne die bereits bekannten Adressen betrachtet werden.

Der hieraus entstehende, zusätzliche Informationsgewinn fällt abhängig vom betrachteten Netzwerk hoch oder niedrig aus. Einige Systeme sind so konfiguriert, dass deren Geräte regelmäßig senden, während in andere Geräte möglicherweise über sehr große Zeiträume nicht aktiv werden. In den letzteren ist ein hoher zusätzlicher Informationsgewinn zu erwarten. Grundsätzlich folgt der Informationsgewinn durch zusätzliches, aktives Scannen dem Beispiel aus Abbildung 4.3.

Kapitel 6

Implementierung des Konzeptes

In diesem Kapitel

Nachdem in den vorhergehenden Kapiteln Grundlagen und Konzepte für die Realisierung der logischen Topologieerkennung vorgestellt wurden, folgt nun die Implementierung eines entsprechenden Prototypen. Dieser nutzt das passive Scannen, welches vor allem in Abschnitt 5.2 behandelt wurde. Als Datenquelle dienen dabei Traffic-Logs, welche zur Erprobung für diese Arbeit bereitgestellt wurden.

Zur Implementierung werden *Python 3.8.2* sowie weitere, dazugehörige Bibliotheken genutzt. Diese werden in Abschnitt 6.1 erläutert. Ziel des Prototyps ist, eine Netzwerktopologie als valide CAEX-Datei zu rekonstruieren. Diese soll im AML-Editor (siehe Abb. 3.3) angezeigt werden können.

6.1 Voraussetzungen

Wie bereits erwähnt, wird der Prototyp in **Python 3.8.2** implementiert. Dabei gab es mehrere Gründe, Python als Programmiersprache zu wählen:

- **Unterstützung**: Python ist eine der am meisten genutzten Programmiersprachen der Welt [SE19] und wird demzufolge auch sehr gut mit Bibliotheken unterstützt – die Grundinstallation und Package-Manager wie *pip* stellen Module für die meisten Einsatzzwecke bereit. Auch existiert eine hilfsbereite Online-Community.
- **Leistung**: Abgesehen von den Bibliotheken der Programmiersprache C bietet Python keine überragende Performance [Fou]. Da die Auswertung von Logs und die Generierung von Topologien jedoch keine solch anspruchsvollen Einsatzzweck ist, bietet es sich an, eine solche Scriptsprache für eine Anwendung zu nutzen.
- **Nutzung am Lehrstuhl** und **Erfahrung**: Weitere Vorteile sind die eigene Programmiererfahrung mit Python sowie die bereits vorhandene Nutzung am Lehrstuhl dieser Arbeit. Hier macht es Sinn, keine neuen, komplexen Tools zu etablieren, wenn dies nicht nötig ist.

Für das Starten der Anwendung und die Initialisierung der virtuellen Umgebung wird **bash 5.0** genutzt. Da dies nur einen marginalen Teil der Implementierung darstellt, wird darauf in Folge jedoch nicht mehr eingegangen.

Weite genutzte Python-Bibliotheken sind **pandas**, **datetime** und **uuid**. Diese werden jeweils genutzt, um die Log-Dateien einzulesen und zu verarbeiten, die auszugebende CAEX-Datei mit dem aktuellen Datum zu versehen und UUIDs für einzelne `InternalElements` zu vergeben.

6.2 Aufbau des Prototyps

Der implementierte Prototyp ist in verschiedene Funktionsbereiche aufgeteilt. So existieren vier Hauptkomponenten: der **KNXReader**, der **Aggregator**, der **AMLWriter** und das **Network** samt dazugehöriger Klassen. Diese werden in Abbildung 6.1 dargestellt.

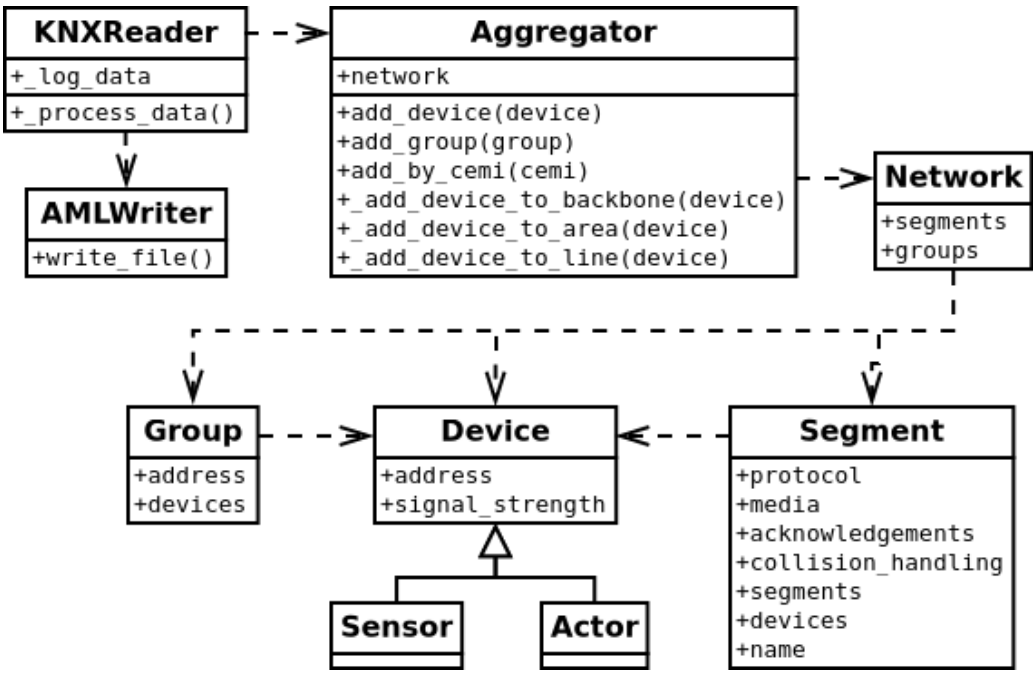


Abbildung 6.1: Die Hauptkomponenten des Prototyps

Gestartet wird die Topologierekonstruktion durch die Klasse **KNXReader** (in Listing 6.1). Dieser erhält die Position der Log-Datei als Parameter und liest sie mit Hilfe des *pandas*-Moduls ein. Die dort vorhandenen Einträge im *cEMI*-Format (siehe Abs. 5.1) werden nach und nach eingelesen.

```
class KNXReader(Reader):
 def _process_data(self):
  for row in self._log_data.iterrows():
   cemi = row[1]["cemi"]
   self.aggregator.add_by_cemi(cemi)

 def __init__(self, path):
  super().__init__(path, sep=";")
```

Listing 6.1: Ausschnitt aus *KNXReader*

Die Klasse **Aggregator** besitzt einen verhältnismäßig großen Funktionsumfang, da dort die Informationen im *cEMI*-Format entgegengenommen und verarbeitet werden. Hierfür werden auch Hilfsfunktionen genutzt, welche nicht im Klassendiagramm in Abbildung 6.1 zu sehen sind.

```
"""Byte 3: Ctrl 1"""
ctrl_1 = cemi[4+offset:6+offset]
ctrl_1_bin = _to_binary(ctrl_1)

ctrl_1_frame_type=int(ctrl_1_bin[0])
# reserved flag at [1]
ctrl_1_repeat = int(ctrl_1_bin[2])
ctrl_1_bc_type = int(ctrl_1_bin[3])
ctrl_1_priority=int(ctrl_1_bin[4:6])
ctrl_1_ack_req = int(ctrl_1_bin[6])
ctrl_1_confirm = int(ctrl_1_bin[6])
```

Listing 6.2: Ausschnitt aus `add_by_cemi`

In Listing 6.2 kann gesehen werden, wie die cEMI-Informationen durch die Funktion `add_by_cemi(cemi)` des Aggregators verarbeitet werden. Hierbei werden alle in Abschnitt 5.1 vorgestellten Informationen ermittelt und gespeichert. Mit diesen soll es zum späteren Zeitpunkt möglich sein, die Klassen der an der Kommunikation beteiligten Geräte zu ermitteln.

Nachdem entschieden wird, welche Klasse dem Gerät zugeordnet wird, wird dieses an die Funktion `add_device(device)` (Listing 6.3) übergeben. Diese erhält als Parameter eine konkrete Geräteinstanz übergeben.

