INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Ein generisches Werkzeug zur Automatisierung der CLI-basierten Konfiguration von Netzkomponenten

Matthias Meyer

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Ein generisches Werkzeug zur Automatisierung der CLI-basierten Konfiguration von Netzkomponenten

Matthias Meyer

Aufgabensteller: Prof. Dr. Dieter Kranzlmüller

Betreuer: Dr. Vitalian Danciu
Dipl.-Ing. Simone Ferlin Oliveira
Dr. Herbert Plansky (ckc AG)

Abgabetermin: 14. August 2012

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 14. August 2012

. . . . . .

(Unterschrift des Kandidaten)

Zusammenfassung

Es existiert eine Vielzahl unterschiedlicher managebarer Endgeräte in der Netztechnik. Um eine automatisierte Konfiguration dieser Endgeräte vorzunehmen, werden in der Regel proprietäre Programme des Herstellers oder auf bestimmte Situationen zugeschnittene Skripte benötigt. Eine Gemeinsamkeit ist jedoch meist die Möglichkeit der Steuerung über eine rudimentäre Benutzerschnittstelle, auch Command-Line Interface genannt. Diese Schnittstelle ist textbasiert, da sie für die manuelle Konfiguration durch Benutzer gedacht ist.

Ziel dieser Arbeit ist es, diese vorhandene Schnittstelle zu analysieren, um ein generisches Datenmodell zu entwickeln. Mit Hilfe dieses Datenmodells soll es ermöglicht werden eine Vielzahl verschiedener Endgeräte ansprechen zu können, um die Vorraussetzung einer automatisierten Steuerung bzw. Konfiguration zu schaffen.

Im weiteren Verlauf dieser Arbeit soll mittels einer Implementierung die korrekte Funktionsweise gezeigt werden. Hierzu wird die Datenstruktur in einem Modul für eine bestehende Managementschnittstelle umgesetzt. Dieses Modul wird abschließend in einer Testumgebung evaluiert.

Inhaltsverzeichnis

1. Einleitung

Große Probleme im Bereich der Administration in Netzen bereiten die erheblichen Unterschiede zwischen den verwendeten Endgeräten. Besonders in einer sehr inhomogenen Netzinfrastrukur steht ein Administrator vor dem Problem, dass die Endgeräte verschiedener Hersteller jeweils meist eine andere Art der Verwaltung und Konfiguration voraussetzen. Selbst bei verschiedenen Baureihen oder Softwareständen kann es sogar bei einem Hersteller zu Unterschieden kommen. Um dieses Problem zu lösen, gibt es mehrere Ansätze.

Einerseits ist es möglich mit propriertärer Software des Geräteherstellers zu arbeiten. Hier kann man als Beispiel den `ProCurve Manager` [HP00] von `HP` oder den `Network Assistant` [CISCO02] von `Cisco` nennen. Natürlich bringt eine solche Software eine Vielzahl verschiedener Funktionen mit sich; jedoch hat man das Problem, dass man verschiedene Management Systeme in einer inhomogenen Netzinfrastruktur betreiben muss, was meist mit relativ hohen Lizenzgebühren verbunden ist.

Der Einsatz von nicht proprietärer Management Software wäre auch ein mögliches Szenario. Dafür gibt es auch bereits OpenSource Lösungen. Als Beispiel lässt sich `NETDISCO` [NETD00] nennen. Jedoch kommt es bei Softwareupdates auf dem Endgerät oft zu Problemen, da sich dadurch das Verhalten der Schnittstellen, die genutzt werden, verändern und somit ein reibungsloser Ablauf nicht mehr gewährleistet werden kann. Es kann natürlich auch sein, dass bestimmte benötigte Funktionen nicht unterstützt werden. Auch gibt es oft Einschränkungen hinsichtlich der Firmware, Hardwaremodulen oder Hardwarerevisionen, die genutzt werden können. Somit ist man auf den Entwickler der Software angewiesen, der regelmäßige Updates bereit stellen oder den Funktionsumfang der Software anpassen muss.

Die Konfiguration aller Netzkomponenten von Hand ist letztlich auch ein denkbares Szenario, welches aber bei großen Umgebungen zu extremem Arbeitsaufwand führt. Auch müssen die Administratoren ein erhebliches Wissen aller Komponenten aufweisen. Zwar könnte man sich mit Skripten die Arbeit erleichtern; diese sind aber in der Regel auf bestimmte Konfigurationen zugeschnitten und müssen deshalb häufig angepasst werden.

Zwischen der Vielzahl der unterschiedlichen Netzkomponenten bestehen einige Gemeinsamkeiten, die man zur Konfiguration nutzen kann. Es existieren unter anderem folgende drei Möglichkeiten:

1. Simple Network Management Protocol (SNMP)
2. Trivial File Transfer Protocol (TFTP)
3. Command Line Interface (CLI)

Bei `SNMP` wird oft die Sicherheit kritisiert. So ist es in den Versionen 1 bis 2c nur möglich über Communities den Zugriff zu regeln (siehe [RFC1157] bzw. [RFC1901]). Meist werden auch aus Bequemlichkeit nur die Standard Communities „public“ und „private“ verwendet. Mit diesen Communities kann man Attribute der Netzkomponenten auslesen (public) oder sogar konfigurieren (private). So rät zum Beispiel das „Bundesamt für Sicherheit in der Informationstechnik“ SNMP nur dann zu nutzen, wenn es wirklich benötigt wird [BSI07].

Die Version 3 ermöglicht zwar eine sichere Benutzerauthentifizierung, wird aber von manchen Herstellern nicht unterstützt.

Dieser Versionskonflikt führt wieder zu einer Inhomogenität, da man für verschiedene Endgeräte die zu verwendenden Version beachten muss.

Viele Hersteller bieten die Möglichkeit an, die Konfiguration den Komponenten über TFTP zur Verfügung zu stellen. Jedoch muss hierfür ein extra TFTP Server aufgesetzt werden dessen Sicherheit auch als relativ schwach zu bewerten ist. So ist laut RFC1350 (siehe [RFC1350]) keinerlei Möglichkeit der Benutzerauthentifizierung implementiert, was zu einer erheblichen Sicherheitslücke in Firmennetzen führt.

Des Weiteren muss für jede Komponente eine extra Konfigurationsdatei zur Verfügung gestellt werden, was bei großen Infrastrukturen wieder zu einem erheblichen Arbeitsaufwand führt.

Ob jedoch SNMP oder TFTP zur Verfügung steht ist nicht unbedingt garantiert. Also kann der Umstieg auf einen anderen Hersteller dazu führen, dass das gesamte System auf die neue zur Verfügung stehende Technologie umgestellt werden muss.

Eine CLI Schnittstelle bietet zwei wichtige Vorteile:

Verfügbarkeit: Eine CLI Schnittstelle zur Konfiguration wird bei Netzkomponenten immer bereit gestellt.

Vollständigkeit: Man ist zum Beispiel nicht wie bei SNMP auf Hersteller spezifische Management Information Base (MIB) Dateien angewiesen und der darin verfügbaren Funktionen, sondern kann das gesamte Spektrum an Funktionen die eine Netzkomponente zur Verfügung stellt adressieren.

1.1. Aufgabenstellung

Im Rahmen der Bachelorarbeit soll eine Möglichkeit gefunden werden mehrere Endgeräte konfigurieren zu können, indem auf die einzelnen Besonderheiten ihrer Schnittstellen eingegangen wird. Diese werden aber innerhalb des Werkzeugs soweit generalisiert, dass man mit einer beschränkten Anzahl an Kommandos eine Vielzahl an Netzkomponenten konfigurieren kann. Es muss also eine Schnittstelle gefunden werden, die unabhängig von Hersteller und Firmware ist. Die bereits vorher erwähnten Nachteile von SNMP bzw. TFTP und die Verfügbarkeit bzw. Vollständigkeit der CLI führen zur Wahl dieser Schnittstelle.

Ein Nachteil bringt die Wahl der CLI Schnittstelle jedoch mit sich. So ist diese normalerweise für die Bedienung durch einen Menschen konzipiert. Der Aufbau ist meist so strukturiert, dass man durch eine Reihe von Befehlen in Form von Worten oder Zahlen navigieren kann. Diese Strukturen sind außerdem so ausgelegt, dass sie von einem Administrator leicht verständlich und gut lesbar sind. Auch sind manchmal weitere Strukturebenen eingefügt die lediglich zur logischen Strukturierung gedacht sind. Solche Ebenen besitzen keinerlei Funktionen zur Konfiguration, sondern dienen nur der Navigation, was besonders bei einem Aufbau in Form eines Menüs auftritt. So können hier zusätzliche künstliche Abstraktionsebenen geschaffen sein.

Also wird im Rahmen dieser Bachelorarbeit ein Werkzeug entwickelt, welches zwar einerseits die statischen Strukturen der CLI ausreichend beschreibt und diese zur automatisierten Konfiguration nutzen kann, andererseits aber auch den dynamischen Aspekt berücksichtigt der für eine automatisierte Konfiguration zwingend notwendig ist. Dieses Werkzeug soll also auf der einen Seite bestehende CLI Schnittstellen generalisieren. Auf der anderen Seite wird aber eine generische Schnittstelle zur Verfügung gestellt. Damit können dann mit Hilfe kleinerer Anpassungen des Werkzeugs alle CLI Schnittstellen ohne weiteren Aufwand zur Konfiguration der Netzkomponenten genutzt werden.

1.2. Methodik

Um die Anforderungen an ein solches Werkzeug ermitteln zu können, wird im ersten Schritt eine Anforderungsanalyse durchgeführt. Hierzu wird in Kapitel 2 auf ein beispielhaftes Managementszenario eingegangen. Dieses wird in einer Testumgebung nachgestellt. Die CLI der verwendeten Netzkomponente in dieser Umgebung, sowie die bereitgestellte Managementsoftware werden analysiert. Anhand der Resultate aus dieser Analyse wird ein Anforderungskatalog erstellt. Der aus Kapitel 2 entstandene Anforderungskatalog wird in Kapitel 3 anhand von bereits bestehenden Umsetzungen verglichen und die Anforderungen an das Werkzeug bestätigt. Im vierten Kapitel werden aus den Anforderungen Modelle definiert, die für den Entwurf des Werkzeugs genutzt werden. Dieser Entwurf wird in Kapitel 5 genutzt um einen Prototyp zu erstellen. Der entstandene Prototyp wird zuletzt in den bestehenden Management Adapter als Modul eingegliedert. Danach wird der Prototyp in der bereits aufgebauten Testumgebung getestet.

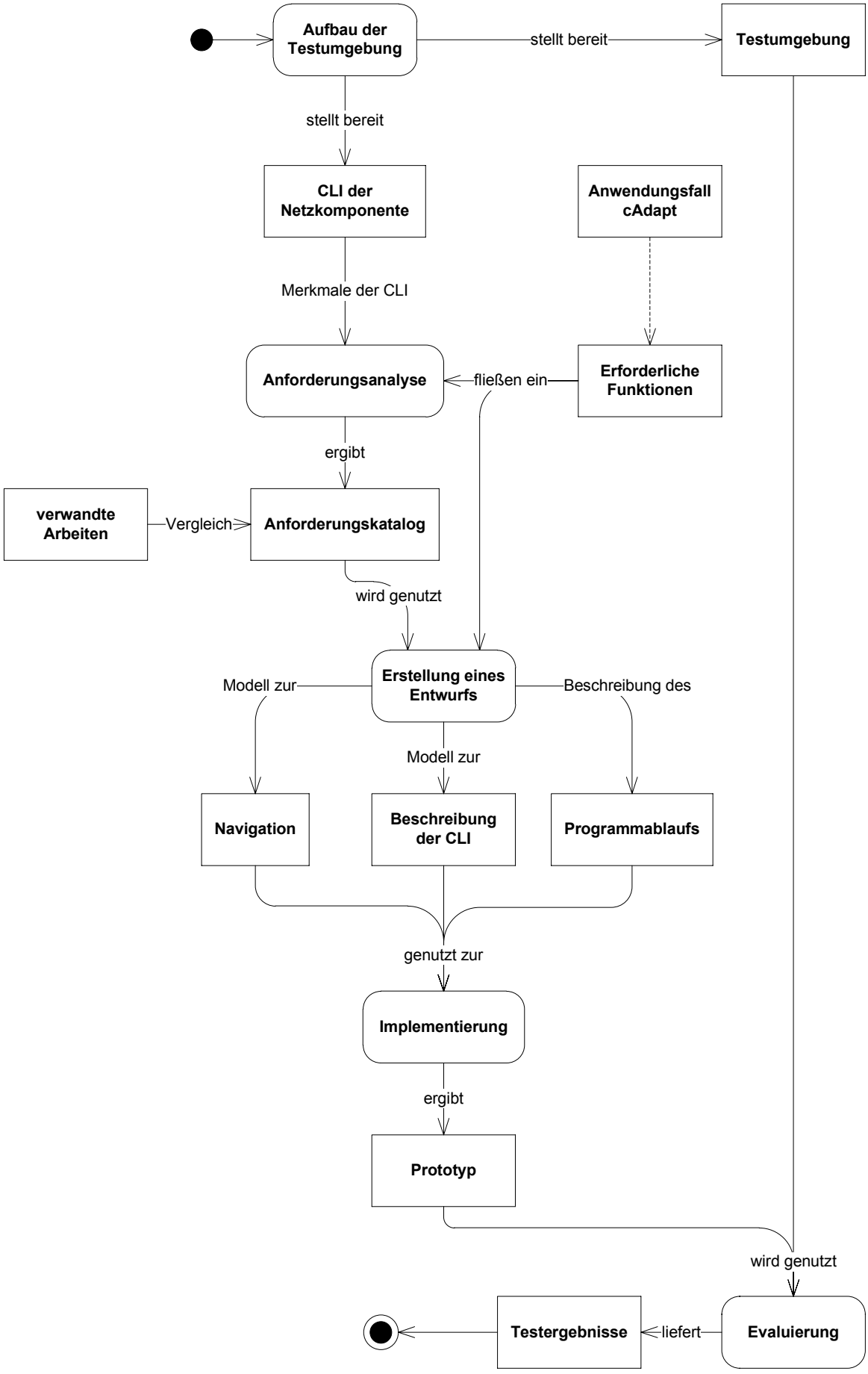

Abbildung 1.1.: Darstellung der Methodik

2. Anforderungsanalyse

Eine CLI Schnittstelle ist in erster Linie zur Bedienung durch einen Menschen gedacht; dadurch ist eine Ansteuerung oder gar Automatisierung kompliziert. Man müsste ein Textverständnis implementieren, das die in der CLI erscheinenden Texte interpretiert. Also muss das zu erstellende Werkzeug die entsprechenden statischen Strukturen der CLI Schnittstelle der zu konfigurierenden Netzkomponente ausreichend beschreiben, was eine Konfiguration erst möglich macht. Das Werkzeug soll aber auch generisch konstruiert werden, um eine Vielzahl verschiedener Netzkomponenten anzusprechen.

