HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

Intent based networking with NSO

Bachelorarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Frühjahrssemester 2019

Autoren:	Patrik Peng, Raphael Hämmerli
Betreuer:	Prof. Laurent Metzger
Co-Betreuer:	Urs Baumann
Projektpartner:	Führungsunterstützungsbasis FUB
Experte:	Marcel Witmer

1 Abstract

Ziel dieser Arbeit ist eine Evaluation vom Cisco Network Services Orchestrator (NSO) für den Einsatz bei der Führungsunterstützungsbasis (FUB) der Schweizer Armee. Es wird ermittelt, ob dieses Werkzeug zur Orchestrierung von Netzwerkkomponenten für eine Automatisierung von MPLS Layer 3 VPN Konfiguration verwendet werden kann.

Es wurde eine Erweiterung für das NSO in Form eines sogenannten Services erstellt. Dieser Service beinhaltet die benötigte Logik in Form von Python Code und die Gerätekonfiguration in Form von Templates mit YANG Models. Für die Kommunikation zwischen NSO und den jeweiligen Netzwerkkomponenten werden sogenannte NETCONF Network Element Drivers (NED) verwendet. Des Weiteren wurde für die Bedienung des Services eine externe grafische Oberfläche als Ruby on Rails Applikation entwickelt, welche mittels RESTCONF Schnittstelle mit dem NSO kommuniziert.

Ein Grossteil der Grundanforderungen konnte im Verlauf dieser Arbeit erfolgreich umgesetzt werden. Verschiedene Probleme mit den NETCONF NEDs führten jedoch zu Mehraufwand und Einbussen in der Performance, sodass nicht alle gewünschten Anforderungen des Industriepartners umgesetzt werden konnten. Die für diese Arbeit verwendete Technologiekonfiguration des NSO ist für den produktiven Einsatz daher nur bedingt empfehlenswert und entsprechend fällt das Fazit dieser Evaluation durchzogen aus. Trotz allem ist das NSO in Kombination mit eigenen Services ein mächtiges und praktisches Werkzeug für die Netzwerkorchestrierung und könnte mit besseren NEDs durchaus sein volles Potenzial ausschöpfen.

2 Management Summary

Ausgangslage
Für die Serviceautomatisierung ihres neuen IP Backbones möchte die Schweizer Armee Cisco NSO, eine industrieführende Orchestrierungsplattform für hybride Netzwerke, evaluieren. NSO bietet einem Netzwerkadministrator die Möglichkeit, sämtliche Konfiguration zentral zu managen, anstatt jedes Netzwerkgerät von Hand konfigurieren zu müssen. Die Arbeit wird betreut von Laurent Metzger und Urs Baumann vom Institute for Networked Solutions (INS) der HSR.

Die Armee will mithilfe von NSO mehrere Standorte, mittels eines gemeinsamen Netzwerkes, über voneinander isolierte Netzwerk-Tunnel verbinden. NSO kann durch sogenannte Services mit zusätzlicher Logik erweitert werden. Innerhalb dieser BA wird ein solcher Service erstellt, der die Verbindung von Standorten erleichtern soll. Zum Service wird auch eine grafische Benutzeroberfläche erstellt, welche dessen Bedienung vereinfacht.

Der Service soll mit Routern von Cisco vom Typ XE und XR und Juniper funktionieren. Die dazu benötigten Testgeräte werden vom INS als virtuelle Router zur Verfügung gestellt. Für die Kommunikation mit diesen Geräten wird das herstellerunabhängige NETCONF Protokoll verwendet.

Vorgehen und Technologien
Die Implementierung wurde aufgeteilt in Arbeiten im Frontend und Backend. Patrik Peng kümmerte sich um die Erstellung der Benutzeroberfläche und Raphael Hämmerli um den NSO Service. Trotz dieser Aufteilung wurde der Grossteil der Software gemeinsam entwickelt. Sämtlicher Code wurde immer von beiden Teammitgliedern geprüft und Probleme wurden gemeinsam angegangen.