Hier wird die KNX-Adresse des Gerätes in ihre drei Bestandteile zerbrochen, um es seiner Position in der Topologie zuzuordnen und auf bereits ermittelte Geräte zu überprüfen. Für die Einordnung verfügt der Aggregator über eine Variable des Typs `Network` als Referenz seines Modells. Wenn die nötigen Segmente zum Zeitpunkt des Hinzufügens nicht existieren, werden diese durch ihre jeweiligen Funktionen erzeugt.

```
def add_device(self, device):
 knx = str.split(device.address, ".")
 """Device on Backbone"""
 if knx[0] == "0":
  self._add_device_to_bb(device)
  return
 """Device in Area"""
 if knx_addr[1] == "0":
  self._add_device_to_area(device)
  return
 """Device in Line"""
 self._add_device_to_line(device)
```

Listing 6.3: Ausschnitt aus `add_device`

Die Elemente der Topologie selbst werden in der Datei `topology.py` implementiert. Dort finden sich alle in Abbildung 4.4 gezeigten Klassen wieder. Wie auch in Abbildung

6.1 zu sehen ist, verfügen `Network` und `Segment` über Variablen namens `devices` und `segments`. Diese wurden als Python-`dict` implementiert, damit schneller auf bekannte Elemente zugegriffen werden kann. Ihre Schlüssel sind jeweils die Adressen der Geräte und die Bezeichner der KNX-Bereiche und -linien.

```
def write_file(network):
 pre ='''<CAEXFile SchemaVersion=...
 <SuperiorStandardVersion>Automat...
 <SourceDocumentInformation Origi...
 '''

 hierarchy=_write_inst_hier(network)
 interfaces=_write_interface_libs()
 roles = _write_role_libs()
 units = _write_unit_class_libs()
 attributes =_write_attribute_libs()
 post = '''</CAEXFile>
 '''

 aml =pre+hierarchy+interfaces+roles
 aml = aml+units+attributes+post

 f = open(file_name, "w")
 f.write(aml)
 f.close()
```

Listing 6.4: Ausschnitt aus `add_device`

Schlussendlich wird die durch den Aggregator ermittelte Topologie mittels des **AMLWriter**s in das AML-Format geschrieben. Die dafür verwendete Funktion `write_file(network)` ist in Listing 6.4 gezeigt. Hierfür werden hauptsächlich die in Anhang A.3 gezeigten AML-Objektstrukturen wiederverwendet und um Informationen aus der rekonstruierten Topologie ergänzt. Die Funktion `_write_segment(segment)` ruft sich rekursiv auf, insofern das als Parameter übergebene Segment weitere Untersegmente beinhaltet. Die verschiedenen Bibliotheken werden durch ihre Funktionen statisch generiert und aus dem Projekt der Abbildung 3.3 wiederverwendet.

Der Ablauf des Programms ist in Abbildung 6.2 zusammengefasst dargestellt. Die `main()`-Funktion befindet sich in der `reconstructor.py`. Hier werden auch alle zur Ausführung nötigen Parameter eingelesen. In der `logs.py` ist das passive Scannen mittels Log-Dateien implementiert. Die `aggregator.py` stellt den Aggregator bereit, der eine Topologie nach dem Konzept in Kapitel 4 erzeugt. Diese wird durch die Funktionen in der Datei `aml_writer.py` in das AML-Format geschrieben. Die einzige Datei, die in Abbildung 6.2 nicht aufgelistet ist, ist die `topology.py`, in der alle Netzwerkelemente aus Abbildung 4.4 implementiert wurden.

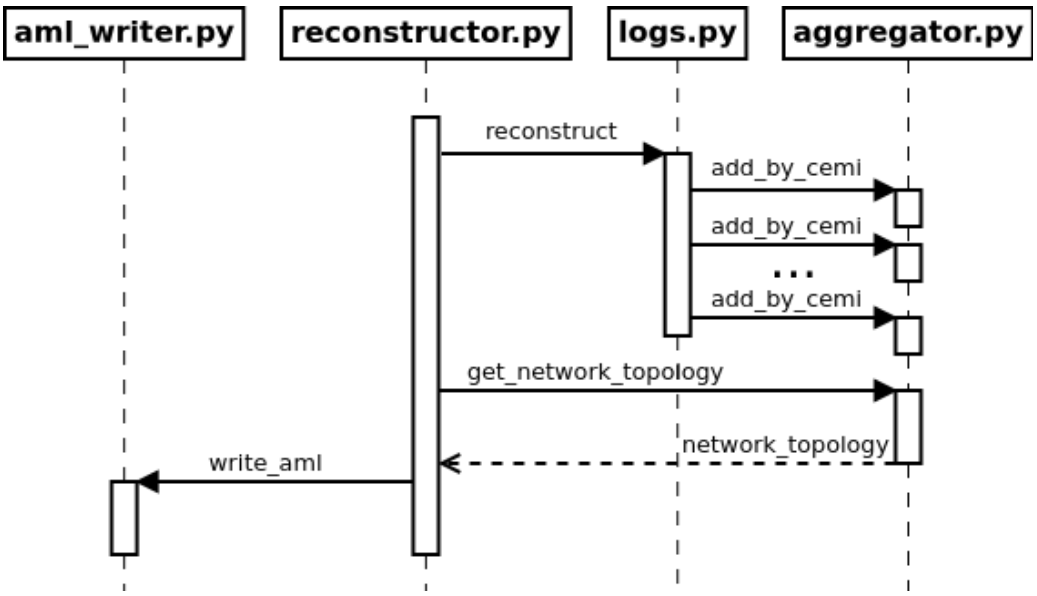


Abbildung 6.2: Der Funktionsablauf des Prototyps

6.3 Passives Scannen mittels Logs

Wie bereits erwähnt, wurde für den hier vorgestellten Prototypen das passive Scannen implementiert. Für diese Arbeit wurde ein KNX-Traffic-Log mit 322.757 Einträgen im CSV-Format bereitgestellt. Die dadurch bereitgestellten Informationen waren die folgenden:

- sequence_nbr
- timestamp
- source_addr
- destination_addr
- apci
- tpci
- priority
- repeated
- hop_count
- apdu
- payload_length
- **cemi**
- payload_data
- is_manipulated
- attack_type_id

Der Eintrag der *cEMI* ist dabei am wichtigsten, da die meisten der anderen Informationen sich aus dieser ableiten lassen. Die vollständige Auswertung dieser Einträge erfolgt wie in Listing 6.2 dargestellt in der Datei `aggregator.py`. *cEMI* ist der einzige der oben genannten Einträge, welcher von dem hier vorgestellten Prototypen verarbeitet wird. Dies sorgt für eine einfache Erweiterbarkeit zu anderen Datenquellen.

6.4 Ergebnis

Das Ergebnis der Rekonstruktion aus den Informationen des Logs ist in Abbildung 6.3 im AML-Editor zu sehen. Die verwendete Log-Datei hatte bei 322.757 Einträgen eine physische Größe von etwa 50 MiB. An den Timestamps lässt sich erkennen, dass ein Zeitraum von 15 Tagen aufgezeichnet wurde. Die Laufzeit der Verarbeitung des Logs betrug auf einem Intel i7-4770k Prozessor wiederholt 35 Sekunden. Die physische Größe der verarbeiteten Datei ist um ein Vielfaches auf 102 KiB geschrumpft.

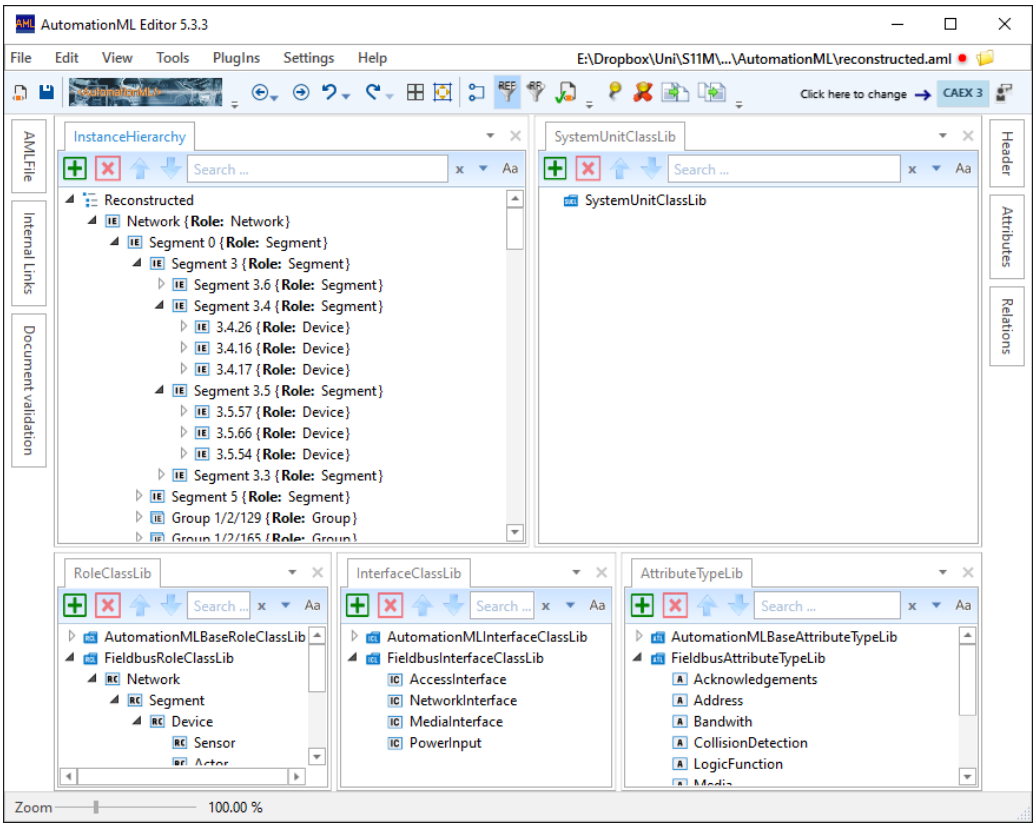


Abbildung 6.3: Das rekonstruierte Netzwerk im AML-Editor

Die Ergebnisse in Abbildung 6.3 zeigen, dass das in Kapitel 4 vorgestellte Konzept praktisch umgesetzt werden kann. Auch zeigen sie, dass der hier entwickelte Prototyp bereits eine valide AML-Datei erzeugen und die Topologie eines KNX-Netzwerkes korrekt nachbilden kann.