Um die statischen CLI Strukturen beschreiben zu können, ist es im ersten Schritt wichtig, die bestehenden Strukturen von verschiedenen Endgeräten und Herstellern zu analysieren. So kann man die verschiedenen Arten der CLI Schnittstellen vergleichen. Diese Analyse wird anhand von verschiedenen Grundfunktionen innerhalb einer Testumgebung durchgeführt.

2.1. Testumgebung

Im Rahmen der Testumgebung wird ein Fibre To The Home (FTTH) Szenario nachgestellt. Anhand dieses Szenarios kann die CLI Schnittstelle einer Netzkomponente analysiert werden. Es handelt sich um ein einfaches Netz das aus einem Switch und zwei Customer Premises Equipment (CPE) besteht (siehe Abb. 2.1). Bei dem Switch handelt es sich um einen Catalyst 4510R der Firma CISCO der mit entsprechendem Small Formfactor Pluggable (SFP) Modul ausgerüstet ist. Weiterhin wurden als CPE zwei Ethernet Gateways der Firma Genexis verbaut.

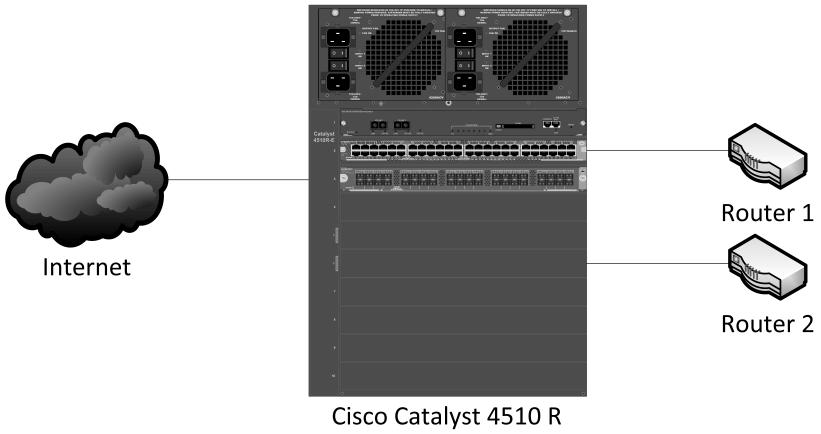

Abbildung 2.1.: Aufbau der Testumgebung

2.2. CLI Strukturen

Da es sich bei der CLI Schnittstelle um eine für den Menschen implementierte Schnittstelle handelt, muss, um die Konfiguration zuverlässig umsetzen zu können, die Position innerhalb der CLI Struktur klar erkennbar sein und die Ein- bzw. Ausgabe der CLI beachtet werden. Damit die Struktur der CLI Schnittstellen untersucht werden konnte, wurden auch Switche der Firma `HP` betrachtet. Diese Untersuchung der Schnittstellen ergab

Unterschiede, die man in zwei verschiedenen Ansätze zusammenfassen kann. Diese lassen sich wie in den folgenden Abschnitten beschrieben unterteilen.

2.2.1. Einfache Eingabezeilen

Eine Herangehensweise an die Strukturierung der CLI ist die Darstellung einer rudimentären Eingabezeile, in der der Benutzer seine Befehle eingeben kann. Die Eingaben werden dann meist mit einem `Carriage Return` bestätigt, ähnlich einer traditionellen Benutzerschnittstelle wie z.B. der `Unix-Shell`. Ein Beispiel ist in Abb. 2.2 zu sehen.

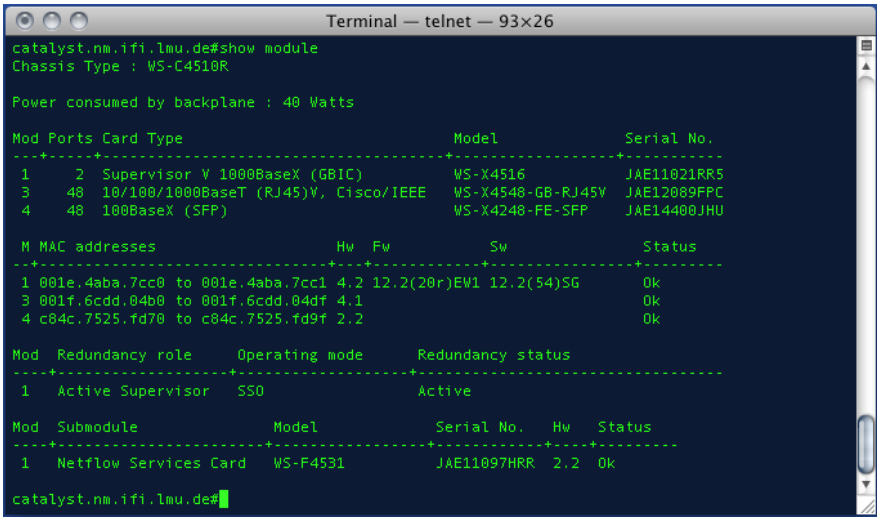

Abbildung 2.2.: Beispiel einer CISCO Schnittstelle

Hier kann man die Position innerhalb der Struktur auf den ersten Blick lediglich an einem String in der Eingabezeile erkennen, was eine Navigation durch die Struktur und damit die Konfiguration zwar möglich macht, aber eine definitive Position schwer erkennbar macht. Ein Administrator kann im Gegensatz zu einem Programm leicht durch die Schritte die er im Zuge der Konfiguration gemacht hat evaluieren an welcher Position er sich befindet.

So ergeben sich aus diesem Zusammenhang zwei Anforderungen an das Werkzeug. Es muss also eine Erkennung der Strings in der Eingabezeile ermöglicht werden um somit die **Position innerhalb der Struktur** zu belegen. Außerdem muss auch eine **Verwaltung der Positionen** innerhalb der Struktur gewährleistet werden.

2.2.2. Menüstrukturen

Einige der CLI Schnittstellen werden in Form einer simplen, einem Inhaltsverzeichnis gleichenden oder Menü ähnelnden Struktur dargestellt. Bei diesen ist es wie bei den einfachen Eingabezeilen möglich sich mit Zahlen oder Wortbefehlen durch die Struktur des Menüs zu bewegen. Oft lassen sich anhand der Überschrift die Position in der Menüstruktur ablesen, in der man sich momentan befindet. Hier wäre zum Beispiel in Abb. 2.3 die aktuelle Position das `VLAN Menu`.

In diesen Menüs lassen sich die Positionen innerhalb der Struktur leicht erkennen. Anhand von Überschriften oder bestimmten Funktionen kann man sich als Administrator leicht positionieren. Auch sind die Möglichkeiten der weiterführenden Navigation leicht ablesbar, weil hier die Funktionen meist in Texten beschrieben sind.

Daraus lässt sich eine weitere Anforderung definieren. Da auf einer Ebene innerhalb der Struktur nur eine gewisse Anzahl von Kommandos zur Verfügung stehen, muss das Werkzeug die **möglichen bzw. ausführbaren Kommandos den Positionen zuordnen** können.

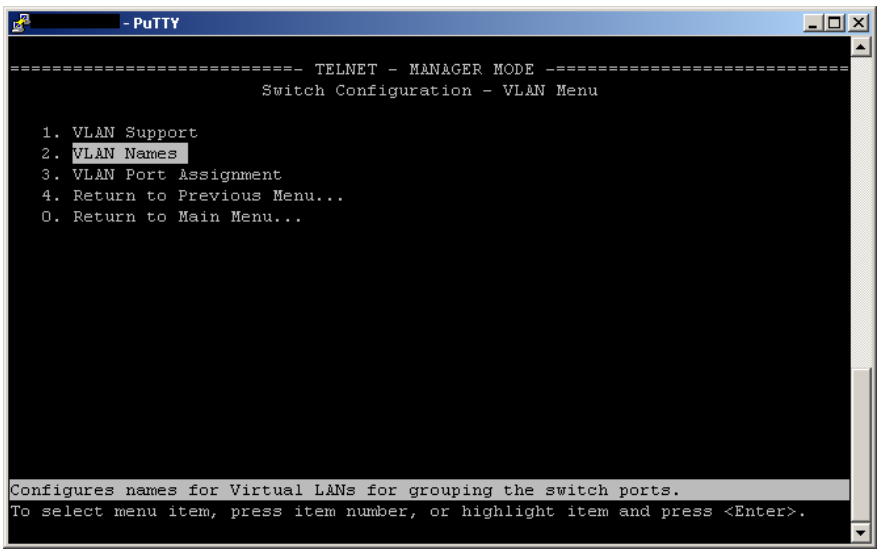

Abbildung 2.3.: Beispiel einer HP Menüstruktur

2.3. ckc cAdapt

Die Anforderungen an die generische Struktur der Schnittstelle, die das Werkzeug selbst bereit stellt, wird anhand eines bestehenden Management Adapters, in den das Werkzeug integriert werden soll, analysiert.

Bei diesem Management Adapter handelt es sich um den `cAdapt` der Firma ckc. Dieser wird in der Telekommunikationsbranche bei Stadtnetzbetreibern eingesetzt, um deren Dienste automatisiert bereit zu stellen und zu verändern (s. [HPL00]). Das ist besonders unter dem Gesichtspunkt der Breitbandstrategie des `Bundesministeriums für Wirtschaft und Technologie` interessant. Hier wird von Seiten des Ministeriums der Ausbau von leistungsfähigen Breitbandanschlüssen in Deutschland unterstützt. Das ermöglicht einem Stadtnetzbetreiber die Kontrolle über die sog. letzte Meile, also die Teilnehmeranschlussleitung zu erlangen, die in der Regel unter Monopolherrschaft steht. Hierzu muss natürlich ein rascher Ausbau der Netzinfrastruktur gewährleistet werden. Um diesen Ausbau leichter umsetzen zu können, ist ein Werkzeug von Nöten, welches die automatisierte Konfiguration von den benötigten Netzkomponenten ermöglicht.

Die Zielsetzung des Adapters ist es sich nahtlos in die bestehende Infrastruktur eines lokalen Netzbetreibers zu integrieren, um die Automatisierung von Managementprozessen zu ermöglichen.

Funktionsweise

Der Adapter besitzt ein sog. „Northbound Interface“, über dieses werden MTOSI (siehe [MTOSI00]) Anfragen von dem Informationssystem der Netzbetreiber entgegen genommen. Bei MTOSI handelt es sich um einen Standard zur Implementierung von Schnittstellen zwischen Netzmanagement Systemen in der Telekommunikation. Hierüber können einzelne Dienste aktiviert oder verändert werden.

Die Anfragen werden dann im „Request Mapper“ in Funktionsaufrufe umgewandelt. Das geschieht mit Hilfe von XML Dateien, in denen ein Aufruf am „Northbound Interface“ eine Anzahl von „Southbound Interface“ Aufrufen zugeordnet wird. Das „Southbound Interface“ verwandelt diese in Befehle, die die Netzinfrastruktur konfigurieren (siehe Abb. 2.4).

Daraus lassen sich weitere Anforderungen formulieren. Um die nahtlose Integration von cAdapt in bestehende Netzstrukturen zu ermöglichen, muss das Werkzeug also eine Vielzahl von Netzkomponenten unterstützen. Egal welche Kommandos eine CLI Schnittstelle unterstützt, sollten diese aber auf **eine feste Anzahl von Befehlen** abstrahiert werden. Trotzdem soll aber das gesamte Funktionsspektrum der Netzkomponente unterstützt werden. So muss im cAdapt nur ein bestimmter, vom Werkzeug zur Verfügung gestellter Befehl genutzt wer-

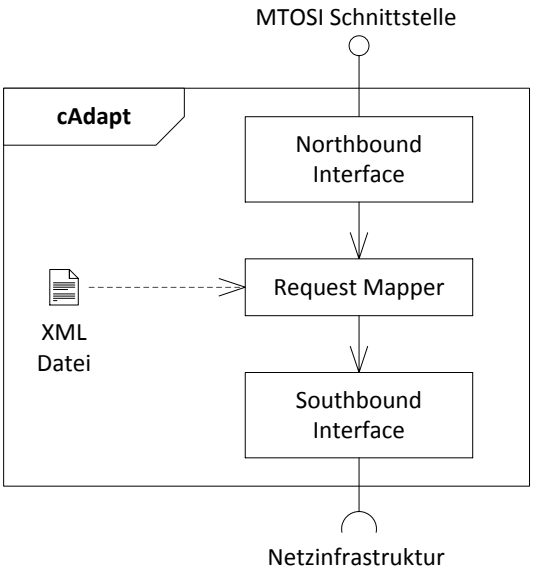

Abbildung 2.4.: Funktionsweise cAdapt

den, um bei allen Netzkomponenten das selbe Resultat zu erzielen. Das hält den Aufwand, z.B. bei einer evtl. Erweiterung der anzusteuernden Netzinfrastruktur, im Schritt des „Request Mapper“ so gering wie möglich.