Da beide Teammitglieder noch nie für Cisco NSO entwickelt haben, wurde zunächst zur Einarbeitung ein einfacher Test Service erstellt und der Vorgang dokumentiert. So konnten auch die benötigten Schnittstellen ausgetestet und potenzielle Technologien evaluiert werden. Der NSO Service wurde mit Python, für die Programmierung der Logik und YANG, für die Datenmodellierung, entwickelt. Die Benutzeroberfläche setzten wir als Website um. Diese wurde mit Ruby on Rails aufgebaut, da wir in der Studienarbeit bereits Erfahrung damit sammeln konnten und das Framework sich gut dafür eignete.

Als nächster Schritt wurde die Architektur der zu erstellenden Software erstellt und als Proof-of-Concept ein Prototyp entwickelt. Auf dieser Basis wurde der Prototyp zum endgültigen Produkt weiterentwickelt. Während der Entwicklung wurde die Architektur den neuen Begebenheiten angepasst und zum Schluss die Dokumentation abgeschlossen.

Ergebnisse
Ein Benutzer ist in der Lage, mittels unserer Website, zwei oder mehr Standorte miteinander zu verbinden. Ein Standort kann beliebig viele solche Tunnels besitzen, ohne das diese miteinander interferieren. Zwecks Ausfallsicherheit, können pro Standort auch zwei Geräte verwendet werden (sog. Dual Attachment). Auf der Website ist zudem einsehbar, ob auf sämtlichen verwendeten Netzwerkgeräten die gewünschte Konfiguration vorhanden ist, oder ob von einer dritten Stelle aus unerwünschte Änderungen vorgenommen wurden.

Die Software wurde aus Zeitgründen nur für Cisco XE Geräte realisiert, dennoch konnte ein Grossteil der obligatorischen Funktionen umgesetzt werden. Die fehlenden Funktionen sind nicht übermässig komplex und könnten ohne grosse Änderungen an der Softwarearchitektur hinzugefügt werden.

Im Laufe dieser Arbeit haben wir die Funktionalität des Cisco NSO evaluiert und konnten, trotz einigen Hindernissen, die Grundanforderungen des Industriepartners umsetzen.

Ausblick

Als nächster, weiterführender Schritt, steht die Umsetzung der nicht realisierten Features an. Dies beinhaltet die Realisierung vom Support für Cisco XR und Juniper Geräte, sowie weitere Konfigurationsmöglichkeiten. Während der Arbeit haben wir mit der NSO-Version 4.7 gearbeitet, mittlerweile ist mit 5.1 eine neue Major-Version erschienen. Diese neue Version bietet neue Features und Möglichkeiten, die bei der Erweiterung der Software um neue Gerätetypen möglicherweise hilfreich sind.

Für das NETCONF Protokoll, welches für die herstellerunabhängige Kommunikation mit den Netzwerkgeräten verwendet wird, existiert eine Alternative von Cisco, die bessere Zuverlässigkeit bieten soll. Falls diese Kommunikationstechnologie hält, was Cisco verspricht, könnte die Performance der Software zusätzlich verbessert werden.

Inhaltsverzeichnis

3 Anforderungen

3.1 Anforderungsanalyse

3.1.1 Einführung

Für eine erfolgreiche Umsetzung des Projekts ist es essentiell, dass die Aufgabenstellung korrekt interpretiert wird. Um dies zu erreichen werden im folgenden Abschnitt die Aufgabenstellung und die zugehörigen Anwendungsfälle genauer analysiert.

3.1.2 Auftrag

Der Auftrag mitsamt den Use Cases wurde uns von der Führungsunterstützungsbasis (FUB) zugestellt. Die Use Cases wurden Wort für Wort übernommen und sind deshalb in English verfasst.

Introduction
For the service automation on its new IP backbone, the swiss army would like to evaluate Cisco NSO, an industry-leading orchestration platform for hybrid networks.

NSO is a model driven (YANG) platform for automating the network orchestration. It supports multi-vendor networks through a rich variety of Network Element Drivers (NEDs). It supports the process of validating, implementing and abstracting the network configuration and network services, providing support for the entire transformation into intent based networking.