6.5 Bewertung

In diesem Kapitel wurde gezeigt, wie ein Prototyp nach den Vorgaben von Kapitel 4 realisiert werden kann. Dabei wurden die verschiedenen in Abschnitt 4.1 vorgestellten Eigenschaften der logischen Topologie und die in Abschnitt 4.1.1 gezeigten Geräteklassen implementiert.

Der hier vorgestellte Prototyp wurde ausschließlich für das passive Scannen von KNX-Netzwerken entwickelt. Eine Erweiterung um das aktive Scannen, wie in Abschnitt 5.3 beschrieben, sollte jedoch ohne großen Aufwand nachgearbeitet werden können, wenn ein entsprechendes KNX-Netzwerk zum Testen zur Verfügung steht.

Abbildung 6.3 zeigt auch, dass es möglich ist, Netzwerke (wie in dem in Kapitel 5 vorgestellten Beispiel) automatisch und korrekt in das AML-Format zu übertragen. Dabei ist jedoch zu beachten, dass die hier vorgestellte Implementierung noch über keine Funktionalität zur Erkennung der Gerätetypen verfügt.

Auch kann ein Großteil der Geräteeigenschaften noch nicht korrekt zugewiesen werden. Hierfür müssen unter Umständen weitere Datenquellen wie die Dokumentation einzelner Standards oder funktionale Abhängigkeiten zwischen Geräten betrachtet werden. Besonders dies und die Klassifizierung der Geräte sollte ein wichtiger Bestandteil weiterführender Arbeiten sein.

Das Ergebnis kann anhand einiger der in Abschnitt 4.2 vorgestellten Qualitätskriterien bewertet werden. Wegen der oben aufgezeigten Gründe ist es jedoch nicht möglich, Aussagen über die **eindeutig erkannten Geräte**, die **Vollständigkeit der Geräteeigenschaften** und das **Passiv-Aktiv-Verhältnis** zu treffen.

Da KNX-Netzwerke generell dem gleichen Aufbau aus Linien und Bereichen folgen, ist zu erwarten, dass die rekonstruierte **Topologie** der tatsächlich vorliegenden entspricht. Eine Auswertung nach der **Anzahl der Geräte** ist in Tabelle 6.1 gezeigt. Die Angaben zu den im Netzwerk verfügbaren Geräten stammen aus interner Dokumentation.

	Linie 3.6	**Linie 3.5**	**Linie 3.4**	**Linie 3.3**	**Linie 5.1**	$\sum$
vorhanden	49	70	37	32	-	9.530
gefunden	37	3	3	1	1	45
Verhältnis	*0,76*	*0,04*	*0,09*	*0,03*	-	*0,005*

Tabelle 6.1: Gefundene Geräte des KNX-Netzwerks

Aus Tabelle 6.1 lässt sich ableiten, dass durch rein passive Scans vor allem diejenigen Geräte entdeckt werden, welche sich in der logischen Nähe des aufzeichnenden Gerätes befinden. In *Linie 3.6* fällt das Ergebnis von 0,76 gut aus – dies bedeutet, dass drei von vier Geräten gefunden werden. Jedoch erreicht das passive Verfahren außerhalb der eigenen Linie keine Werte über 0,1 (abgesehen von *Linie 5.1*), was einen sehr kleinen Informationsgewinn bedeutet. Als Lösung für dieses Problem wäre es möglich, weitere Beobachtungspunkte in anderen Linien einzurichten oder einen aktiven Scanner zu nutzen.

Die **benötigte Zeit** des Verfahrens ist durchweg als positiv zu bewerten. Die Auswertung von 322.757 Einträgen in 35 Sekunden lässt darauf schließen, dass die Anwendung in Echtzeit auch mit intelligenten Geräteklassifizierungsalgorithmen möglich ist. Die **Größe der Log-Datei** von 50 MiB hingegen ist ein Nachteil des Verfahrens. Bei 45 erkannten Geräten werden hier im Durchschnitt etwa 1.1 MiB für die Identifizierung eines Gerätes benötigt (Vergleich: rekonstruierte AML-Datei mit 102 KiB). Um den relativen Informationsgewinn im Bezug auf die Dateigröße zu ermitteln, wurde die Topologie erneut mit verschiedenen Log-Größen rekonstruiert und unter Betrachtung von *Linie 3.6* ausgewertet. Die Ergebnisse sind in Tabelle 6.2 zu sehen.

Einträge	1.000	10.000	100.000	322.757
Größe	158,4 KiB	1,6 MiB	15,7 MiB	50,6 MiB
gefunden	6	22	34	37
Verhältnis	*0,12*	*0,45*	*0,69*	*0,76*

Tabelle 6.2: Gefundene Geräte in *Linie 3.6* in Relation zur Anzahl der Einträge

Hier kann erkannt werden, dass exponentiell größere Logs benötigt werden, um im späten Verlauf der Messung weitere Informationen zu gewinnen. Tabelle 6.2 zeigt somit die Beschränkungen des passiven Verfahrens auf. Wie bereits in Abschnitt 4.2.2 angemerkt, sind einige Geräte unter Umständen so konfiguriert, dass sie nur sehr

selten Informationen senden. Dies macht sie passiv nur schwer identifizierbar. Durch aktives Scannen hingegen sollte die Rekonstruktion der Topologie um ein Vielfaches effizienter werden.

Zusammenfassend lässt sich sagen, dass das hier vorgestellte System eine gute Grundlage für weiterführende Arbeiten bildet. Es konnte gezeigt werden, dass GAS wie KNX durch das in Kapitel 4 vorgestellte Konzept in AML dargestellt werden können.

Kapitel 7

Abschließende Betrachtungen

In diesem Kapitel

Zum Abschluss dieser Arbeit sollen ein letztes Mal dessen Grundlagen und Ergebnisse zusammenfasst werden. Hierfür werden alle Kapitel kurz behandelt und wichtige Aspekte aufgezeigt. Schlussendlich wird erwähnt, wie künftige Arbeiten die Ergebnisse aus Kapitel 4 und 6 erweitern können.

7.1 Zusammenfassung

Die Grundlage für diese Arbeit wurde in Kapitel 1 vorgestellt. Dies ist die Dokumentenerosion, welche ein Problem für Feldbussysteme verschiedenster GAS darstellt. Hier wird das Problem betrachtet, dass mit steigendem Alter einer GAS-Installation dessen modifizierte Netzwerktopologie durch die dazugehörige Dokumentation schlechter nachvollzogen werden kann. Aus diesem Grund wird die Rekonstruktion des Netzwerks

in einem dreiteiligen Modell, bestehend aus der *logischen Topologie*, der *physischen Topologie* und der *funktionalen Topologie*, vorgeschlagen.

Um ein Modell zu spezifizieren, welches mit den unterschiedlichen auf dem Markt verfügbaren GAS kompatibel ist, werden in Kapitel 2 verschiedene Zugriffsmedien und Standards betrachtet. Hier wird ein besonderes Augenmerk auf *KNX* gelegt, da im späteren Verlauf der Arbeit ein System auf dessen Basis implementiert wurde. Um die Relevanz der Netzwerktopologien zu zeigen, wird auf einige Topologieerkennungsprojekte im Bereich des Internets hingewiesen. Diese sind des Weiteren auch aus dem Gesichtspunkt der dafür genutzten Algorithmen von Interesse.

In der Forschung existieren bereits mehrere Sprachen zur Beschreibung von Netzwerktopologien. Kapitel 3 widmet sich der Erklärung einiger dieser Frameworks und zeigt deren Relevanz für die vorliegende Arbeit auf. Besonders das letztendlich zur Modellierung gewählte AML wird im Detail behandelt. Hier wird explizit gezeigt, wie die *logische Topologie*, die *physische Topologie* und die *funktionale Topologie* in dem Standard modelliert werden können.

Das konkrete Modell wird in Kapitel 4 vorgestellt. Es werden eine Vielzahl an darstellbaren Geräteklassen und -eigenschaften aufgezeigt und diese ausführlich in Anhang A.1 und A.2 im AML-Format definiert. Auch werden mögliche Datenquellen für die automatisierte Topologieerkennung in Feldbussen behandelt. Die Einschätzung der Ergebnisse kann anhand einiger in Kapitel 4 vorgestellter Qualitätskriterien erfolgen.

Da diese Arbeit sich praktisch mit der Darstellung von KNX-Netzwerken beschäftigt, wird in Kapitel 5 ein beispielhaftes Netzwerk im Detail ausgewertet und im vorher vorgestellten Modell dargestellt. Es wird gezeigt, dass sich eine KNX-Topologie in diesem darstellen lässt. Auch wird in diesem Kapitel das auf KNX-Feldbussen genutzte Nachrichtenformat *cEMI* ausführlich erläutert und in Bezug zur Informationsgewinnung in solchen Netzwerken gestellt.

Schlussendlich zeigt Kapitel 6, wie der Prototyp einer automatisierten Topologieerkennung für KNX implementiert werden kann. Dieser wurde auf Basis des passiven Scannens anhand von Log-Dateien implementiert. Die Ergebnisse der Rekonstruktion können im AML-Format gespeichert und im offiziellen AML-Editor angezeigt werden. Auch auf mögliche Erweiterungen wird hier eingegangen.