Des Weiteren muss das Werkzeug, um alle Komponenten konfigurieren zu können, auch eine Möglichkeit bieten sich auf allen **authentifizieren** zu können.

2.4. Anforderungskatalog

Um die Umsetzung eines generischen Werkzeugs zur Konfiguration über eine CLI Schnittstelle zu realisieren, ergeben sich die nachstehenden Anforderungen.

2.4.1. Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben den tatsächlichen Funktionsumfang des Werkzeugs.

F1: Empfangen der Befehle vom Management Adapter Das Werkzeug muss, um die Konfigurationsanfragen der Management Software umsetzen zu können, eine geeignete Schnittstelle bereit stellen.

F2: Ermittlung der Position anhand der CLI Ausgabe Um die Position innerhalb der CLI Struktur ermitteln zu können, muss das Werkzeug die Ausgabe der CLI beachten. Die Ausgaben der CLI geben Rückschluss auf diese Position. Hierfür ist eine korrekte und vollständige Ausgabe zwingend notwendig. Ansonsten kann von einem Fehlerfall ausgegangen werden.

F3: Verwaltung der Positionen Da sich ein Administrator seine Schritte durch die CLI Struktur merken und somit entscheiden kann, ob er sich an der richtigen Position befindet, muss auch das Werkzeug selbst die Positionen verwalten können. Das erhöht die Sicherheit der richtigen Konfiguration.

F4: Erkennung von Fehlerfällen Fehlverhalten in der Firmware der Netzkomponente oder hohe Last können zu Fehlern während der Konfiguration führen. Diese muss das Werkzeug abfangen und entsprechend reagieren. Außerdem muss auf falsche oder unvollständige Ausgaben der CLI reagiert werden.

F5: Ausführen von Kommandos auf Netzkomponente Um die Konfiguration der Netzkomponente zu ermöglichen, müssen die einzelnen Kommandos an die entsprechenden Komponenten weitergereicht

werden können.

F6: Authentifizierungsverwaltung Um die Sicherheit während der Konfiguration gewährleisten zu können, muss die Authentifizierung an der Netzkomponente für den Zugriff verwaltet werden.

2.4.2. Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen beschreiben lediglich die Eigenschaften des Werkzeugs. Sie dienen aber nicht als Beschreibung der Funktionalität.

N1: Erweiterbarkeit Da das Werkzeug generisch eine Vielzahl an verschiedenen CLI Schnittstellen bedienen soll, muss es leicht erweiterbar sein um diesem Anspruch zu entsprechen.

N2: Kommandos pro Position In bestimmten Position innerhalb einer CLI Struktur stellt eine Netzkomponente nur eine gewisse Anzahl an Kommandos zur Verfügung. Um Fehlerfälle während der Konfiguration auszuschließen, dürfen also nur eine gewisse Anzahl Kommandos zur Verfügung stehen, wenn sich die CLI der Netzkomponente in einer bestimmten Position befindet.

Anhand von Literatur werden jetzt in Kapitel 3, die hier formulierten Anforderungen nocheinmal geprüft. Außerdem werden bereits existierende Implementierungen untersucht.

3. Verwandte Arbeiten

Eines der wichtigsten Ziele beim Entwurf von großen Netzen ist die Managebarkeit (z.B. [KDO01][Kap. 9]). Diese wird in fünf Gebiete unterteilt:

1. Konfigurationsmanagement
2. Fehlermanagement
3. Leistungsmanagement
4. Sicherheitsmanagement
5. Abrechnungsmanagement

Auf die Anforderung an das Gebiet des `Konfigurationsmanagements` soll hier kurz eingegangen werden. Eine Anforderung ist die Möglichkeit der Konfiguration aller Komponenten im Netz. Die Umsetzung dieser Aufgabe in einer heterogenen Umgebung setzt ein Werkzeug voraus, das jedes Element konfigurieren kann, unabhängig davon von welchem Hersteller es ist.

In sehr großen Netzstrukturen, zum Beispiel in denen eines Internet Service Provider (ISP), ist eine automatisierte Konfiguration wichtig, da solche Netze sehr starken Veränderungen ausgesetzt sind (siehe [ENCK07]). So können neue oder veränderte Anforderungen eines Endkunden zu einer komplexen Konfigurationsänderung der Netzkomponenten führen. Diese Änderungen gehören zu den täglichen Herausforderungen eines solchen Anbieters. Eine manuelle Konfiguration ist in diesem Anwendungsfall undenkbar. Also muss es ermöglicht werden, die gesamte Netzstruktur automatisiert anzupassen. An diesem Beispiel lässt sich erahnen, dass es bereits eine Vielzahl verschiedener Herangehensweisen an dieses Problem geben muss, zwei davon sollen an dieser Stelle kurz diskutiert werden.

Auf der LISA Konferenz im Jahr 2003 [ABKMP08] wurde eine Studie der Universität von Wisconsin über die Konfiguration von Komponenten in ihrem universitären Netz vorgestellt. Im Rahmen dieser Studie wurde auch eine technische Umsetzung mit dem Namen `Splat` implementiert. Das zu verwaltende Netz bestand in diesem Fall aus über 50 verwaltbaren Netzkomponenten. Des Weiteren wurden die vormals fünfzig bestehenden gerouteten Subnetze durch VLANs abgelöst. Auf zwei der Anforderungen aus diesem Dokument wird an dieser Stelle kurz eingegangen, da sie Anforderungen aus der vorliegenden Bachelorarbeit widerspiegeln.

1. Es sollen einfache Routinen zur Verfügung stehen, die auf allen Komponenten genutzt werden können ohne sich an jeder einzeln anmelden zu müssen (vgl. Anforderung F6, 2.4).
2. Jegliches zur Verfügung stehende VLAN soll auf jedem Switch nutzbar sein (vgl. Anforderung F5, 2.4).

Der erste Punkt spiegelt sich in der Anforderung an eine Authentifizierungsverwaltung wieder. Im nächsten Punkt lässt sich noch eine Anforderung wieder erkennen:

Jedes zur Verfügung stehende VLAN soll auf jedem Switch nutzbar sein. Also muss es eine feste Anzahl an Befehlen geben, die auf den verschiedenen Komponenten den selben Effekt erwirken, nämlich das VLAN nutzbar zu machen. Außerdem müssen die Kommandos, die hierfür nötig sind, auf allen Netzkomponenten ausgeführt werden können.

In einem weiteren Vortrag, der auf der LISA Konferenz 2005 [CL05] vorgestellt wurde, wird ein Beispiel zur Konfiguration über Telnet betrachtet, motiviert durch Fragestellungen in einem universitären Netz der Universität von Waterloo. Es werden Befehle, die in eine Web-Oberfläche eingegeben werden, übersetzt. Diese übersetzten Kommandos werden dann per Telnet und SNMP weitergereicht, was auf den ersten Blick die Wahl der Schnittstelle aus der vorliegenden Bachelorarbeit widerspiegelt. Die Nutzung von Telnet ermöglicht das Ausführen von Kommandos auf der Netzkomponente (vgl. Anforderung F5, 2.4).

Bei beiden oben genannten Ansätzen wird die Konfiguration der Netzkomponenten mit Hilfe von Konfigurationsvorlagen umgesetzt. Solche Vorlagen werden Anwendungs und Geräte spezifisch erstellt. So können Gerätetypen, die dieselbe Schnittstellensyntax aufweisen, mit Hilfe dieser Vorlagen, für ein bestimmtes Szenario konfiguriert werden. Wenn aber eine Umstellung auf andere Netzkomponenten geplant ist, müssen diese Vorlagen angepasst werden, da die Vorlagen eng mit der Software der Komponenten verbunden ist. Eine solche Umstellung kann zum Beispiel durch einen Herstellerwechsel hervor gerufen werden. Außerdem müssen, wenn andere Anwedungsszenarien umgesetzt werden sollen, neue Vorlagen erstellt werden.

Da im Rahmen dieser Bachelorarbeit ein generisches Werkzeug entwickelt wird, muss von der Benutzung solcher Vorlagen abgesehen werden. Das Werkzeug soll nämlich nicht ausschließlich auf ein bestimmtes Szenario angepasst werden. Im Gegenteil soll es mit dem Wissen über die CLI Struktur selbständig die auszuführenden Kommandos ermitteln. Dennoch soll die Erweiterbarkeit gegeben sein, um Syntaxen anderer Schnittstellen nutzen zu können (vgl. Anforderung N2, 2.4).

Solche Vorlagen bestehen aus einer festen Abfolge von Kommandos, die auf dem Endgerät auszuführen sind. Diese Abfolge führt schlussendlich zu der gewünschten Konfiguration. Da von der Nutzung von Vorlagen aber abgesehen wird, muss das Werkzeug selbständig die Position innerhalb der CLI Syntax erkennen und verwalten können, sonst ist eine automatisierte Konfiguration nicht umsetzbar
(vgl. Anforderungen F2 & F3, 2.4).

4. Modellierung der CLI Struktur

Um ein generisches Werkzeug zu schaffen, das der automatisierten Konfiguration über eine CLI Schnittstelle dient, wird zuerst die Struktur der CLI verallgemeinert. Die Verallgemeinerung ermöglicht es, die Struktur der Schnittstelle so abzubilden, dass das Werkzeug diese Abbildung zur automatisierten Konfiguration nutzen kann. Bei dieser Abbildung handelt es sich um ein generisches Modell. Dieses Modell beschreibt die CLI der zu konfigurierenden Netzkomponenten.

4.1. Unterschied zwischen Kommandos und Zuständen

Trotz der verschiedenen CLI Strukturen lassen sich die Positionen innerhalb dieser und deren Übergänge verallgemeinert darstellen. Eine Unterscheidung zwischen Kommandos und Zuständen hilft dabei. Beide werden wie folgt definiert:

Kommandos sind die elementaren Befehle, die ein Endgerät über die CLI Schnittstelle zur Verfügung stellt. Je nach Zustand in dem man sich befindet sind unterschiedliche Befehle ausführbar. Des Weiteren sollen die Kommandos in transitive und intransitive unterschieden werden.

Zustände bezeichnen die verschiedenen Positionen innerhalb der CLI Struktur. Das heißt je nach Position, wird in unterschiedliche Zustände gewechselt. Somit beschreibt ein Zustand genau eine Position innerhalb der CLI Gesamtstruktur.

4.2. Kommandos

Bestimmte Kommandos ermöglichen den Übergang von einem in einen anderen Zustand. Diese werden als **transitive Kommandos** definiert. So wird also die Position innerhalb der CLI Struktur verändert, wenn ein solches Kommando über die Schnittstelle auf dem Endgerät ausgeführt wird.

Andere Kommandos verändern lediglich Eigenschaften des Endgeräts. Sie dienen also dazu an einer bestimmten Stelle innerhalb der CLI Struktur die Konfiguration des Endgeräts zu verändern. Bei diesen sogenannten **intransitiven Kommandos** gibt es keinen Zustandsübergang.

Kommandos pro Zustand

Wie bereits in den Anforderungen 2.4 beschrieben, gibt es pro Zustand nur eine Reihe von verfügbaren Kommandos. Hierbei kann es sich dann natürlich um transitive oder intransitive Kommandos handeln. Im Listing 4.1 erkennt man ein Beispiel für Kommandos, die in diesem Zustand ausführbar sind. Eine Eingabe anderer Kommandos würde zu einem Fehler führen. Bei einigen Herstellern wird die Eingabe eines Kommandos zum falschen Zeitpunkt schlichtweg ignoriert.

In der Regel bieten die CLI Schnittstellen spezielle Kommandos an, um sich eine Liste der zur Verfügung stehenden Kommandos anzeigen zu lassen.

Listing 4.1: Beispiel für eine Kommandoliste

```
catalyst.nm.ifi.lmu.de>?
Exec commands:
  <1-99>           Session number to resume
```

```
  access-enable    Create a temporary Access-List entry
  clear            Reset functions
  connect          Open a terminal connection
  crypto           Encryption related commands.
  disable          Turn off privileged commands
  disconnect       Disconnect an existing network connection
  enable           Turn on privileged commands
  ethernet         Ethernet parameters
  exit             Exit from the EXEC
  help             Description of the interactive help system
  lock             Lock the terminal
  login            Log in as a particular user
  logout           Exit from the EXEC
  mrinfo           Request neighbor and version information from a multicast router
  mstat            Show statistics after multiple multicast traceroutes
  mtrace           Trace reverse multicast path from destination to source
  name-connection  Name an existing network connection
  ping             Send echo messages
  platform         Platform specific command
  rcommand         Run command on remote switch
  release          Release a resource
  renew            Renew a resource
  rep              Resilient Ethernet Protocol Exec Commands
  resume           Resume an active network connection
  set              Set system parameter (not config)
  show             Show running system information
  ssh              Open a secure shell client connection
  systat           Display information about terminal lines
  tclquit          Quit Tool Command Language shell
  telnet           Open a telnet connection
  terminal         Set terminal line parameters
  traceroute       Trace route to destination
  tunnel           Open a tunnel connection
  where            List active connections
```

Ausführen von Kommandos

Das Ausführen eines Kommandos erfolgt in den meisten Fällen durch die Eingabe des Kommandos gefolgt von einem `Carriage Return`. Bei einigen Kommandos müssen aber Werte übergeben werden, die dann das Ergebnis des Kommandos beeinflussen. Im folgenden Beispiel ist `GigabitEthernet 3/43` der Wert, der beim Wechsel in den `Schnittstellenmodus` angibt, welcher Port des Switches konfiguriert werden soll.