Goal
The goal of that thesis is to implement multi-vendor Service automation using NSO and to build a GUI that support the operation.

The configuration tool will be developed in the laboratory of the INS on Cisco and Juniper virtual routers.

3.1.3 Use Cases

UC-1 Layer3 VPN [Priorität 1]
It should be possible to deploy a Layer 3 VPN on a network composed of virtual routers IOS-XE, IOS-XR and JUN-OS.

The L3 VPN configuration should be deployed on the PEs and on the CEs.
On a customized web GUI, the user should be able to fill in in a form the following parameters:

- Site (for example Dübendorf)
- Single/ redundant access

A drop-down list of the available CEs should pop up and the one to configure should be selected.

- VPN name (for example Komsys)
- VPN ID (for example 1010)
- LAN mask (for example /24)
- QOS policy (drop-down list)

The IP addresses have to be taken from Infoblox and the reservation has to be documented in Infoblox (Description, router name, port).

The Routing Protocol between PE-CE must be E-BGP and each L3 service will be placed in a VRF and transported by a sub-interface using a dot1q encapsulation (VPN ID) between PE and CE.

UC-1.1	L3VPN single attachment with IP addresses that are manually assigned (= no IPAM) with a single service.
UC-1.2	L3VPN single attachment with IP addresses that are extracted from the Infoblox IPAM.
UC-1.3	L3VPN single attachment with IP addresses that are extracted from the Infoblox IPAM and with several services (= with subinterfaces)
UC-1.4	L3VPN dual attachment with IP addresses that are extracted from the Infoblox IPAM and with several services (= with subinterfaces)

UC-2 Service Übersicht [Priorität 1]
On the web GUI, there should be an option to display a list of all the sites where a specific VPN has been deployed. The VPN name can be chosen from a drop-down list.

By clicking check box(es), I should be able to modify the parameters or decommission the VPN service on one or more sites concurrently.

UC-3 Out-Of Sync Verhalten [Priorität 2]
It should be possible to get a graphical view of the out-of-sync configuration in the network with the option to sync from the NSO DB or to sync from the network.

UC-4 Flexalgo Unterstüzung [Priorität 2]
The Deployment of the Layer 3 VPN Service should support a triple control-plane backbone. (Flexalgo)

On a customized web GUI, the user should be able to specify which backbone, Prefix-SID and ALGO Number is going to be used for a specific service.

For example, backbone RED, Prefix-SID 16809 (ALGO 128) and Prefix-SID 16909 (ALGO 129) for backup path of backbone GREEN.

UC-5 Layer2 Pseudowire [Priorität 1, Optional]
It should be possible to deploy a Layer 2 Pseudowire from CE to CE. The web GUI should ask in a form for the following parameters:

- Site A and Site B

A drop-down list of the available CEs should pop up.

The IP addresses has to be taken from Infoblox and the reservation has to be documented (Description, router name, interface type and number)

UC-6 Health Check [Priorität 2, Optional]
Health check of the VPN status on a dashboard of the GUI. IP SLA will be used to guarantee the IP connectivity within a VRF.

Thanks to IP SLA ICMP echo probes, end-to-End response time between sites can be measured and be displayed in that dashboard.

UC-7 Management von CE & PE [Priorität 2]
The GUI should support the addition of new PEs and new CEs into NSO.

Use Case Diagramm

In Abbildung 1 ist das Use Case Diagramm zu sehen. Dieses stellt die Use Cases in Relation zueinander und zu den externen Systemen dar.