7.2 Weiterführende Überlegungen

Kapitel 6 zeigt die ersten Bemühungen zur Implementierung der automatisierten Topologieerkennung anhand des in Kapitel 4 vorgestellten Konzeptes. Obwohl hier bereits valide AML-Strukturen zur Darstellung von logischen KNX-Topologien entstehen, kann trotzdem lediglich von einem *Proof of Concept* geredet werden. In diesem Abschnitt werden weitere Möglichkeiten zum Aufbau auf die in dieser Arbeit vorgestellten Ergebnisse gezeigt.

Zunächst sollte genau beobachtet werden, was mit Hilfe des in Kapitel 6 vorgestellten Prototypen erzeugt werden kann: Hierbei handelt es sich um eine einfache KNX-Netzwerktopologie, welche aus verschiedenen Segmenten, Geräten und Gruppen besteht. In Abschnitt 4.1 und 4.1.1 werden eine Vielzahl von Eigenschaften und Gerätetypen definiert. Diese werden zwar in dem vorgestellten Prototypen implementiert, jedoch noch nicht in der automatisierten Topologieerkennung genutzt. In weiterführenden Arbeiten sollten diese Geräteeigenschaften vollständiger ausgefüllt werden. Auch sollte ein System konzipiert werden, welches zuverlässig Geräteklassen der am Netzverkehr teilnehmenden Kommunikationspartner erkennen kann.

Bereits in Abschnitt 5.3 erhielten die Möglichkeiten des aktiven Scannens Ansprache. Auch wurde dort postuliert, dass die Erweiterung eines passiven Scanners (wie in Kapitel 6 vorgestellt) um einen aktiven Teil nur wenig Aufwand erfordern würde. Diese Aussage sollte durch die Konzipierung und Implementierung eines entsprechenden aktiven Moduls untermauert werden. Hierfür würde jedoch auf jeden Fall der Zugang zu einem KNX-Netzwerk zum Testen der Komponente benötigt werden.

Wie in Kapitel 3.4 gezeigt, unterstützt AML mit CAEX nicht nur die Möglichkeit zur Darstellung von logischen Topologien, sondern dank der Teilstandards COLLADA und PLCopen XML auch Erweiterungen für die physische und funktionale Topologie. Aufgrund des Umfangs jeder dieser Standards, wurde in dieser Arbeit lediglich die logische Topologie im Detail behandelt. Für die vollständige Modellierung von Feldbussen, muss in weiterführenden Arbeiten auch auf die Darstellung der anderen Topologien eingegangen werden. In diesen sollte insbesondere geprüft werden, inwiefern diese Standards sich für die Repräsentation dieser Topologien eignen.

Die Rekonstruktion in Abbildung 6.3 wurde anhand eines passiven Scans mittels Log-Dateien ermittelt. Da im Rahmen dieser Arbeit keine vollständigen, dazugehörigen Topologieinformationen vorlagen, konnten viele der in Abschnitt 4.2 behandelten Kriterien nicht zur Bewertung der Qualität herangezogen werden. Wie diese sich zur Einschätzung des Verfahrens eignen, sollte in aufbauenden Untersuchungen betrachtet werden.

Das Konzept in Kapitel 4 ist im Hinblick auf Kompatibilität zu einer größtmöglichen Anzahl von GAS spezifiziert worden. In dieser Arbeit wurde jedoch ausschließlich KNX behandelt. Aus diesem Grund sollte weiterhin beobachtet werden, wie gut das vorgeschlagene Modell zur Darstellung anderer GAS genutzt werden kann. Auch muss für jedes weitere GAS eine Komponente implementiert werden, welche die Ergebnisse eines Scans in die vorgestellte Struktur übersetzen kann. Erst wenn diese Untersuchungen zeigen, dass sich eine Vielzahl von Systemen durch das hier vorgestellte Modell im für Sicherheitsanalysen nötigen Detail darstellen lassen, kann dieses wahrlich als erfolgreich betrachtet werden.

Anhang A

Konzept

A.1 Rollen

Tabelle A.1: Modellierung der Klasse ”Device” in AutomationML

Klassenname	Device
Beschreibung	Die Klasse ”Device” dient zur Darstellung eines abstrakten Gerätes in einem GAS. Dieses soll vor allem genutzt werden, wenn nicht genügend Information über ein Gerät vorliegen, um es in eine konkrete Klasse einzuordnen.
Elternklasse	`FieldbusRoleClasseLib/Network/Segment`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment/Device`
Attribute	**Name:** Address **RefAttributeType:** `FieldbusAttributeLib/Address` Semantik in Abschnitt 4.3.5
	Name: SignalStrength **RefAttributeType:** `FieldbusAttributeLib/SignalStrength` Semantik in Abschnitt 4.3.5

Tabelle A.2: Modellierung der Klasse "Segment" in AutomationML

Klassenname	Segment
Beschreibung	Die Klasse "Segment" dient zur Darstellung einzelner Netzwerksegmente. Diese können durch Aufteilung in Linien und Bereiche, die Verbindung mit anderen Medien und Protokollen, oder durch das Verbundensein mit kabellosen Routern und Repeatern entstehen. Ein Segment kann weitere Segmente und Geräte beinhalten. Zur Verknüpfung von Segmenten werden die Interfaces von Routern oder Repeater in den beteiligten Segmenten verknüpft. Segmente können hierarchisch in anderen Segmenten, oder im Wurzelelement der Instanzhierarchie (Network) vorkommen.
Elternklasse	`FieldbusRoleClasseLib/Network`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment`
Attribute	**Name:** Protocol **RefAttributeType:** `FieldbusAttributeLib/Protocol` Semantik in Abschnitt 4.3.5
	Name: Bandwidth **RefAttributeType:** `FieldbusAttributeLib/Bandwith` Semantik in Abschnitt 4.3.5
	Name: Media **RefAttributeType:** `FieldbusAttributeLib/Media` Semantik in Abschnitt 4.3.5
	Name: Acknowledgements **RefAttributeType:** `FieldbusAttributeLib/Acknowledgements` Semantik in Abschnitt 4.3.5
	Name: CollisionHandling **RefAttributeType:** `FieldbusAttributeLib/CollisionHandling` Semantik in Abschnitt 4.3.5

Tabelle A.3: Modellierung der Klasse ”Network” in AutomationML

Klassenname	Network
Beschreibung	Die Klasse ”Network” dient als Elternklasse für alle zu modellierenden Elemente. Dazu gehören Segmente, in welchen wiederum weitere Segmente und Geräte definiert werden können. Sie ist in einem Modell genau einmal zu definieren.
Elternklasse	Keine
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network`
Attribute	Keine

Tabelle A.4: Modellierung der Klasse ”Sensor” in AutomationML

Klassenname	Sensor
Beschreibung	Die Klasse ”Sensor” dient zur Darstellung eines Sensors in einem GAS und ist eine Konkretisierung der Klasse ”Device”.
Elternklasse	`FieldbusRoleClasseLib/Network/Segment/Device`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment/Device/Sensor`
Attribute	**Name:** Address **RefAttributeType:** `FieldbusAttributeLib/Address` Semantik in Abschnitt 4.3.5
	Name: SignalStrength **RefAttributeType:** `FieldbusAttributeLib/SignalStrength` Semantik in Abschnitt 4.3.5

Tabelle A.5: Modellierung der Klasse ”Actor” in AutomationML

Klassenname	Actor
Beschreibung	Die Klasse ”Actor” dient zur Darstellung eines Aktoren in einem GAS und ist eine Konkretisierung der Klasse ”Device”.
Elternklasse	`FieldbusRoleClasseLib/Network/Segment/Device`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment/Device/Sensor`
Attribute	**Name:** Address **RefAttributeType:** `FieldbusAttributeLib/Address` Semantik in Abschnitt 4.3.5
	Name: SignalStrength **RefAttributeType:** `FieldbusAttributeLib/SignalStrength` Semantik in Abschnitt 4.3.5

Tabelle A.6: Modellierung der Klasse ”Router” in AutomationML

Klassenname	Router
Beschreibung	Die Klasse ”Router” dient zur Darstellung eines Routers in einem GAS und ist eine Konkretisierung der Klasse ”Device”. Router dienen zur Verknüpfung von Segmenten und werden deshalb in jedem beteiligten Segment als eigenständiges Gerät dargestellt – auch dann, wenn sie im realen System nur ein einziges Mal existieren. Um eine Verbindung zwischen Segmenten herzustellen, können die Interfaces zweier Router miteinander verknüpft werden.
Elternklasse	`FieldbusRoleClasseLib/Network/Segment/Device`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment/Device/Router`
Attribute	**Name:** Address **RefAttributeType:** `FieldbusAttributeLib/Address` Semantik in Abschnitt 4.3.5