Listing 4.2: Port konfigurieren auf Cisco Switch

```
catalyst.nm.ifi.lmu.(config)#interface GigabitEthernet 3/43
catalyst.nm.ifi.lmu.(config-if)#
```

4.3. Zustände

Neben einer festen Anzahl an Kommandos, die ein Zustand liefern kann braucht er aber auch etwas um ihn identifizieren zu können. Hierfür muss die Ausgabe der CLI Schnittstelle erkannt und interpretiert werden. Wie in der Anforderungsanalyse im Abschnitt über die CLI Strukturen 2.2 bereits beschrieben, gibt es zwei verschiedene Ansätze zur Strukturierung der CLI Schnittstellen. Bei einfachen Eingabezeilen kann der String am Anfang der aktuellen Zeile zur Interpretation genutzt werden. Für die Interpretation von Strukturen, die als

Menü aufgebaut sind, kann zum Beispiel die Überschrift des Menüs genutzt werden. Diese muss das Werkzeug in der Ausgabe der Schnittstelle erkennen.

Zum Beispiel ist die CLI des Cisco Switches als einfache Eingabezeile ausgelegt. Hier kann der String `(config-if)#` zur Identifikation genutzt werden. Dieser sagt aus, dass man sich im sog. `Schnittstellenmodus` befindet (siehe Listing 4.2).

4.4. CLI Zustandsmodell

An einem Zustandsmodell lassen sich die Übergänge zwischen den Zuständen mit Hilfe von Kommandos erläutern. Im Listing 4.3 sieht man eine Beispielkonfiguration einer Netzkomponente. In dem Fall handelt es sich um den Cisco Switch aus der Testumgebung. Als Beispiel für eine Konfiguration wird das Abschalten eines bestimmten Switchports beschrieben.

Listing 4.3: Port deaktivieren

```
Shaun:~ shaun$ telnet catalyst.nm.ifi.lmu.de
Trying 141.84.218.239...
Connected to catalyst.nm.ifi.lmu.de.
Escape character is '^]'.


User Access Verification

Password:
catalyst.nm.ifi.lmu.de>enable
Password:
catalyst.nm.ifi.lmu.de#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
catalyst.nm.ifi.lmu.(config)#interface GigabitEthernet 3/41
catalyst.nm.ifi.lmu.(config-if)#shutdown
catalyst.nm.ifi.lmu.(config-if)#exit
catalyst.nm.ifi.lmu.(config)#exit
catalyst.nm.ifi.lmu.de#exit
Connection closed by foreign host.
```

Um den gewünschten Port deaktivieren zu können, müssen verschiedene Schritte ausgeführt werden (vgl. Listing 4.3):

Im ersten Schritt muss die Verbindung zu der Netzkomponente aufgebaut werden. Danach wechselt man nacheinander durch die verschiedenen Modi der Schnittstelle. Der `Nutzer` bzw. `privilegierte Modus` dient lediglich zum Auslesen von Systeminformationen und zur Fehlersuche. Im `Konfigurationsmodus` können systemweite Einstellungen vorgenommen werden, während der `Schnittstellenmodus` zur Konfiguration eines bestimmten oder einer Reihe von Ports des Switches dient. In diesem Modus wird dann mittels des Kommandos `shutdown` die Schnittstelle abgeschaltet. Zum Schluss wird die Verbindung schrittweise beendet.

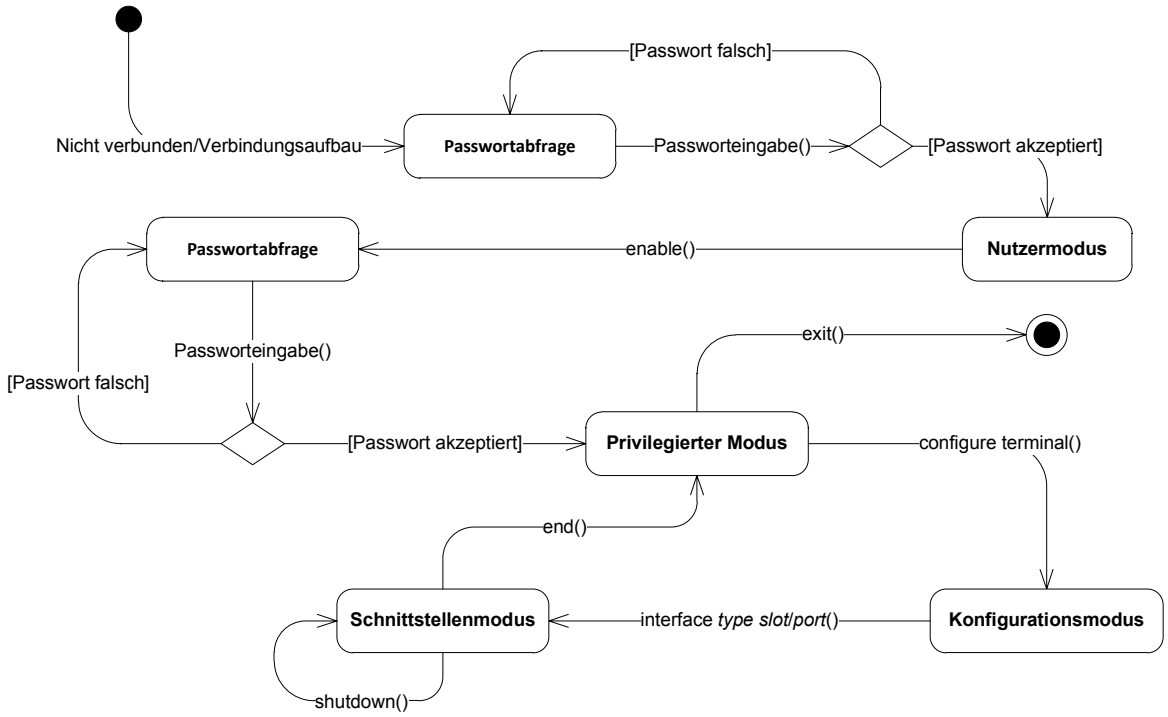

Abbildung 4.1.: Konfigurationsbeispiel

Die Konfiguration aus Listing 4.3 wird in Abb. 4.1 graphisch dargestellt. Wie man erkennt, handelt es sich bei der Untermenge an Kommandos, die für diese Konfiguration benötigt werden, ausschließlich bei dem Kommando `shutdown` um ein intransitives Kommando. Alle anderen Kommandos (z.B. `enable`) dienen dazu um in einen anderen Zustand überzugehen. Die zu durchlaufenden Zustände im Zuge des Konfigurationsbeispiels sind:

1. Passwortabfrage
2. Nutzermodus
3. Passwortabfrage
4. privilegierter Modus
5. Konfigurationsmodus
6. Schnittstellenmodus

Verallgemeinert lässt sich ein Zustandsmodell wie folgt modellieren:

Es existieren eine Reihe von Zuständen und Übergängen. Bei den Übergängen handelt es sich um die transitiven bzw. intransitiven Kommandos. Da für eine Konfiguration immer eine Verbindung aufgebaut werden muss, ist auch das ein Teil der allgemeinen Darstellung. Wenn die Verbindung aufgebaut ist, folgen eine Reihe von Zuständen, die erreicht werden und Kommandos, die ausgeführt werden. Abschließend wird die Verbindung wieder abgebaut.

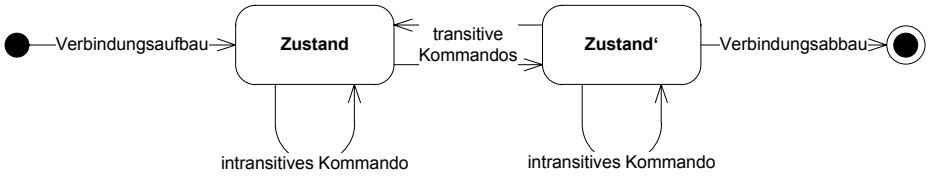

Abbildung 4.2.: allgemeines Zustandsmodell

4.5. Darstellung der CLI Struktur

Um dem zu entwickelten Werkzeug eine Möglichkeit zu bieten, mit der entsprechenden CLI Struktur umgehen zu können, muss diesem eine Darstellung der CLI zur Verfügung gestellt werden. Mit dieser kann es dann die zu verarbeitenden Anfragen umsetzen.

Die eben definierten Kommandos und Zustände lassen sich wie folgt in Klassen darstellen. Die einzelnen

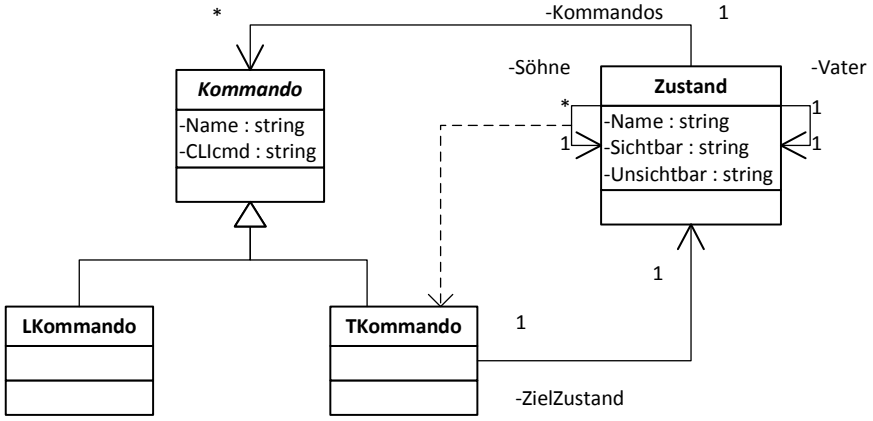

Abbildung 4.3.: Klassendiagramm Zustände/Kommandos

Variablen werden in den folgenden Abschnitten 4.5.1 bzw. 4.5.2 erläutert.

4.5.1. Klasse Zustand

Ein `Zustand` besteht aus einem eindeutigen `Namen`, den Variablen `Sichtbar` und `Unsichtbar` sowie einer Anzahl an `Kommandos`, die dieser Zustand zur Verfügung stellt. Des Weiteren hat ein Zustand einen Vater und eine Menge an Söhnen. Diese Beziehungen dienen dazu um die Verbindungen zwischen den Zuständen auszudrücken.

Zur Ermittlung der Position dient die Ausgabe der Schnittstelle. Hierfür wird entweder der String in der Eingabezeile oder die Überschriften der Menüs genutzt. Die erwartete Ausgabe wird in der Variablen `Sichtbar` hinterlegt und dient für die Identifizierung des aktuellen Zustands.

Die Variable `Unsichtbar` kann Informationen beinhalten, die für den Übergang in diesen Zustand oder innerhalb eines Zustands benötigt werden. Bei diesen Informationen handelt es sich zum Beispiel um Werte, die für bestimmte Kommandos benötigt werden. Das heißt, wenn ein Kommando innerhalb dieses Zustands aufgerufen werden soll, können weitere Werte, die für dieses Kommando benötigt werden hier hinterlegt werden. Das ermöglicht es, Informationen die sich erst während einer laufenden Konfiguration ergeben, hier zwischen zu speichern, damit diese nicht für den weiteren Konfigurationsverlauf verloren gehen.

4.5.2. Klasse Kommando

Ein `Kommando` lässt sich durch eine abstrakte Klasse darstellen, die wiederum einen eindeutigen `Namen` sowie eine Variable `CLIcmd` besitzt, die den Befehl der an das Endgerät weitergegeben werden soll, beinhaltet. Von dieser abstrakten Klasse lassen sich zwei weitere Klassen ableiten, jeweils für ein transitives und ein intransitives Kommando (siehe Abschnitt 4.2); in Abb.4.3 als `LKommando` und `TKommando` bezeichnet.

Ein transitives Kommando braucht noch eine weitere Variable `ZielZustand`, welche den Zustand angibt, in den das Endgerät nach der Ausführung des Kommandos übergeht.

4.6. Navigieren durch die Struktur

Wie bereits erwähnt gehört ein Kommando zu einem Zustand. Soll für eine Konfiguration ein bestimmtes Kommando ausgeführt werden, muss erst in diesen Zustand gewechselt werden. Nämlich genau in diesen, der das auszuführende Kommando zur Verfügung stellt. Um jedoch in diesen Zustand zu gelangen, muss in den meisten Fällen ein transitives Kommando ausgeführt werden, durch welches das Endgerät in den entsprechenden Zustand übergeht. Dieses transitive Kommandos hängt natürlich auch wieder von einem Zustand ab bzw. wird von einem bestimmten Zustand bereit gestellt. Somit lässt sich der Weg von Zustand zu Zustand als Pfad darstellen. Der Pfad beinhaltet genau die einzelnen Zustände, die besucht werden müssen, um das gewünschte Kommando auszuführen.

Die Abarbeitung des Pfades erfolgt dann wieder durch das Ausführen einer Reihe von Kommandos. Es müssen immer die Kommandos ausgeführt werden, die es ermöglichen die Zustände des Pfades zu erreichen. Während dieser Abarbeitung gibt es zwei verschiedene Arten von Übergängen zwischen Zuständen. Einmal gibt es den Übergang von einem Zustand in einen weiteren Zustand; dieser wird durch ein transitives Kommando hervorgerufen. Der Übergang in denselben Zustand passiert durch die Ausführung eines intransitiven Kommandos.

Wenn eine andere Konfiguration erreicht werden soll, muss lediglich ein anderer Pfad durch die Struktur gefunden werden. Also eine andere Aneinanderreihung von Zuständen bzw. eine andere Abfolge von Kommandos. So kann dynamisch auf eine Konfigurationsanforderung reagiert werden, ohne dass die Struktur angepasst werden muss, da lediglich ein anderer Pfad gesucht wird.

4.7. CLI Struktur als Graph

Wie in Abb. 4.1 erkennbar ist, lässt sich die CLI Struktur mit deren Zuständen und Kommandos als Graph darstellen. Der Vorteil bei der dieser Darstellung ist, das durch einen Graphen die benötigte Abfolge von Zuständen bzw. Kommandos als Pfad resultiert. So ist, je nach Konfiguration, ein bestimmter Navigationspfad durch den Graph erforderlich.