Tabelle A.7: Modellierung der Klasse "Repeater" in AutomationML

Klassenname	Repeater
Beschreibung	Die Klasse "Repeater" dient zur Darstellung eines Repeaters in einem GAS und ist eine Konkretisierung der Klasse "Device". Repeater dienen zur Verknüpfung von Segmenten und werden deshalb in jedem beteiligten Segment als eigenständiges Gerät dargestellt – auch dann, wenn sie im realen System nur ein einziges Mal existieren. Um eine Verbindung zwischen Segmenten herzustellen, können die Interfaces zweier Repeater miteinander verknüpft werden.
Elternklasse	`FieldbusRoleClasseLib/Network/Segment/Device`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment/Device/Repeater`
Attribute	**Name:** Address **RefAttributeType:** `FieldbusAttributeLib/Address` Semantik in Abschnitt 4.3.5

Tabelle A.8: Modellierung der Klasse "Gateway" in AutomationML

Klassenname	Gateway
Beschreibung	Die Klasse "Gateway" dient zur Darstellung eines Gateways in einem GAS und ist eine Konkretisierung der Klasse "Device". Gateways dienen zur Verknüpfung von Segmenten **verschiedener Medien** und werden deshalb in jedem beteiligten Segment als eigenständiges Gerät dargestellt – auch dann, wenn sie im realen System nur ein einziges Mal existieren. Um eine Verbindung zwischen Segmenten herzustellen, können die Gateways zweier Router miteinander verknüpft werden.
Elternklasse	`FieldbusRoleClasseLib/Network/Segment/Device`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment/Device/Gateway`
Attribute	**Name:** Address **RefAttributeType:** `FieldbusAttributeLib/Address` Semantik in Abschnitt 4.3.5
	Name: MediaConnection **RefAttributeType:** `FieldbusAttributeLib/MediaConnection` Semantik in Abschnitt 4.3.5

Tabelle A.9: Modellierung der Klasse ”Central” in AutomationML

Klassenname	Central
Beschreibung	Die Klasse ”Central” dient zur Darstellung der Zentrale eines GAS und ist eine Konkretisierung der Klasse ”Device”. Zentralen sind in einigen GAS die kontrollierenden Geräte für die Steuerung des gesamten Systems.
Elternklasse	`FieldbusRoleClasseLib/Network/Segment/Device`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment/Device/Central`
Attribute	**Name:** Address **RefAttributeType:** `FieldbusAttributeLib/Address` Semantik in Abschnitt 4.3.5

Tabelle A.10: Modellierung der Klasse ”Interface” in AutomationML

Klassenname	Interface
Beschreibung	Die Klasse ”Interface” dient zur Darstellung eines Businterfaces und ist eine Konkretisierung der Klasse ”Device”. Mit Hilfe von Interfaces ist es möglich, auf den Busverkehr zuzugreifen.
Elternklasse	`FieldbusRoleClasseLib/Network/Segment/Device`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment/Device/Interface`
Attribute	**Name:** Address **RefAttributeType:** `FieldbusAttributeLib/Address` Semantik in Abschnitt 4.3.5

Tabelle A.11: Modellierung der Klasse ”Logic” in AutomationML

Klassenname	Logic
Beschreibung	Die Klasse ”Logic” dient zur Darstellung logischer Geräte jeder Art eines GAS und ist eine Konkretisierung der Klasse ”Device”. Mit Hilfe von Logic-Devices können logische Funktionen ausgeführt werden.
Elternklasse	`FieldbusRoleClasseLib/Network/Segment/Device`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment/Device/Logic`
Attribute	**Name:** Address **RefAttributeType:** `FieldbusAttributeLib/Address` Semantik in Abschnitt 4.3.5
	Name: LogicFunction **RefAttributeType:** `FieldbusAttributeLib/LogicFunction` Semantik in Abschnitt 4.3.5
	Name: SignalStrength **RefAttributeType:** `FieldbusAttributeLib/SignalStrength` Semantik in Abschnitt 4.3.5

Tabelle A.12: Modellierung der Klasse "Voltage" in AutomationML

Klassenname	Voltage
Beschreibung	Die Klasse "Voltage" dient zur Darstellung jeglicher stromregulierender Geräte eines GAS und ist eine Konkretisierung der Klasse "Device". Zu diesen zählen Geräte wie Phasenkoppler, Bandsperren, Stromzähler und die Stromversorgung für den Feldbus.
Elternklasse	`FieldbusRoleClasseLib/Network/Segment/Device`
Pfad für Elementreferenz	`FieldbusRoleClasseLib/Network/Segment/Device/Voltage`
Attribute	**Name:** Address **RefAttributeType:** `FieldbusAttributeLib/Address` Semantik in Abschnitt 4.3.5
	Name: VoltageHertz **RefAttributeType:** `FieldbusAttributeLib/VoltageHertz` Semantik in Abschnitt 4.3.5
	Name: VoltageVolts **RefAttributeType:** `FieldbusAttributeLib/VoltageVolts` Semantik in Abschnitt 4.3.5

A.2 Attribute

Tabelle A.13: Definitionen der in Abschnitt 4.3.4 verwendeten Attribute

Attribut	Beschreibung
Acknowledgements	Das Attribut "Acknowledgements" dient zur Kennzeichnung der Nutzung von Acknowledgements in einem Segment. Beispiel: `Acknowledgements="true"` **AttributeDataType:** `xs:boolean` **Pfad:** `FieldbusAttributeLib/Acknowledgements`
Address	Das Attribut "Address" dient zur Darstellung der Adresse eines Gerätes in einem GAS. Abhängig von dem konkret verwendeten System hat diese eine vorgeschriebene Form und muss nicht kompatibel zu den verwendeten Adresstypen fremder Segmente sein. Beispiel: `Address="0.8.15"` **AttributeDataType:** `xs:string` **Pfad:** `FieldbusAttributeLib/Address`
Bandwith	Das Attribut "Bandwith" dient zur Darstellung der maximalen Übertragungsgeschwindigkeit eines Segments. Sie wird als Gleitkommazahl und in kbit/s angegeben. Beispiel: `Bandwith="250.0"` **AttributeDataType:** `xs:decimal` **Pfad:** `FieldbusAttributeLib/Bandwith`
CollisionHandling	Das Attribut "CollisionHandling" dient zur Kennzeichnung der Kollisionsbehandlung eines Segments. Dies kommt typischerweise bei kabellosen Protokollen vor. Beispiel: `CollisionHandling="CA"` **AttributeDataType:** `xs:string` **Pfad:** `FieldbusAttributeLib/CollisionHandling`
LogicFunction	Das Attribut "LogicFunction" dient Erläuterung der logischen Funktion eines Logik-Gerätes in einem GAS. Abhängig von dessen Funktion kann dieses Attribut für ein Gerät folgende Werte annehmen: `AND, OR, XOR, MULTIPLEX, SHADE, TIME, DELAY, FILTER, ARITHMETIC, INVERT, THRESHOLD ,BYCHANGE, LOCKOUT, GENERATOR, CONVERT, COUNT` (abgeleitet von [Gir] abgeleitet). Beispiel: `LogicFunction="AND"` **AttributeDataType:** `xs:string` **Pfad:** `FieldbusAttributeLib/LogicFunction`

Media	Das Attribut "Media" dient zur Beschreibung des von einem Segment genutzten Mediums. Es wird empfohlen einen der folgenden Werte dafür zu nutzen: `TP, RF, PL, IP`. Beispiel: `Media="IP"` **AttributeDataType:** `xs:string` **Pfad:** `FieldbusAttributeLib/Media`
MediaConnection	Das Attribut "MediaConnection" dient zur Beschreibung der möglichen Medienverbindungen eines Gateways. Es wird empfohlen, einen der folgenden Werte dafür zu nutzen: `TP, RF, PL, IP`. Es können weitere dieser Werte hinzugefügt werden, wenn diese durch die Verwendung von Kommas voneinander getrennt werden. Beispiel: `MediaConnection="TP,RF"` **AttributeDataType:** `xs:string` **Pfad:** `FieldbusAttributeLib/MediaConnection`
Protocol	Das Attribut "Protocol" dient zur Beschreibung des von einem Segment genutzten Protokolls. Hier sollen möglichst kurze Werte zur Kennzeichnung des GAS-Typs genutzt werden. Beispiele sind: `KNX, LON, ZigBee, ZWave`. Beispiel: `Protocol="KNX"` **AttributeDataType:** `xs:string` **Pfad:** `FieldbusAttributeLib/Protocol`
SignalStrength	Das Attribut "SignalStrength" dient in kabellosen Segmenten zur Kennzeichnung der durchschnittlichen Signalstärke zum segmenterzeugenden Router, Repeater oder Gateway. Die Angabe der Signalstärke erfolgt in dBm. Beispiel: `SignalStrength="-57.0"` **AttributeDataType:** `xs:decimal` **Pfad:** `FieldbusAttributeLib/SignalStrength`
VoltageHertz	Das Attribut "VoltageHertz" dient zur Darstellung der Netzfrequenz in Voltage-Devices. Die Angabe erfolgt in Hz. Beispiel: `VoltageHertz="60.0"` **AttributeDataType:** `xs:decimal` **Pfad:** `FieldbusAttributeLib/VoltageHertz`
VoltageVolts	Das Attribut "VoltageVolts" dient zur Darstellung der elektrischen Spannung in Voltage-Devices. Die Angabe erfolgt in Volt. Beispiel: `VoltageVolts="230.0"` **AttributeDataType:** `xs:decimal` **Pfad:** `FieldbusAttributeLib/VoltageVolts`

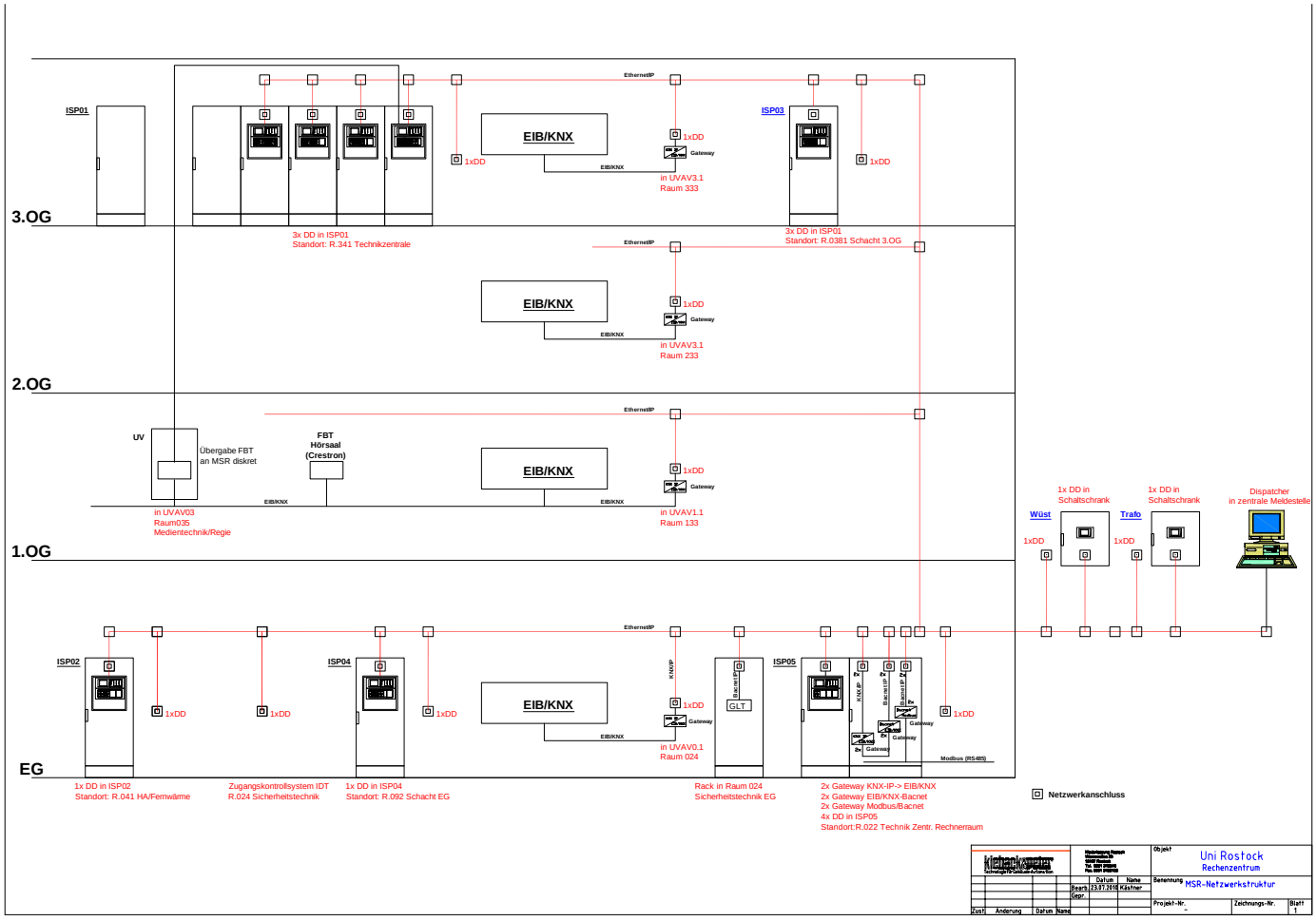
A.3 Darstellung von Abbildung 5.3 in CAEX

```
<InstanceHierarchy Name="TwoAreas">
 <Version>0</Version>
 <InternalElement Name="Network" ID="68eb519a-ba81-47e3-8b32-08e56da41fda">
  <InternalElement Name="Segment␣0" ID="03f7a5da-9ef3-48b2-ae45-6658a3472db4">
   <Attribute Name="Protocol" AttributeDataType="xs:string" RefAttributeType="
       FieldbusAttributeTypeLib/Protocol">
    <Value>KNX</Value>
   </Attribute>
   <Attribute Name="Bandwith" AttributeDataType="xs:decimal" RefAttributeType="
       FieldbusAttributeTypeLib/Bandwith" />
   <Attribute Name="Media" AttributeDataType="xs:string" RefAttributeType="
       FieldbusAttributeTypeLib/Media" />
   <Attribute Name="Acknowledgements" AttributeDataType="xs:boolean" RefAttributeType="
       FieldbusAttributeTypeLib/Acknowledgements" />
   <Attribute Name="CollisionHandling" AttributeDataType="xs:boolean" RefAttributeType="
       FieldbusAttributeTypeLib/CollisionHandling" />
    <InternalElement Name="0.0.17" ID="1dc48bc4-0ff8-4cbc-a6e4-a168386d342a">
     <Attribute Name="Address" AttributeDataType="xs:string" RefAttributeType="
         FieldbusAttributeTypeLib/Address">
      <Value>3.5.3</Value>
     </Attribute>
     <Attribute Name="SignalStrength" AttributeDataType="xs:string" RefAttributeType="
         FieldbusAttributeTypeLib/SignalStrength" />
     <RoleRequirements RefBaseRoleClassPath="FieldbusRoleClassLib/Network/Segment/Device">
      <Attribute Name="Address" AttributeDataType="xs:string" RefAttributeType="
          FieldbusAttributeTypeLib/Address" />
      <Attribute Name="SignalStrength" AttributeDataType="xs:string" RefAttributeType="
          FieldbusAttributeTypeLib/SignalStrength" />
     </RoleRequirements>
    </InternalElement>
   <InternalElement Name="0.0.52" ID="3be64df3-5bc6-49bb-9a7d-c2ba02d66a18">
    ... </InternalElement>
   <InternalElement Name="Segment␣3" ID="d19417de-0bc7-4253-b00e-ecd66d37ebf9">
    ...
    <InternalElement Name="Segment␣3.5" ID="63f3fa92-0ff0-47c6-a8d6-b5900309c75f">
     ...
     <InternalElement Name="3.5.3" ID="9b76c545-5fd5-4e04-a35e-c76504d5db1f">
      ... </InternalElement>
     <InternalElement Name="3.5.47" ID="68a3c3ba-99c3-436a-8539-8e0ef4e032c6">
```

```
      ... </InternalElement>
     <InternalElement Name="3.5.82" ID="70c1bed3-3d0e-4889-9c63-e224a2fffc9a">
      ... </InternalElement>
     <InternalElement Name="3.5.178" ID="202ae8d2-b62c-44a8-93cd-a8db86ba540b">
      ... </InternalElement>
     <InternalElement Name="3.5.201" ID="d25432ba-7a29-4ab2-a807-76feceb8fa64">
      ... </InternalElement>
     <RoleRequirements RefBaseRoleClassPath="FieldbusRoleClassLib/Network/Segment">
      <Attribute Name="Protocol" AttributeDataType="xs:string" RefAttributeType="
          FieldbusAttributeTypeLib/Protocol" />
      <Attribute Name="Bandwith" AttributeDataType="xs:decimal" RefAttributeType="
          FieldbusAttributeTypeLib/Bandwith" />
      <Attribute Name="Media" AttributeDataType="xs:string" RefAttributeType="
          FieldbusAttributeTypeLib/Media" />
      <Attribute Name="Acknowledgements" AttributeDataType="xs:boolean" RefAttributeType="
          FieldbusAttributeTypeLib/Acknowledgements" />
      <Attribute Name="CollisionHandling" AttributeDataType="xs:boolean" RefAttributeType=
          "FieldbusAttributeTypeLib/CollisionHandling" />
     </RoleRequirements>
    </InternalElement>
    <InternalElement Name="Segment␣3.7" ID="e9ef1516-ed65-44af-b245-77ce08af6ef8">
     ...
     <InternalElement Name="3.7.12" ID="62ed9c6b-25f6-467c-983a-53a8ccca7dcd">
      ... </InternalElement>
     <InternalElement Name="3.7.33" ID="1c394282-85fd-43c0-8310-e1ed1a8fc64e">
      ... </InternalElement>
     <InternalElement Name="3.7.78" ID="070ae3b5-77c6-41c7-9e50-e92a4ed6fcd1">
      ... </InternalElement>
     <RoleRequirements RefBaseRoleClassPath="FieldbusRoleClassLib/Network/Segment">
      ... </RoleRequirements>
    </InternalElement>
    <RoleRequirements RefBaseRoleClassPath="FieldbusRoleClassLib/Network/Segment">
     ... </RoleRequirements>
   </InternalElement>
   <InternalElement Name="Segment␣5" ID="d8e0b9c8-500f-48fb-b7dd-849c4b5469a3">
    ...
    <InternalElement Name="Segment␣5.12" ID="93c6618d-a123-47fb-b517-664add2657f9">
     ...
     <InternalElement Name="5.12.66" ID="174e8d93-0456-44b7-a839-73311efcfbd9">
      ... </InternalElement>
```

```
      <InternalElement Name="5.12.128" ID="fed10f74-8f46-4d1e-aef2-25d2b24730e1">
       ... </InternalElement>
      <InternalElement Name="5.12.222" ID="d0d90e7a-e774-4d1d-af29-517ace0aff11">
       ... </InternalElement>
      <InternalElement Name="5.12.250" ID="a7f97df5-f3ce-4bc1-97dd-89eb0897089e">
       ... </InternalElement>
      <RoleRequirements RefBaseRoleClassPath="FieldbusRoleClassLib/Network/Segment">
       ... </RoleRequirements>
      </InternalElement>
     <RoleRequirements RefBaseRoleClassPath="FieldbusRoleClassLib/Network/Segment">
      ... </RoleRequirements>
    </InternalElement>
    <RoleRequirements RefBaseRoleClassPath="FieldbusRoleClassLib/Network/Segment">
     ... </RoleRequirements>
   </InternalElement>
   <InternalElement Name="Group 3/5/120" ID="c04da652-ace0-4647-ab0e-3725fde4c661">
    <RoleRequirements RefBaseRoleClassPath="AutomationMLBaseRoleClassLib/
        AutomationMLBaseRole/Group">
     <Attribute Name="AssociatedFacet" RefAttributeType="AutomationMLBaseAttributeTypeLib/
         AssociatedFacet" AttributeDataType="xs:string" />
    </RoleRequirements>
   </InternalElement>
   <RoleRequirements RefBaseRoleClassPath="FieldbusRoleClassLib/Network" />
  </InternalElement>
</InstanceHierarchy>
```