Um die Erweiterung der Darstellung zu erleichtern, wird für die Speicherung der CLI Struktur ein XML Schema gewählt. Eine XML Syntax hat den Vorteil, dass sie leicht lesbar ist und damit jederzeit die Möglichkeit gegeben ist einfach Anpassungen vorzunehmen. So kann auf evtl. Änderungen an der CLI Schnittstelle eingegangen werden, indem das zugrunde liegende XML Schema angepasst wird. Außerdem können mit Hilfe von Schema-Definitionen die Richtigkeit der XML Datei überprüft werden.

In Listing 4.4 ist ein Ausschnitt des Schemas zur Beschreibung der Schnittstelle des Switches aus der Testumgebung dargestellt.

Listing 4.4: Schema zur Beschreibung der CLI Struktur des Cisco Switches

```
<!-- state element -->
<!ELEMENT state (stateString, stateDescr, (cmd)*, (state)*)>
    <!ATTLIST state name CDATA #REQUIRED>

    <!ELEMENT stateString (#PCDATA)>

<!-- cmd element -->
<!ELEMENT cmd (cmdCli, (var)*, (cmdDep)?)>
    <!ATTLIST cmd trans (true) #IMPLIED>
    <!ATTLIST cmd name CDATA #REQUIRED>
    <!ATTLIST cmd target CDATA #IMPLIED>

    <!ELEMENT cmdCli (#PCDATA)*>
        <!ATTLIST cmdCli lit CDATA #IMPLIED>
```

Die Definitionen der einzelnen Elemente des XML Schema aus Listing 4.4 werden in der nachfolgenden Aufzählung erläutert.

state Ein Zustand mit entsprechendem Namen als Attribut zum identifizieren. Der Zustand besitzt eine Menge von Kommandos und wiederum Zustände.

stateString beinhaltet die Zeichenkette, die von der Schnittstelle im entsprechenden Zustand ausgegeben wird.

stateDescr Eine Beschreibung des Zustands.

cmd Ein Kommando mit Attributen zur Identifizierung (`name`) und der Auszeichnung, ob es sich um ein transitives Kommando handelt (`trans`). Wenn es sich um ein solches Kommando handelt, steht in `target` der Name des Zustands, in den übergegangen wird. Des Weiteren beinhaltet ein Kommando ein `cmdCli` und evtl. eine `var` Liste bzw. eine `cmdDep` Variable.

cmdCli ist der Befehl, der an die Schnittstelle übergeben wird. Das Attribut gibt die Anzahl der Variablen an, die benötigt werden.

var gibt die Namen der Variablen an, die für den Befehl benötigt werden.

cmdDep beschreibt die Abhängigkeit, da für manche Kommandos das Ergebnis `cmdRes` eines anderen Kommandos `cmdCall` ausschlaggebend ist. Je nachdem ob das Ergebnis erzielt wurde, wird das Kommando selbst oder ein als `cmdFalse` bezeichnetes ausgeführt.

Außerdem lassen sich Variablen der Klassen aus Abb. 4.3 in Listing 4.4 ebenfalls wiederfinden. Die Tabelle 4.1 führt die Zuordnung auf. Eine genauere Beschreibung befindet sich in Kapitel 5.

Klasse		**XML Schema**	
Zustand		ELEMENT state	
	Name : string		ATTLIST state name
	Sichtbar : string		ELEMENT statestring
	Kommandos : Kommando		(cmd)*
	Söhne : Zustand		(state)*
Kommando		ELEMENT cmd	
	Name : string		ATTLIST cmd name
	CLIcmd : string		ELEMENT cmdCli
TKommando	ZielZustand : Zustand		ATTLIST cmd target

Tabelle 4.1.: Zuordnung Klassen zu XML Schema

Im Rahmen dieses Kapitels wurden im ersten Schritt die Unterschiede zwischen Kommandos und Zuständen beleuchtet. Anhand eines Zustandsmodells, welches die Übergänge zwischen den Zustände mit Hilfe der Kommandos beschreibt, wurde der Ablauf einer Konfiguration beschrieben und verallgemeinert. Somit konnte im letzten Schritt die CLI Struktur als Graph beschrieben werden. Mit Hilfe des Graphen, der definierten Klassen und dem Wissen über die Konfigurationsabläufe kann im nächsten Kapitel das geforderte Werkzeug, welches die automatisierte Konfiguration von CLI-basierten Netzkomponenten ermöglicht, implementiert werden.

5. Implementierung als SBI Modul

An dieser Stelle werden die gefundenen Anforderungen aus Kapitel 2.4 und die Erkenntnisse aus dem voran gegangenen Kapitel 4 heran gezogen, um diese in einem Modul für den cAdapt umzusetzen. Bevor jedoch auf die eigentliche Implementierung eingegangen wird wird zuerst noch einmal genauer die im Abschnitt 2.1 bereits vorgestellte bestehende Testumgebung beschrieben.

5.1. Beschreibung Testumgebung

Wie bereits erwähnt soll in der Testumgebung ein FTTH Szenario nachgestellt werden, also die Verbindung von einem netzfähigen Endgerät bei einem Kunden eines ISP zum Internet, über eine Glasfaseranbindung. Hierfür sind natürlich entsprechende Netzkomponenten von Bedarf, die diese Verbindung ermöglichen. Der Aufbau der Testumgebung besteht demnach aus folgenden Komponenten:

1. Cisco Catalyst 4510R
2. VM mit Windows Server 2008R2 und cAdapt
3. VM mit Windows 7 als Testclient
4. VM mit Debian als Testgateway
5. Laptop zur Steuerung von cAdapt

Der Switch wurde von der Firma ckc gestellt. Bei dem Cisco Catalyst Switch handelt es sich um ein Modell einer Baureihe, welches für große Backbones ausgelegt ist. Aufgrund des großen Funktionsumfangs ist er gut für eine solche Testumgebung geeignet.

Die Wahl des Betriebssystems ist bei dem Testclient auf Windows 7 gefallen, da dieses Betriebssystem stark verbreitet ist und somit das FTTH Szenario unterstützt.

Windows Server 2008 R2 bzw. ein Serverbetriebssystem von Microsoft ist durch die Managementsoftware cAdapt vorgegeben. Diese setzt eine Java Glassfish Instanz voraus, welche auf einem Windowsbetriebsystem laufen muss. Außerdem existiert hier bereits ein fundiertes Wissen seitens der Firma ckc bezüglich Installation und Konfiguration des Java Glassfish, auf das zurückgegriffen werden kann. Bei Java Glassfish handelt es sich um einen Anwendungsserver von Oracle. Er ermöglicht es eine Java Anwendung, in diesem Fall cAdapt, als Webanwendung bereit zu stellen. So können über die Webschnittstelle der Anwendung Konfigurationsanfragen an die cAdapt Software gestellt werden.

Die virtuelle Maschine mit installiertem Debian fungiert als Gateway in der Testumgebung. Hier wurde Debian gewählt, da die Konfiguration eines Gateways hier recht leicht umgesetzt werden kann und dieses Betriebssystem wenig Ressourcen benötigt.

Des Weiteren ist innerhalb der Testumgebung noch ein Laptop vorhanden. Auf diesem läuft SoapUI mit dem die cAdapt Managementsoftware angesteuert und somit der Switch konfiguriert werden kann.

5.1.1. Funktionalitäten und Schnittstellen

In der folgenden Abbildung 5.1 ist der Aufbau des Testnetzes dargestellt. Des Weiteren wird im Folgenden auf die Funktionen der einzelnen Komponenten eingegangen. Außerdem werden die Schnittstellen zwischen den einzelnen Komponenten beschrieben.

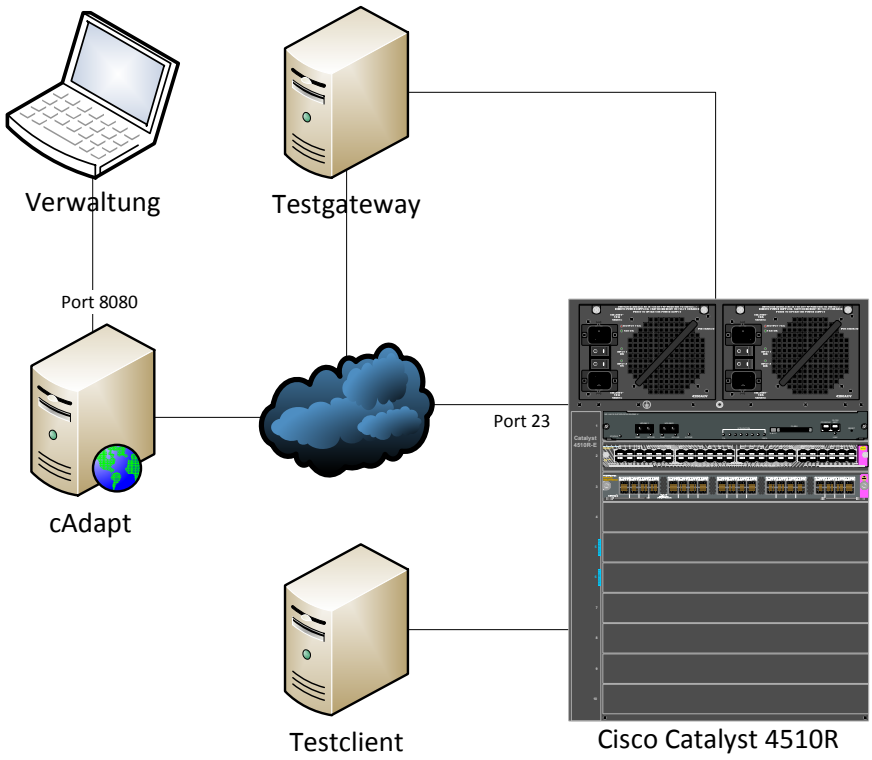

Abbildung 5.1.: Aufbau des Testnetzes

Cisco Catalyst

Der Cisco Switch selbst ist eine zentrale Komponente der Testumgebung. Es handelt sich um einen Catalyst 4510R. Im Switch verbaut sind zwei Module mit je 48 Ports. Das eine Modul hat 48 Steckplätze für Glasfaser Transceiver während das andere Modul 48 normale RJ-45 Anschlüsse besitzt. An dem Switch laufen alle Verbindungen zusammen und die eingesetzte Managementsoftware konfiguriert seine einzelnen Parameter. Er ist sowohl mit dem Testclient als auch mit dem Testgateway verbunden. Sie sind beide jeweils an einem eigenen Switchport angeschlossen. Des Weiteren ist der Switch an das Internet angeschlossen. Der Switch kann somit über den telnet Port 23 vom Internet aus verwaltet werden.

cAdapt Server

Auf einer der virtuellen Maschinen ist ein Windows Server 2008 R2 installiert, worauf ein Java Glassfish Server läuft, welcher die Anwendung cAdapt zur Verfügung stellt. Bei dieser Anwendung handelt es sich, wie bereits in der Anforderungsanalyse 2.3 erwähnt, um eine Software Lösung der ckc AG. Diese Management Software wird eingesetzt, um das implementierte Werkzeug als Modul aufzunehmen und den Switch innerhalb der Testumgebung zu konfigurieren. Zur Konfiguration greift dieser Server über die Management IP Adresse auf den Switch zu.

Windows 7 Testclient

Die virtuelle Maschine mit installiertem Windows 7 simuliert einen an den Switch angeschlossenen Computer wie er in einem beispielhaften Haushalt zu finden ist. Außerdem wird über den Testclient mittels Schicht 3 ICMP Paketen die vorgenommene Konfiguration getestet. Die ICMP Pakete werden bei richtiger Konfiguration des Switches an das entsprechende Ziel weiter gereicht.

Testgateway

Das Testgateway läuft ebenfalls auf einer virtuellen Maschine; hier ist Debian installiert. Diese Maschine spiegelt ein Gateway im FTTH Szenario wieder. Der an den Switch angeschlossene Testclient kann bei richtiger Konfiguration des Switches hierüber auf das Internet zugreifen.

Laptop zur Steuerung

Der Laptop in der Testumgebung dient zur Ansteuerung des cAdapt und ist deshalb direkt mit diesem verbunden. Der Java Glassfish Server bietet eine Schnittstelle auf Port 8080, über den Konfigurationsanfragen mittels SOAP entgegen genommen werden. Zur Erstellung von SOAP Anfragen wird auf dem Laptop SoapUI verwendet. Hierbei handelt es sich um eine Software, die SOAP Templates erstellen kann und mit Hilfe dieser Anfragen erstellen kann.

5.1.2. Switch Konfiguration

Der Catalyst Switch soll das zur Verfügung stehende Netz eines ISP simulieren, an das die einzelnen Haushalte der Endkunden per Glasfaser angeschlossen sind. Damit ein Test des Werkzeuges innerhalb der Testumgebung erfolgreich umgesetzt werden kann, muss bereits eine gewisse Grundkonfiguration vorgenommen werden. So müssen hier entsprechende VLANs eingerichtet werden, über die nachher die Bereitstellung eines ISP Dienstes nachgestellt werden kann. Des Weiteren muss eine Management IP Adresse vergeben werden und der Port, an dem der nachgestellte Haushalt angeschlossen ist, vorbereitet werden. Außerdem muss ein Uplinkport zum nachgestellten ISP Backend konfiguriert werden. Über diesen Uplinkport ist der Switch mit den weiteren Komponenten des ISP Netzes verbunden. In Falle der Testumgebung ist das Testgateway hier angeschlossen, um einen Zugang zum Internet zu ermöglichen. Nachfolgend befindet sich ein Listing mit Ausschnitten aus der Grundkonfiguration des Cisco Switches.

Listing 5.1: Grundkonfiguration des Cisco Switches

```
!
hostname catalyst.nm.ifi.lmu.de
!
ip name-server 141.84.218.30
!
interface FastEthernet2/1
 ip vrf forwarding Mgmt-vrf
 no switchport
 ip address 141.84.218.239 255.255.255.128
 no shutdown
!
define interface-range allports FastEthernet2/25 - 48 , GigabitEthernet3/1 - 48
!
ip default-gateway 141.84.218.254
ip route 0.0.0.0 0.0.0.0 141.84.218.254
```

Diese Punkte der Grundkonfiguration dienen dazu, eine einfache Verbindung zum Netz und dadurch eine Konfiguration zu ermöglichen. Der Switch selbst bekommt einen Namen sowie eine IP eines Nameservers und ein Standardgateway zugewiesen. In Zeile 6-10 des Listings wird einem Port des Switches eine IP Adresse zugewiesen. Die eingerichtete Management IP Adresse dient im weiteren Verlauf zur Konfiguration des Switches. Diese befindet sich, um auch eine Konfiguration vornehmen zu können ohne in direkter Umgebung zum Switch zu sein, in einem öffentlichen IP Adressbereich. Der hier konfigurierte Port dient somit lediglich zur Konfiguration und kann nicht mehr als normaler Switchport genutzt werden. Zur leichteren Ersteinrichtung der anderen Switchports wird in Zeile 12 noch ein Bereich an Ports definiert. Über den hier definierten Bezeichner können gleich mehrere Ports auf einmal konfiguriert werden.

Außerdem müssen noch eine Reihe von VLANs angelegt werden. Nur VLANs, die dem Switch bekannt sind, können im Zuge von weiteren Konfigurationen einem Switchport zugewiesen werden. Über die verschiedenen

VLANs werden bei den ISPs Dienste, wie zum Beispiel `Voice over IP`, bereitgestellt. Somit kann durch die Zuweisung eines VLANs auf einen Switchport ein Dienst einem Endkunden zur Verfügung gestellt werden oder durch die Aufhebung der Zuweisung dieser Dienst wieder genommen werden. Um das jedoch zu ermöglichen, müssen die einzelnen Switchports für die spätere Konfiguration vorbereitet werden. Generell gibt es im Bezug auf VLANs zwei verschiedene Arten von Ports. Ein Port, der sich im sog. Trunkmode befindet, leitet Pakete, die für ein VLAN vorgesehen sind, weiter. Solche Ports sind meist zwischen Switchen konfiguriert. In dem nachfolgenden Ausschnitt ist der Port **FastEthernet2/25** als ein solcher Port vorkonfiguriert. Bei Port **FastEthernet2/24** handelt es sich um einen Accessport, bei dem Pakete, die über diesen entgegengenommen werden, einem VLAN zugeordnet werden. In diesem Beispiel, werden Pakete dem VLAN 600 zugewiesen. Als Accessport sind in der Regel die Ports konfiguriert, an denen lediglich Endgeräte hängen.

Listing 5.2: VLAN Konfiguration des Cisco Switches

```
!
vlan 600
 name EXAMPLE
!
vlan 700
 name EXAMPLE_1
!
!
interface FastEthernet2/24
 switchport access vlan 600
!
interface FastEthernet2/25
 switchport mode trunk
 switchport nonegotiate
!
```

5.1.3. Ablauf einer Konfigurationsanfrage

Um die Konfiguration des Switches durchzuführen, müssen innerhalb der Testumgebung folgende Schritte erfolgen. Eine Darstellung eines solchen Ablaufs ist in Abb. 5.2 zu sehen.

1. Konfigurationsanfrage übermitteln

Als erstes wird entsprechend der umzusetzenden Konfiguration eine Anfrage mittels SoapUI auf dem Laptop innerhalb der Testumgebung erstellt. Diese wird über den Port 8080 an den cAdapt Server weitergereicht.

2. Verarbeitung der Anfrage

Die Anfrage wird von der cAdapt Instanz auf dem virtuellen Server entgegengenommen. Hier wird dann mit Hilfe des implementierten Moduls die Konfiguration der Cisco Switches vorgenommen. Einen genaueren Ablauf des Programms befindet sich im Abschnitt 5.2.2 in diesem Kapitel. Die Konfiguration wird mittels einer telnet Verbindung durchgeführt, welche über Port 23 zwischen dem cAdapt Server und dem Switch aufgebaut wird.

3. Rückgabe des Resultats

Je nach Ergebnis der Konfiguration wird von der cAdapt Software eine Rückgabe erzeugt, die an das auf dem Laptop laufenden SoapUI weiter gegeben wird.

4. Testen der Konfiguration

Um die Konfiguration, die erzielt werden sollte, zu testen, wird von dem Testclient mittels Ping, also mit Schicht 3 ICMP Paketen, die Verbindung zum Internet über das Testgateway überprüft. Parallel wird manuell per telnet die laufende Konfiguration des Switches überprüft, ob die erforderlichen Änderungen erfolgreich vorgenommen wurden.

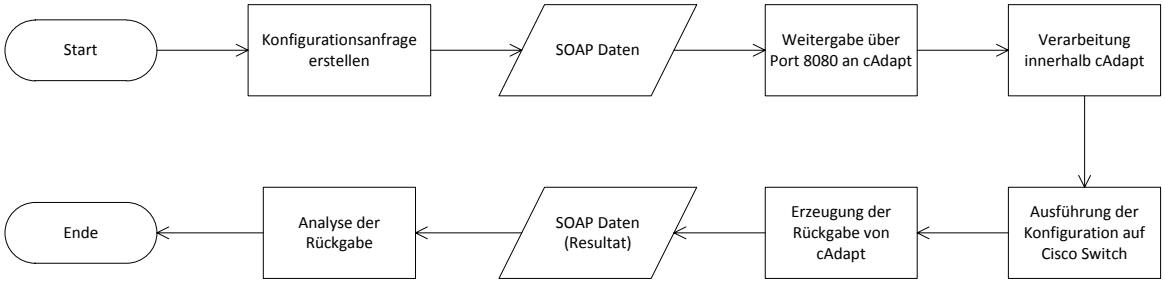

Abbildung 5.2.: Ablauf der Konfiguration

5.2. SBI Modul

Im Folgenden wird die Implementierung des zu entwickelnden Werkzeuges als SBI Modul bezeichnet. Es wird als Modul des Southbound Interfaces der cAdapt Management Software eingebunden.

Eine Konfiguration innerhalb der Management Software läuft wie folgt ab. Die Konfiguration wird an das sog. Northbound Interface der Software gestellt. Hierzu stellt der cAdapt Server auf Port 8080 eine SOAP Schnittstelle zur Verfügung, die die Konfigurationsanfragen nach MTOSI entgegen nimmt. Diese Anfragen werden dann durch den Request Mapper auf eine Reihe von Aufrufen übersetzt. Hierzu hat der Request Mapper Zugriff auf eine Reihe von XML Dateien. In diesen XML Dateien werden die Aufrufe aus dem Northbound Interface in ein oder mehrere Befehle umgewandelt. Außerdem werden Werte aus den Aufrufen bestimmten Variablen zugewiesen. Diese Befehle werden an das Modul des Southbound Interfaces weitergereicht. Hier wird dann je nach Befehl und evtl. übergebenen Variablen, die Konfiguration der Netzkomponente über telnet vorgenommen. Eine Übersicht der Schnittstellen findet sich in Abb. 5.3.

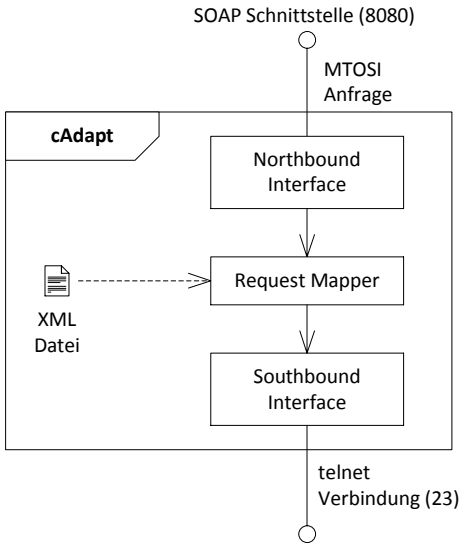

Abbildung 5.3.: Schnittstellen des cAdapt

Der in Java geschriebene cAdapt ermöglicht es über eine Interface Klasse `ISBInterface` die verschiedenen Module anzusprechen. Dieses Interface muss von dem entsprechenden Modul genutzt werden. Die Interface Klasse setzt die Nutzung folgender Grundfunktionen voraus:

getVersion() gibt die aktuelle Version des Moduls zurück.

getID() gibt eine eindeutige Modul ID zurück.

getName() gibt den Namen des Moduls zurück.

process(IAdapterRequest request) ist die Funktion, die bei der Konfiguration aufgerufen wird. Ihr wird ein IAdapterRequest übergeben, der alle benötigten Werte zur Konfiguration übergibt. Die eigentliche Funktionalität des SBI Moduls wird in dieser Funktion abgewickelt.

Neben den oben beschriebenen, vorgegebenen Grundfunktionen mussten noch weitere eigene Variablen und Funktionen definiert werden, die zur Implementierung des Werkzeugs benötigt wurden. Das nachfolgende Listing 5.3 ist ein Ausschnitt aus dem Code des SBI Moduls (vgl. Listing A.1) mit diesen Variablen, sowie den Rümpfen der implementierten Funktionen. In den Zeilen 3 bis 5 werden Variablen definiert und bekommen entsprechende Werte zugewiesen anhand derer das Modul und seine Version identifiziert werden kann. Hierbei handelt es sich um eine Vorgabe des cAdapt. Die Variablen aus den Zeilen 7 bis 14 werden innerhalb des SBI Moduls benötigt:

stateTree beinhaltet den Graphen der die CLI Struktur beschreibt.

cmdPath ist der Stack auf dem der Pfad durch den Graphen gespeichert wird.

varList enthält optionale Variablen die für die Ausführung der CLI Befehle benötigt werden.

device beinhaltet eine Reihe von Werten die für den Verbindungsaufbau zu der zu konfigurierenden Netzkomponente benötigt werden.

connection stellt die telnet Verbindung zur Verfügung.

execCmd ist der für die gewünschte Konfiguration auszuführende Befehl.

pathToJar enthält den Pfad an welcher die JAR Datei des Moduls liegt. Dieser Pfad wird für das Öffnen einer Datenbank benötigt.

responseImpl ist eine Instanz einer Klasse die vom cAdapt vorgegeben ist und die die Rückgabe des SBI Moduls beinhaltet.

Die Funktionen aus den Zeilen 29 ff, 35 ff, 39 ff und 43 ff sind die oben beschriebenen Grundfunktionen der Interface Klasse `ISBInterface`. Die Funktion `getPath` aus Zeile 46 ff dient dazu, um den Pfad zu dem auszuführenden Kommando durch den Graphen zu finden. Hierzu wird eine Hilfsfunktion aus der Klasse `StateTree` genutzt, die jeweils den Pfad zu dem gesuchten CLI Befehl und den abschließend Pfad zu einem Verbindungsabbau bildet. Dieser Pfad wird dann mit den evtl. benötigten Variablen gefüllt und auf einem Stack zur späteren Abarbeitung gespeichert. Mit der Funktion `execPath`, die in Zeile 50 beginnt wird der zuvor gesuchte Pfad ausgeführt, hierzu wird zuerst der aktuelle Zustand des CLI Graphen auf deren Wurzel gesetzt. Während die Konfiguration vorgenommen wird und die einzelnen CLI Befehle aus dem Stack abgearbeitet werden, wird der Zustand des CLI Graphen laufend aktualisiert. So kann die Ausgabe der CLI mit der im Graphen hinterlegten Referenz verglichen werden. Das stellt zusätzlich sicher, dass die CLI in den richtigen Zustand gewechselt ist. Die Ausgabe der CLI und die Ausführung der Befehle wird über die, in der Variablen `telnet` hinterlegten, telnet Verbindung ermöglicht.

Listing 5.3: Funktionsrümpfe des SBIModules

```
1  public class SouthBoundCLIImpl implements ISBInterface {
2
3    public static final double VERSION = 1.0d;
4    public static final Long ID = 0L;
5    public static final String SBI_NAME = "CLI";
6
7    private StateTree stateTree;
8    private Stack<cmd> cmdPath;
```

```
9     private List<cliVar> varList;
10    private Device device;
11    private TelnetConnect connection;
12    private String execCmd;
13    private String pathToJar;
14    private AdapterResponseImpl responseImpl;
15
16    private Logger LOGGER;
17
18    @Inject
19    LoggingService loggingService;
20
21    @Inject
22    ConfigurationService confService;
23
24    @PostConstruct
25    void init(){
26    }
27
28    @Override
29    public IAdapterResponse process(IAdapterRequest request)
30      throws RequestTypeNotSupportedException,
31      NotEnoughParameterException, IllegalParameterValueException {
32    }
33
34    @Override
35    public String getName() {
36    }
37
38    @Override
39    public Long getID() {
40    }
41
42    @Override
43    public double getVersion() {
44    }
45
46    private boolean getPath(String command, List<cliVar> varList)
47      throws NotEnoughParameterException{
48    }
49
50    private boolean execPath(){
51    }
52  }
```

5.2.1. Kommunikation zu dem Switch

Die Verbindung zur Netzkomponente wird mittels dem Netzprotokoll telnet hergestellt, da für telnet bereits viele Implementierungen existieren und eine Vielzahl an Netzkomponenten dieses Protokoll ohne vorhergehende Konfiguration unterstützen. Beim SBI Modul wird der Telnet Client [APACH01] der Apache Foundation genutzt. Hier existiert eine gut beschriebene Dokumentation der Funktionen, was bei der Implementierung von Vorteil ist.

Listing 5.4: Variablen in Klasse Device

```
class Device {
    private String ident;
    private String type;
    private String cliVar;
```

```
    private String user;
    private String password;
    private Integer port;
    private long timeout;
    private long wait;
```

Die für die Verbindung erforderlichen Parameter werden in einer Instanz der Klasse `Device` hinterlegt (siehe Listing 5.4). Hierbei handelt es sich zum einen um einen String `ident`, welcher die zu konfigurierende Netzkomponente eindeutig identifiziert. Da es sich um eine Netzkomponente handelt wird hierzu deren Management IP Adresse gewählt. Außerdem gibt es noch einen Typenbezeichner `type` und einen String, der als Variable in den auszuführenden CLI Befehlen genutzt werden kann (siehe Abschnitt 5.2.2). Des Weiteren werden hier noch Benutzername und Passwort (`user` bzw. `password`) für eine evtl. Authentifizierung angegeben. Zuletzt werden noch drei Werte, die für den Verbindungsaufbau ausschlaggebend sind, gespeichert. Die Variablen `timeout` und `wait` dienen einerseits dazu eine Verbindung die nicht mehr reagiert zu beenden und andererseits dazu die Reaktionszeit der Netzkomponente zu definieren.