A.4 Planungsunterlagen als Datenquellen



Anhang B

Datenträger

Abbildungen

Literatur

[AC17] Robert Lipovsky Anton Cherepanov. *Industroyer: Biggest threat to industrial control systems since Stuxnet*. https://www.welivesecurity.com/2017/06/12/industroyer-biggest-threat-industrial-control-systems-since-stuxnet/. [Online; Zugriff am 15.08.2019]. 2017 (siehe S. 3).

[Adh13] Richard Adhikari. *Webcam Maker Takes FTC's Heat for Internet-of-Things Security Failure*. https://www.technewsworld.com/story/78891.html. [Online; Zugriff am 14.08.2019]. 2013 (siehe S. 3).

[Asc14] Bernd Aschendorf. *Energiemanagement durch Gebäudeautomation: Grundlagen-Technologien-Anwendungen*. Springer-Verlag, 2014 (siehe S. 2, 9–13, 15, 17, 46).

[Aut14] AutomationML e.V. c/o IAF. *Whitepaper AutomationML Part 2 - Role class libraries*. Techn. Ber. AutomationML e.V. c/o IAF Universitätsplatz 2 39106 Magdeburg Germany: AutomationML consortium, Okt. 2014 (siehe S. 33, 61).

[Aut17a] AutomationML e.V. c/o IAF. *Whitepaper AutomationML Part 3 - Geometry and Kinematics*. Techn. Ber. AutomationML e.V. c/o IAF Universitätsplatz 2 39106 Magdeburg Germany: AutomationML consortium, Jan. 2017 (siehe S. 33, 61).

[Aut17b] AutomationML e.V. c/o IAF. *Whitepaper AutomationML Part 4 - AutomationML Logic*. Techn. Ber. AutomationML e.V. c/o IAF Universitätsplatz 2 39106 Magdeburg Germany: AutomationML consortium, Okt. 2017 (siehe S. 33, 61).

[Aut18] AutomationML e.V. c/o IAF. *Whitepaper AutomationML Edition 2.1 Part 1 - Architecture and General Rquirements*. Techn. Ber. AutomationML e.V. c/o IAF Universitätsplatz 2 39106 Magdeburg Germany: AutomationML consortium, Juli 2018 (siehe S. 30–33, 61).

[Ber+16] Luca Berardinelli, Rainer Drath, Emanuel Maetzler und Manuel Wimmer. „On the evolution of CAEX: A language engineering perspective“. In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2016, S. 1–8. DOI: 10.1109/ETFA.2016.7733497 (siehe S. 31).

[Cai] *Archipelago (Ark) Measurement Infrastructure*. http://www.caida.org/projects/ark/. abgerufen am 30.01.2020 (siehe S. 21).

[Col] *COLLADA – Digital Asset Schema Release 1.5.0 – Specification*. Apr. 2008 (siehe S. 35–39).

[Cou10] Council of the European Commission and others. „Directive 2010/31/EU of the European Parliament and of the council of 19 May 2010 on the energy performance of buildings“. In: *Official Journal of the European Union* 153 (2010), S. 13–35 (siehe S. 2, 8).

[Cra04] William C Craig. „Zigbee: Wireless control that simply works“. In: *Zigbee Alliance ZigBee Alliance* (2004) (siehe S. 16).

[DEH] DEHOF. *cEMI Message Format*. https://dehof.de/eib/pdfs/EMI-FT12-Message-Format.pdf. abgerufen am 1.3.2020 (siehe S. 68).

[DFC05] Benoit Donnet, Timur Friedman und Mark Crovella. „Improved Algorithms for Network Topology Discovery“. In: *Passive and Active Network Measurement*. Hrsg. von Constantinos Dovrolis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, S. 149–162. ISBN: 978-3-540-31966-5 (siehe S. 21–23).

[Dom+16] Pedro Domingues, Paulo Carreira, Renato Vieira und Wolfgang Kastner. „Building automation systems: Concepts and technology review“. In: *Computer Standards & Interfaces* 45 (2016), S. 1–12. DOI: 10.1016/j.csi.2015.11.005. URL: https://doi.org/10.1016/j.csi.2015.11.005 (siehe S. 23).

[Don+18] Michele De Donno, Nicola Dragoni, Alberto Giaretta und Angelo Spognardi. „DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation“. In: *Security and Communication Networks* 2018 (2018), 7178164:1–7178164:30. DOI: 10.1155/2018/7178164. URL: https://doi.org/10.1155/2018/7178164 (siehe S. 3).

[Dra+08] Rainer Drath, Arndt Lüder, Jörn Peschke und Lorenz Hundt. „AutomationML - the glue for seamless automation engineering“. In: *Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2008, September 15-18, 2008, Hamburg, Germany*. 2008, S. 616–623. DOI: 10.1109/ETFA.2008.4638461. URL: https://doi.org/10.1109/ETFA.2008.4638461 (siehe S. 60).

[Dra+10] Rainer Drath, Björn Grimm, Lorenz Hundt, Andreas Keibel, Steffen Lips, Arndt Lüder, Miriam Schleipen und Dirk Weidemann. *Datenaustausch in der Anlagenplanung mit AutomationML*. Hrsg. von Rainer Drath. Springer, 2010 (siehe S. 30, 31, 35, 36, 40, 41).

[Dro15] Alex Drozhzhin. *Black Hat USA 2015: The full story of how that Jeep was hacked*. https://www.kaspersky.com/blog/blackhat-jeep-cherokee-hack-explained/9493/. [Online; Zugriff am 14.08.2019]. 2015 (siehe S. 3).

[FG13] Behrang Fouladi und Sahand Ghanoun. „Security evaluation of the Z-Wave wireless protocol“. In: *Black hat USA* 24 (2013), S. 1–2 (siehe S. 16, 17).

[Fou] Python Software Foundation. *LanguageComparisons*. https://wiki.python.org/moin/LanguageComparisons. [Online; Zugriff am 29.04.2020] (siehe S. 76).

[Fra+13] Mirko Franceschinis, Claudio Pastrone, Maurizio A. Spirito und Claudio Borean. „On the performance of ZigBee Pro and ZigBee IP in IEEE 802.15.4 networks“. In: *2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. 2013, S. 83–88. DOI: 10.1109/WiMOB.2013.6673344 (siehe S. 15).

[Gay10] Sandro Gaycken. *Wer war's? Und wozu?* https://www.zeit.de/2010/48/Computerwurm-Stuxnet. [Online; Zugriff am 06.08.2019]. 2010 (siehe S. 3).

[Geo+01] Fotis Georgatos, Florian Gruber, Daniel Karrenberg, Mark Santcroos, Ana Susanj, Henk Uijterwaal und René Wilhelm. „Providing active measurements as a regular service for ISPs“. In: *PAM*. 2001 (siehe S. 18, 20).

[Gir] Gira. *Gira L1 Logic Nodes*. https://www.gira.com/en/gebaeudetechnik/systeme/knx-eib_system/knx-produkte/systemgeraete/logikmodul/logikbausteine.html. abgerufen am 29.2.2020 (siehe S. 97).

[GKK16] Harald Glanzer, Lukas Krammer und Wolfgang Kastner. „Increasing security and availability in KNX networks“. In: *Sicherheit 2016: Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 8. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 5.-7. April 2016, Bonn*. 2016, S. 241–252. URL: https://dl.gi.de/20.500.12116/875 (siehe S. 5).

[Gol18] Johannes Goltz. „Sicherheitsanalyse von Gebäudeautomationsnetzen auf Feldbusebene am Beispiel von KNX“. Magisterarb. Universität Rostock, 2018 (siehe S. 57).