Die telnet Verbindung selbst wird mit Hilfe der Klasse `TelnetConnect` (Listing A.2) verwaltet. Wie an den Funktionsrümpfen im Listing 5.5 ersichtlich besitzt diese Klasse je eine Funktion, die mit Hilfe des TelnetClients der Apache Foundation eine Verbindung aufbaut bzw. beendet (Zeilen 8 und 11). Des Weiteren existiert noch eine Funktion, die die Befehle auf der CLI Schnittstelle ausführt (`sendCommand` Zeile 14). Die Funktion `getResult` ab Zeile 17 liest die Rückgabe der CLI Schnittstelle aus.

Listing 5.5: Funktionsrümpfe TelnetConnect Klasse

```
1  class TelnetConnect {
2    private OutputStream output;
3    private TelnetClient client;
4    private long timeout;
5
6    static Logger LOGGER = Logger.getLogger(TelnetConnect.class.getName());
7
8    public boolean connect(Device device){
9    }
10
11   public void disconnect(){
12   }
13
14   public boolean sendCommand(String command){
15   }
16
17   public String getResult(){
18   }
19 }
```

5.2.2. Programmablauf

Eine Konfigurationsanfrage an das SBI Modul läuft wie folgt ab (vgl. Abbildung 5.4):

Anhand der IP Adresse, der zu konfigurierenden Netzkomponente, werden aus einer Datenbank zusätzlich benötigte Informationen ausgelesen. In dieser Datenbank sind für die jeweilige Komponente Werte hinterlegt. Da der cAdapt auf einem Windows Server Betriebssystem betrieben wird, liegen diese Daten in einer Microsoft Access Datenbank. Außerdem bietet Microsoft Access den Vorteil, dass die Daten leicht über eine graphische Oberfläche gepflegt und erweitert werden können.

Nachfolgend werden die einzelnen Werte der Datenbank kurz beschrieben:

user Falls für die Konfiguration eine Authentifizierung mittels Benutzernamen benötigt wird enthält diese Variable den entsprechenden Namen.

password Das benötigte Passwort für eine Authentifizierung ist hier hinterlegt.

port Wenn die Netzkomponente einen anderen Port als den Standard telnet Port (23) verwendet, wird dieser hier angegeben.

wait/timeout Hierbei handelt es sich um zwei Werte, die für die telnet Kommunikation benötigt werden; wie in Abschnitt 5.2.1 beschrieben.

cliVar Wie bereits in Kapitel 4.2 beschrieben, müssen für manche Kommandos weitere Parameter angegeben werden. Diese könnten z.B. der zu konfigurierende Port oder das VLAN, das zugewiesen werden soll, sein. Damit innerhalb des Kommandos der Parameter an der richtigen Position eingefügt wird, muss im Kommando eine Variable hinterlegt sein. Da aber der String, welcher als Variable genutzt wird, vielleicht als Steuerzeichen innerhalb der Netzkomponente genutzt werden könnte, muss die Möglichkeit gegeben sein, diesen für jede Netzkomponente unterschiedlich zu definieren. Das geschieht über den Wert `cliVar`.

Anhand eines Strings, der neben der IP Adresse auch mit übergeben wird, wird im nächsten Schritt eine XML Datei eingelesen. Dieser String gibt an um welchen Typ von CLI Schnittstelle es sich bei der Netzkomponente, die angesprochen werden soll, handelt. In der XML Datei ist die Struktur der CLI Schnittstelle beschrieben, so, wie sie in Kapitel 4.7 definiert wurde.

Im folgenden Schritt wird die XML Datei als Graph eingelesen und, wie in Kapitel 4.6 beschrieben, ein Pfad durch den Graphen gesucht. Dieser Pfad führt von einem Verbindungsaufbau zu dem für die Konfiguration notwendigen auszuführenden Kommando. Wenn dieses Kommando gefunden ist, wird der Pfad zum Verbindungsabbau gesucht. Somit enthält der Pfad alle Kommandos, die notwendig zur Umsetzung der Konfiguration sind. Der gesamte Pfad wird in einem Stack zwischen gespeichert. Falls das Kommando nicht im Graph enthalten ist, wird ein Fehler, dass das auszuführende Kommando nicht gefunden wurde, zurück gegeben und die Ausführung des SBI Moduls beendet.

Wenn die Pfadsuche ohne Fehler durchgeführt wurde, werden die einzelnen Kommandos, die sich auf dem Stack befinden, abgearbeitet. Hierdurch wird die Netzkomponente in die entsprechenden Zustände überführt. Anhand der Rückgabe der CLI Schnittstelle wird geprüft, ob der zu erwartende Zustand erreicht wurde. Wenn während der Abarbeitung der Kommandos ein Zustand nicht entsprechend identifiziert werden kann, wird ein Fehler erzeugt und die Ausführung des Moduls abermals beendet. Ist die Konfiguration ohne Fehler erfolgt wird eine positive Rückmeldung erzeugt.

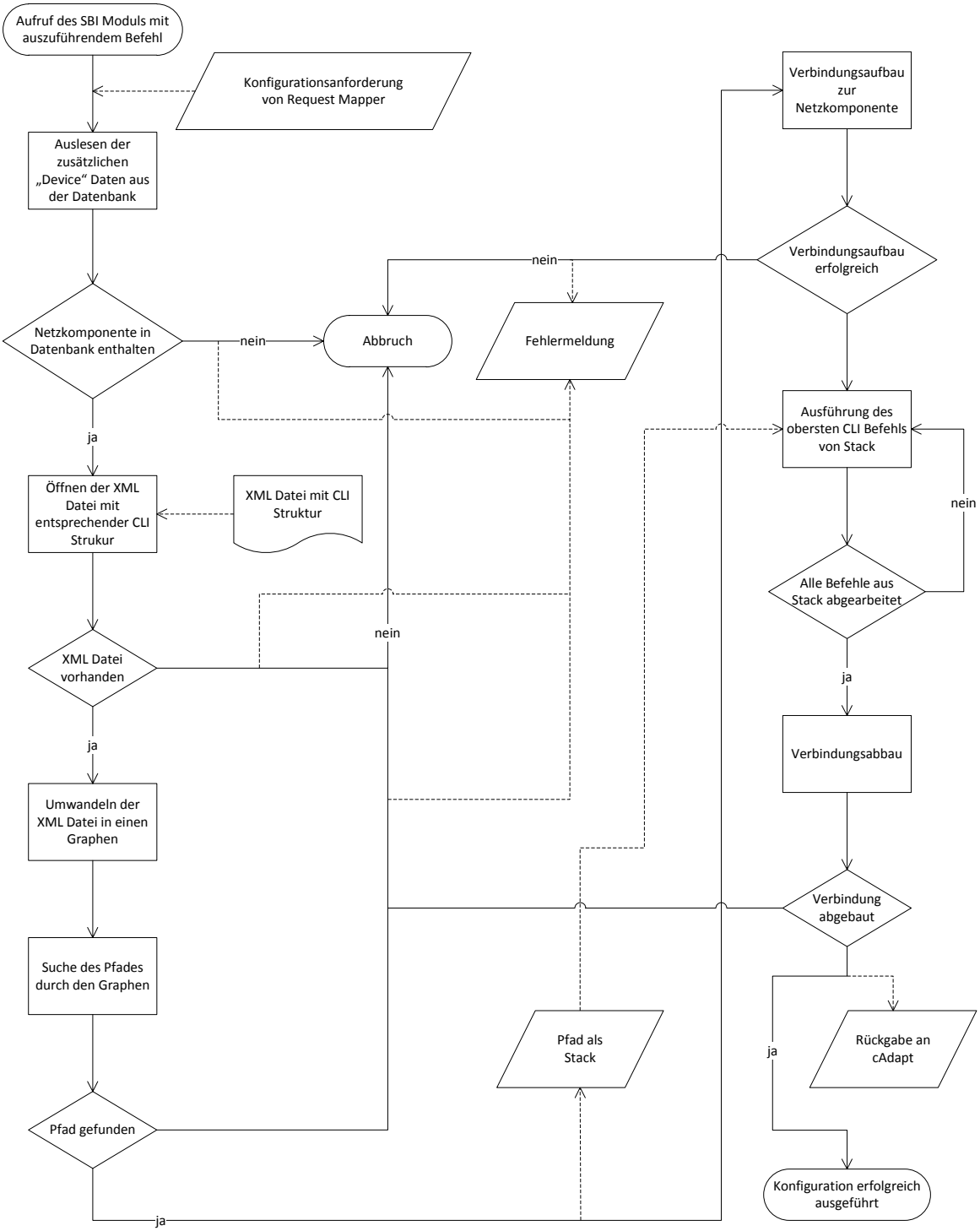

Abbildung 5.4.: Programmablauf

5.3. Evaluation

Im Folgenden werden die Anforderungen an das zu implementierende Werkzeug aus Kapitel 2.4 noch einmal aufgegriffen und analysiert, ob diese durch die Implementierung tatsächlich umgesetzt wurden.

F1: Empfangen der Befehle vom Management Adapter Wie in Abschnitt 5.2 beschrieben wird das Werkzeug als Modul in den cAdapt eingebunden. Hierdurch kann das Werkzeug direkt als Funktion aufgerufen werden.

F2: Ermittlung der Position anhand der CLI Ausgabe Die Funktion `getResult` liest, wie in Abschnitt 5.2.1 beschriebenen, die Ausgabe der CLI aus. Mit Hilfe der im Abschnitt 4.5.1 beschriebenen Variablen `Sichtbar` wird die Ausgabe dann verifiziert.

F3: Verwaltung der Positionen Die Verwaltung der Position innerhalb der CLI Struktur wird durch den in Abschnitt 4.7 Graphen erleichtert. Hierdurch kann eine Navigation durch die Struktur einfach als Pfad dargestellt werden und in einem Stack gespeichert schrittweise abgearbeitet werden. Außerdem kann mit Hilfe des Graphen der jeweilige Zustand, in dem sich die CLI Struktur befindet, evaluiert werden.

F4: Erkennung von Fehlerfällen Je nach Ausgabe der CLI wird ein Fehler zurück gegeben, wenn die erwartete Ausgabe nicht erfolgt ist (vgl. Abschnitt 5.2.2).

F5: Ausführen von Kommandos auf Netzkomponente Um die Kommandos auf der Netzkomponente ausführen zu können, wird, wie in Abschnitt 5.2.1 beschrieben, eine Verbindung mittels telnet aufgebaut.

F6: Authentifizierungsverwaltung Die Authentifizierung wird anhand von in einer Datenbank gespeicherten Datensätzen ermöglicht (siehe Abschnitt 5.2.2).

N1: Erweiterbarkeit Durch die im vorigen Punkt erwähnte Datenbank und die CLI Struktur beschreibenden XML Dateien können leicht weitere Netzkomponenten unterstützt werden.

N2: Kommandos pro Position Die Anzahl der Kommandos, die von einer Position zur Verfügung gestellt werden, ist durch die Struktur der XML Datei, die die CLI Struktur beschreibt, abgebildet (siehe Abschnitt 4.7).

Es wurde ein Modul für den ckc cAdapt entwickelt welches die an ihn gerichteten Konfigurationsanfragen auf dem zu konfigurierenden Switch automatisiert umsetzen kann. Dazu identifiziert das Modul die einzelnen Positionen innerhalb der CLI Struktur anhand von Strings in der Ausgabe. Durch diese Identifizierung und eine Reihe von Kommandos ist es möglich die Konfiguration schrittweise umzusetzen. Die Reihe von auszuführenden Kommandos wird durch einen Pfad durch den Graphen, der die CLI Struktur beschreibt, ermittelt.

Um eine fehlerfreie Konfiguration zu gewährleisten wird parallel die erwartete Position innerhalb der CLI Struktur ermittelt, damit, wenn ein Kommando ausgeführt wurde, geprüft werden kann ob die richtige Position erreicht ist bevor das nächste Kommando ausgeführt wird.

Die Kommunikation zu dem Switch erfolgt mittels telnet. Hier ist zwar eine Authentifizierung gewährleistet, die Kommunikation selber verläuft aber unverschlüsselt. Da die Kommunikation aber lediglich innerhalb des gesicherten Netzes eines ISP ablaufen würde, ist eine verschlüsselte Kommunikation nicht unbedingt nötig.

Die Funktionalität des Moduls wurde nach der Implementierung erfolgreich innerhalb einer Testumgebung evaluiert. Eine Konfiguration konnte ohne Probleme automatisiert umgesetzt werden.

6. Zusammenfassung und Ausblick

Abschließend wird an dieser Stelle das Resultat der vorliegenden Bachelorarbeit noch einmal beleuchtet. Als erstes wird die im letzen Kapitel durchgeführte Evaluation 5.3 noch einmal, mit Blick auf die Fragestellung, aufgegriffen und weitere Erkenntnisse aus der Arbeit zusammengefasst.

Das Ziel der vorliegenden Bachelorarbeit war es, ein generisches Werkzeug zur Automatisierung der CLI-basierten Konfiguration von Netzkomponenten zu entwickeln. Die hierzu erstellten Anforderungen (siehe Kapitel 2) wurden, wie im Abschnitt 5.3 erkennbar, erfüllt. So macht das entwickelte Werkzeug es möglich eine Netzkomponente über eine CLI Schnittstelle zu konfigurieren.