[Gra14] Wolfgang Granzer. „Secure Communication in Home and Building Automation Systems“. Diss. Technische Universität Wien, 2014 (siehe S. 17).

[Ham+13] Jeroen van der Ham, Freek Dijkstra, Roman Lapacz, Jason Zurawski u. a. „Network markup language base schema version 1“. In: Muncie, INOpen Grid Forum. 2013 (siehe S. 29).

[Huf+02] Bradley Huffaker, Daniel Plummer, David Moore und KC Claffy. „Topology discovery by active probing“. In: *Proceedings 2002 Symposium on Applications and the Internet (SAINT) Workshops*. IEEE. 2002, S. 90–96 (siehe S. 18, 20, 21).

[Int] Intenis. *Legacy Products*. https://www.intesis.com/support/discontinued-products. abgerufen am 24.3.2020 (siehe S. 17).

[JKK14] Aljosha Judmayer, Lukas Krammer und Wolfgang Kastner. „On the security of security extensions for IP-based KNX networks“. In: *2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014)*. IEEE. 2014, S. 1–10 (siehe S. 5).

[Kas+05] Wolfgang Kastner, Georg Neugschwandtner, Stefan Soucek und H Michael Newman. „Communication systems for building automation and control“. In: *Proceedings of the IEEE* 93.6 (2005), S. 1178–1203 (siehe S. 8).

[Kat19] Srinivas Katipamula. „Building Automation: Where is it Today and Where it Should be“. In: *15th IEEE International Conference on Automation Science and Engineering, CASE 2019, Vancouver, BC, Canada, August 22-26, 2019*. 2019 (siehe S. 1, 2, 8).

[Kel] Kele. *BACnet - LonWorks - Modbus - SNMP - Wireless Gateways*. https://www.kele.com/network-and-wireless/babel-buster-series.aspx. abgerufen am 24.3.2020 (siehe S. 17).

[Knxa] *EN 50090*. Standard. European Committee for Electrotechnical Standardization, 2011 (siehe S. 12).

[Knxb] *Grundlagenwissen zum KNX Standard*. KNX Association. 2013 (siehe S. 2, 3, 10, 11, 13).

[Knxc] *KNX Sicherheit - KNX Positionspapier zu Datensicherheit und Datenschutz*. KNX Association. 2018 (siehe S. 5, 11).

[Knxd] *KNX System Specifications*. 2009 (siehe S. 13).

[Kol+17] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou und Jeffrey M. Voas. „DDoS in the IoT: Mirai and Other Botnets“. In: *IEEE Computer* 50.7 (2017), S. 80–84. DOI: 10.1109/MC.2017.201. URL: https://doi.org/10.1109/MC.2017.201 (siehe S. 3).

[Kus13] David Kushner. *The Real Story of Stuxnet*. https://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet. [Online; Zugriff am 06.08.2019]. 2013 (siehe S. 3).

[Lar17] Selena Larson. *FDA confirms that St. Jude's cardiac devices can be hacked*. https://money.cnn.com/2017/01/09/technology/fda-st-jude-cardiac-hack/. [Online; Zugriff am 14.08.2019]. 2017 (siehe S. 3).

[LGS06] A. Luntovskyy, D. Gutter und A. Schill. „Network Design and Optimization Under Use of Candy Framework“. In: *2006 16th International Crimean Microwave and Telecommunication Technology*. Bd. 1. 2006, S. 394–397 (siehe S. 26, 27).

[LTD] DUSUN Electron LTD. *Legacy Products*. https://www.dusuniot.com/zigbee-gateway. abgerufen am 24.3.2020 (siehe S. 17).

[Lun+07] Andriy Luntovskyy, Taissia Trofimova, Natalia Trofimova, Dietbert Gütter und Alexander Schill. „To a proposal towards standardization of network design markup language“. In: *International Network Optimization Conference (INOC'07), Spa, Belgium*. 2007 (siehe S. 26, 27).

[MBB00] Tony McGregor, H-W Braun und Jeff Brown. „The NLANR network analysis infrastructure“. In: *IEEE Communications Magazine* 38.5 (2000), S. 122–128 (siehe S. 18, 19).

[MHH09] Hermann Merz, Thomas Hansemann und Christof Hübner. *Building automation*. Springer, 2009 (siehe S. 8, 15).

[MHH16] Hermann Merz, Thomas Hansemann und Christof Hübner. *Gebäudeautomation: Kommunikationssysteme mit EIB/KNX, LON und BACnet*. Carl Hanser Verlag GmbH Co KG, 2016 (siehe S. 9).

[Our+20] M. Ourahou, W. Ayrir, B. E. L. Hassouni und A. Haddi. „Review on smart grid control and reliability in presence of renewable energies: Challenges and prospects“. In: *Mathematics and Computers in Simulation* 167 (2020), S. 19–31. DOI: 10.1016/j.matcom.2018.11.009. URL: https://doi.org/10.1016/j.matcom.2018.11.009 (siehe S. 2).

[PDR08] Vamsi Paruchuri, Arjan Durresi und M Ramesh. „Securing powerline communications“. In: *2008 IEEE International Symposium on Power Line Communications and Its Applications*. IEEE. 2008, S. 64–69 (siehe S. 11).

[Saj+06] Ondrej Sajdl, Zdenek Bradác, Petr Fiedler und Ondrej Hyncica. „ZigBee Technology and Device Design“. In: *Fifth International Conference on Networking and the International Conference on Systems (ICN / ICONS / MCL 2006), 23-29 April 2006, Mauritius*. 2006, S. 129. DOI: 10.1109/ICNICONSMCL.2006.233. URL: https://doi.org/10.1109/ICNICONSMCL.2006.233 (siehe S. 15, 16).

[Sch17] Fabian A. Scherschel. *Industroyer: Fortgeschrittene Malware soll Energieversorgung der Ukraine gekappt haben*. https://www.heise.de/security/meldung/Industroyer-Fortgeschrittene-Malware-soll-Energieversorgung-der-Ukraine-gekappt-haben-3740606.html. [Online; Zugriff am 15.08.2019]. 2017 (siehe S. 3).

[SE19] Inc. Stack Exchange. *Developer Survey Results 2019*. https://insights.stackoverflow.com/survey/2019. [Online; Zugriff am 29.04.2020]. 2019 (siehe S. 76).

[SHS17] Frank Sokollik, Peter Helm und Ralph Seela. *KNX für die Gebäudesystemtechnik in Wohn- und Zweckbau*. VDE Verlag GMBH, 2017 (siehe S. 11–13).

[SL15] Nicole Schmidt und Arndt Lüdter. *AutomationML in a Nutshell*. Techn. Ber. AutomationML e.V. Office, 2015 (siehe S. 30).

[sol] see solutions.de. *KNX Twisted Pair Protokollbeschreibung*. http://www.see-solutions.de/sonstiges/KNXTwistedPairProtokollbeschreibung.pdf. [Online; Zugriff am 29.04.2020] (siehe S. 69).

[Sym18] Symantec Security Response. *VPNFilter: New Router Malware with Destructive Capabilities*. https://www.symantec.com/blogs/threat-intelligence/vpnfilter-iot-malware. [Online; Zugriff am 13.08.2019]. 2018 (siehe S. 3).

[Tho16] Ian Thomson. *Wi-Fi baby heart monitor may have the worst IoT security of 2016*. https://www.theregister.co.uk/2016/10/13/possibly_worst_iot_security_failure_yet/. [Online; Zugriff am 14.08.2019]. 2016 (siehe S. 3).

[Vas+] Volodymyr Vasyutynskyy, A Schill, A Luntovskyy, G Pfeifer, D Gütter, A Panchenko und V Vasyutynskyy. „Computer Network Modeling and Analysis Using XML-Descriptions“. In: () (siehe S. 27).

[Wan10] H. Wang. „NevML: A Markup Language for Network Topology Visualization“. In: *2010 Second International Conference on Future Networks*. 2010, S. 119–123 (siehe S. 27, 28).

[WL18] Hamza Wertani und Mohamed Najeh Lakhoua. „Overview of Smart Grids Architecture and Design“. In: *International Conference on Smart Communications and Networking, SmartNets 2018, Yasmine Hammamet, Tunisia, November 16-17, 2018*. 2018, S. 1–5. DOI: 10.1109/SMARTNETS.2018.8707404. URL: https://doi.org/10.1109/SMARTNETS.2018.8707404 (siehe S. 2).

[WS08] Woo Suk Lee und Seung Ho Hong. „KNX — ZigBee gateway for home automation“. In: *2008 IEEE International Conference on Automation Science and Engineering*. 2008, S. 750–755. DOI: 10.1109/COASE.2008.4626433 (siehe S. 17).

[YMK16] M. B. Yassein, W. Mardini und A. Khalil. „Smart homes automation using Z-wave protocol“. In: *2016 International Conference on Engineering MIS (ICEMIS)*. 2016, S. 1–6. DOI: 10.1109/ICEMIS.2016.7745306 (siehe S. 16, 17).

Erklärung

Hiermit erkläre ich, Richard Dabels, dass ich die vorliegende Masterarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Die Arbeit ist noch nicht veröffentlicht und ist in ähnlicher oder gleicher Weise noch nicht als Prüfungsleistung zur Anerkennung oder Bewertung vorgelegt worden.

Rostock, den 1. Mai 2020