Hierfür wird lediglich das Wissen über die Struktur der CLI Schnittstelle vorausgesetzt. Dieses sollte bei der Konfiguration einer Netzkomponente sowieso vorhanden sein. Außerdem ist es in der Regel von den Herstellern gut dokumentiert. Es besteht jedoch das Problem, dass die einzelnen Positionen innerhalb der gesamten CLI Struktur lediglich an Hand der Zeichenketten, die von der CLI Schnittstelle ausgegeben werden, erkannt werden können. So könnte es durchaus bei Updates der Firmware dazu kommen, dass sich die zur Identifizierung genutzten Zeichenketten ändern und somit eine Ermittlung der Position verhindert werden könnte. Diese Zeichenketten können aber ohne Probleme, bei anstehenden Updates der Firmware in den entsprechenden XML Dateien, die die CLI Struktur beschreiben, angepasst werden.

Die Konfiguration selbst wird dann mittels des Pfades durch den Graphen, welcher die CLI Struktur beschreibt, ermöglicht. Da die CLI Struktur und somit der resultierende Graph die Menge aller möglichen Kommandos der Schnittstelle beinhaltet, kann somit auch der gesamte Konfigurationsumfang einer Netzkomponente genutzt werden. Bei der Konfiguration über andere Management Protokolle kann hingegen meist nur eine gewisse Teilmenge des Konfigurationsumfanges angesprochen werden. So ist es bei manchen Komponenten zum Beispiel der Fall, dass ein vorhandenes Webinterface, welches zur Konfiguration genutzt werden kann, nur ein Bruchteil der Funktionen dieser Komponente konfigurieren kann. Also ist man nicht auf einen bestimmten Anwendungsfall festgelegt und muss nicht für die verschiedensten Konfigurationsanforderungen, die sich ergeben, diverse Werkzeuge einsetzen.

6.1. Weiterführende Themen

Schlussendlich folgt noch ein Ausblick auf Themen, die auf die vorliegende Bachelorarbeit aufbauen könnten.

Das entwickelte Werkzeug dient lediglich zur Konfiguration einer Netzkomponente. Es ist jedoch auch vorstellbar eine gleiche Herangehensweise zum Auslesen der Netzkomponenten zu nutzen.

So könnte man zum Beispiel die CLI Struktur, die von dem entwickelten Werkzeug genutzt wird, auslesen und somit die XML Dateien, welche die Struktur beschreiben, erstellen lassen. Das hätte den Vorteil, dass eine Vielzahl an Netzkomponenten ausgelesen werden und die zur Konfiguration notwendigen XML Dateien automatisiert erstellt werden können.

Außerdem könnte man neben der CLI Struktur auch einzelne Parameter auslesen. Die so ausgelesenen Daten könnte man dann zur Überwachung von Fehlern, Performance oder der Sicherheit von Netzkomponenten heranziehen, da oftmals die bestehenden Möglichkeiten zur Überwachung von Netzkomponenten für solche Anwendungsgebiete nicht vollkommen ausreichen. So würde man die Gesamtheit der Daten, welche eine Netzkomponente zur Verfügung stellt, erhalten.

Des Weiteren wurde bei der Umsetzung des zu entwickelnden Werkzeuges die Management Software cAdapt als Grundlage genutzt. Diese ruft bei einer Konfigurationsanfrage jeweils pro auszuführenden Teilbefehl das Werkzeug auf, welches dann die Verbindung zu der Netzkomponente aufbaut, die nötigen Kommandos

ausführt und die Verbindung wieder trennt. Somit müssen für eine Konfigurationsanfrage mit mehreren Teilbefehlen auch mehrfach die Verbindung auf- und abgebaut werden. Außerdem muss, da das Werkzeug nach jedem Aufruf terminiert, die CLI Struktur wiederholt eingelesen werden. Wenn man aber das Werkzeug in einer anderen Management Software einsetzt bzw. die Implementation entsprechend erweitert, könnte man eine Reihe von Konfigurationen durchführen und den Verbindungsauf- bzw. abbau minimieren. Das Werkzeug müsste dann lediglich den Pfad zu dem nächsten Kommando suchen und diesen ausführen.

A. Anhang

Listing A.1: SBIModul Klasse

```
public class SouthBoundCLIImpl implements ISBInterface {

  public static final double VERSION = 1.0d;
  public static final Long ID = 0L;
  public static final String SBI_NAME = "CLI";

  private StateTree stateTree;
  private Stack<cmd> cmdPath;
  private List<cliVar> varList;
  private Device device;
  private TelnetConnect connection;
  private String execCmd;
  private String pathToJar;
  private AdapterResponseImpl responseImpl;

  private Logger LOGGER;

  @Inject
  LoggingService loggingService;

  @Inject
  ConfigurationService confService;

  @PostConstruct
  void init(){
    File file =
      new File(
        SouthBoundCLIImpl.class.getProtectionDomain().getCodeSource()
                .getLocation().getFile()
      );
    pathToJar = file.getAbsolutePath();
    responseImpl = new AdapterResponseImpl();
    LOGGER = loggingService.getSBILogger(this);
  }

  @Override
  public IAdapterResponse process(IAdapterRequest request)
    throws RequestTypeNotSupportedException,
    NotEnoughParameterException, IllegalParameterValueException {
    /*
     *  Funktionalität des SBI Moduls
     */
  }

  @Override
  public String getName() {
    return SouthBoundCLIImpl.SBI_NAME;
  }

  @Override
```

```
51    public Long getID() {
52      return SouthBoundCLIImpl.ID;
53    }
54
55    @Override
56    public double getVersion() {
57      return SouthBoundCLIImpl.VERSION;
58    }
59
60    private boolean getPath(String command, List<cliVar> varList)
61      throws NotEnoughParameterException{
62      /*
63       *  Funktion zur Findung des Pfades durch den Graphen
64       */
65      Stack<cmd> tmpPath = new Stack<cmd>();
66
67      if(!stateTree.getCmdPath(tmpPath, command)){
68        LOGGER.warning("Command not in tree");
69        return false;
70      }
71
72      while(!tmpPath.isEmpty()){
73        cmd tmpCmd = tmpPath.pop();
74        if(tmpCmd.getLit()!=0){
75          if(varList.size()-1>=tmpCmd.getLit()){
76            String tmpCli = varList.get(
77                        varList.indexOf(new cliVar("cli",""))
78                  ).getValue();
79            for(String tmpArg: tmpCmd.getArgList()){
80              try{
81                tmpCmd.setCli(tmpCmd.getCli().replaceFirst(tmpCli,
82               varList.get(varList.indexOf(new cliVar(tmpArg,""))).getValue())
83                );
84              }catch(ArrayIndexOutOfBoundsException e){
85                LOGGER.warning("ERROR Missing variable: "+e.getMessage());
86                throw new NotEnoughParameterException("There is a parameter"+
87                  " missing for command: "+tmpCmd);
88              }
89            }
90          }else{
91            LOGGER.warning("ERROR Invalid no of parameters");
92            throw new NotEnoughParameterException("There are not enough"+
93                  " parameters for command: "+tmpCmd);
94          }
95        }
96        cmdPath.push(tmpCmd);
97      }
98      return true;
99    }
100
101   private boolean execPath(){
102     /*
103      *  Ausführung des Pfades
104      */
105     stateTree.setActual(stateTree.getRoot());
106
107     while(!cmdPath.isEmpty()){
108       cmd tmpCmd = cmdPath.pop();
109       boolean gotRes = false;
110       String tmpCli = "";
111
```

```
112         for(int i=0;i<3;i++){
113           tmpCli = connection.getResult();
114           if(tmpCli.contains(stateTree.getActual().getString())){
115             gotRes = true;
116             break;
117           }
118         }
119         if(gotRes){
120           if(!connection.sendCommand(tmpCmd.getCli())){
121             LOGGER.warning("Sending cmd to telnet went wrong");
122             return false;
123           }
124         }else{
125           LOGGER.warning("ERROR in CLI: Got:"+tmpCli+" ;Need:"+
126                   stateTree.getActual().getString());
127           return false;
128         }
129         if(!tmpCmd.getCmdState().equals("")){
130           //check if actual state has expandation and has son=next state
131           //if so go to son
132           if(stateTree.getActual().getExpand() &&
133             stateTree.getActual().isStateSon(tmpCmd.getCmdState())){
134             stateTree.setActual(stateTree.getActual().getStateSon(
135                         tmpCmd.getCmdState())
136                   );
137             //check if next state is parent of actual
138           }else{
139             while(stateTree.getActual() != stateTree.getRoot()){
140               if(stateTree.getActual().getIdent().equals(tmpCmd.getCmdState())){
141                 break;
142               }else{
143                 stateTree.goUp();
144               }
145             }
146           }
147         }
148         try {
149           Thread.sleep(device.getWait());
150         } catch (InterruptedException e) {
151           LOGGER.warning(e.getMessage());
152           return false;
153         }
154       }
155       return true;
156     }
157 }
```

Listing A.2: TelnetConnect Klasse

```
1  package de.ckc.SoutBound.CLI;
2
3  import java.io.*;
4  import java.net.SocketException;
5  import java.util.logging.Logger;
6  import org.apache.commons.net.telnet.TelnetClient;
7
8  class TelnetConnect {
9    private OutputStream output;
10   private TelnetClient client;
11   private long timeout;
12
13   static Logger LOGGER = Logger.getLogger(TelnetConnect.class.getName());
14
15   public boolean connect(Device device){
16     this.client = new TelnetClient();
17
18     if(device.getIdent().equals("")){
19       LOGGER.warning("No device IP given.");
20       return false;
21     }
22     try {
23       this.client.connect(device.getIdent(),device.getPort());
24     } catch (SocketException e) {
25       LOGGER.warning(e.getMessage());
26       return false;
27     } catch (IOException e) {
28       LOGGER.warning(e.getMessage());
29       return false;
30     }
31     this.output = this.client.getOutputStream();
32     this.timeout = device.getTimeout();
33
34     return true;
35   }
36
37   public void disconnect(){
38     try {
39       this.client.disconnect();
40     } catch (IOException e) {
41       LOGGER.warning(e.getMessage());
42     }
43   }
44
45   public boolean sendCommand(String command){
46     boolean retVal = true;
47     String sendTmp = command+"\n";
48     try {
49       this.output.write(sendTmp.getBytes());
50       this.output.flush();
51     } catch (Exception e) {
52       LOGGER.warning(e.getMessage());
53       return false;
54     }
55     return retVal;
56   }
57
58   public String getResult(){
59     final StringBuilder retVal = new StringBuilder();
```

```
60      final byte[] buffer = new byte[1024];
61      final InputStream bufIn = this.client.getInputStream();
62      final Thread reader = new Thread() {
63        @Override
64        public void run() {
65          try {
66            bufIn.read(buffer);
67            retVal.append(new String(buffer));
68          } catch (Exception e) {
69            LOGGER.warning(e.getMessage());
70          }
71        }
72      };
73
74      Thread interrupt = new Thread() {
75        @Override
76        public void run() {
77          reader.interrupt();
78        }
79      };
80
81      reader.start();
82      while(true) {
83        try{
84          reader.join(this.timeout);
85          if (!reader.isAlive()) {
86            break;
87          }
88          interrupt.start();
89          interrupt.join(1000);
90          reader.join(1000);
91          if (!reader.isAlive()) {
92            break;
93          }
94
95          LOGGER.warning("Reader thread hung up");
96          System.exit(1);
97        } catch (Exception e){
98                  LOGGER.warning(e.getMessage());
99              }
100         }
101
102         return retVal.toString();
103     }
104 }
```

Abbildungsverzeichnis

Literaturverzeichnis

[ABKMP08] CARY ABRAHAMSON, MICHAEL BLODGETT, ADAM KUNEN, NATHAN MUELLER AND DAVID PARTER: *Splat: A Network Switch/Port Configuration Management Tool*, Oktober 2003.

[APACH01] FOUNDATION, APACHE SOFTWARE: *Example of use of Telnet Client*, http://www.java2s.com/Code/Java/Network-Protocol/ExampleofuseofTelnetClient.htm .

[BSI07] BSI - BUNDESAMT FÜR SICHERHEIT IN DER INFORMATIONSTECHNIK: *Geeignete Auswahl eines Netzmanagement-Protokolls*, 2007.

[CISCO02] CISCO: *Cisco Network Assistant Data Sheet*, 2006.

[CL05] BRUCE CAMPBELL, ROBYN LANDERS: *Open Network Administrator (ONA) - A Web-Based Network Management Tool*, 2005.

[ENCK07] WILLIAM ENCK, PATRICK MCDANIEL, ALBERT GREENBERG, SANJAY RAO, PURDUE UNIVERSITY, WILLIAM AIELLO, SUBHABRATA SEN, PANAGIOTIS SEBOS AND SYLKE SPOEREL: *Configuration Management at Massive Scale: System Design and Experience*, 2007.

[HP00] HP: *HP ProCurve Manager RN0810*, September 2010.

[HPL00] PLANSKY, DR. HERBERT: *Automatisierung für Stadtnetzbetreiber.* CKC Telecom IT, Juli 2011.

[KDO01] DOOLEY, KEVIN: *Designing Large Scale LANs.* OReilly Media, November 2001.

[MTOSI00] TMFORUM: *MTOSI Release Note RN306 2.0*, März 2007.

[NETD00] *NETDISCO - Network Management Tool*, http://www.netdisco.org .

[RFC1157] CASE, J.D., M. FEDOR, M.L. SCHOFFSTALL und J. DAVIN: *Simple Network Management Protocol (SNMP).* RFC 1157 (Historic), Mai 1990.

[RFC1350] SOLLINS, K.: *The TFTP Protocol (Revision 2).* RFC 1350 (Standard), Juli 1992. Updated by RFCs 1782, 1783, 1784, 1785, 2347, 2348, 2349.

[RFC1901] CASE, J., K. MCCLOGHRIE, M. ROSE und S. WALDBUSSER: *Introduction to Community-based SNMPv2.* RFC 1901 (Historic), Januar 1996.