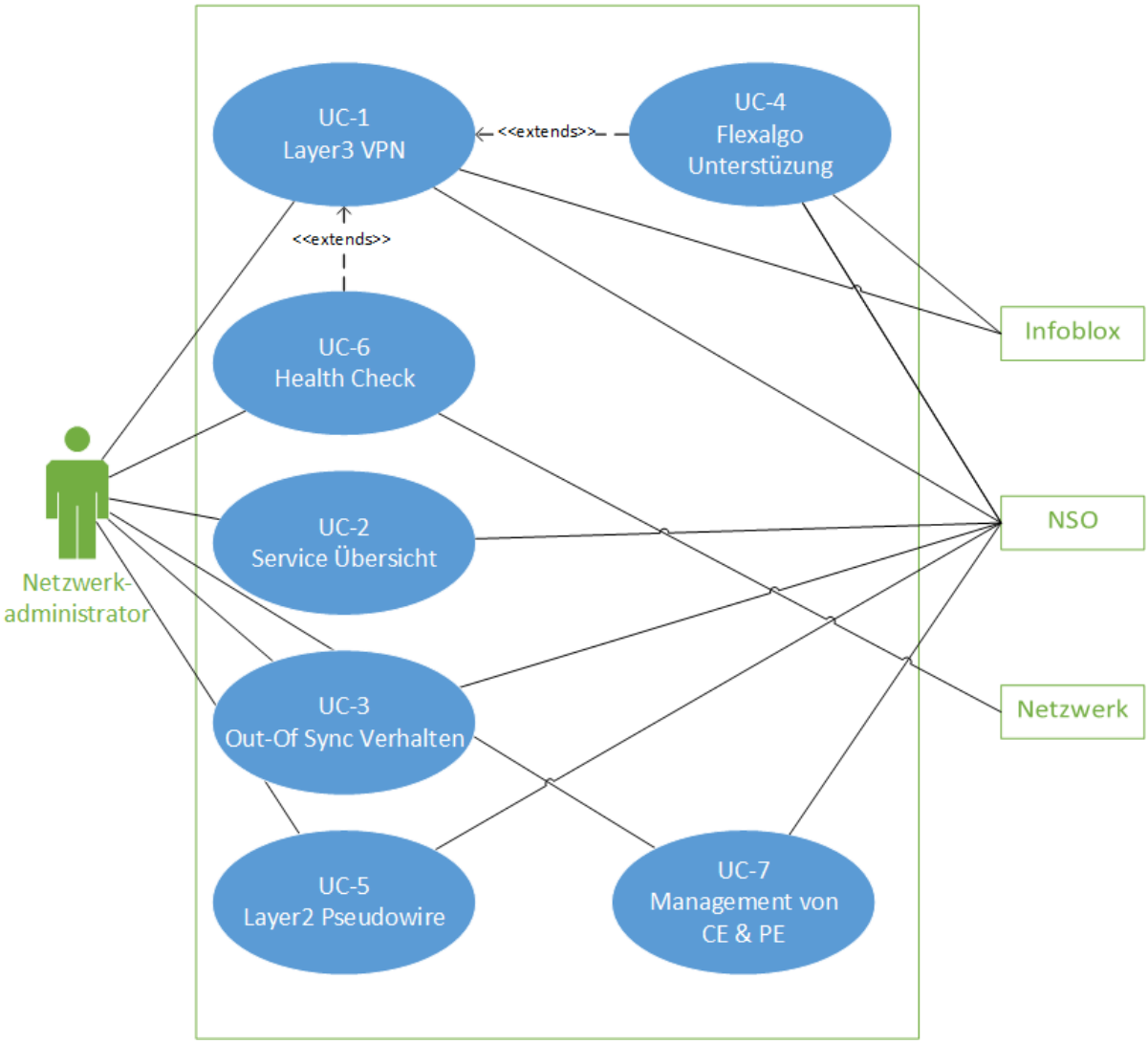

Abbildung 1: Use Case Diagramm

3.1.4 Abgrenzung

Die Anwendung wird mit Network Services Orchestrator (NSO) interagieren um MPLS-VPNs zu konfigurieren. Sie wird nicht für andere Konfigurationen auf dem NSO zuständig sein. Sämtliche Konfigurationsänderungen werden nur über das NSO erfolgen und nicht direkt auf den Geräten. Die Anwendung ist auf die Topologie der FUB zugeschnitten und soll nicht universell anwendbar sein. Es werden nur Geräte mit IOS-XE, IOS-XR und JUN-OS unterstützt.

3.1.5 Bestehende Lösungen

Bei der Erarbeitung der Anforderungen stellte sich die Frage, ob nicht bereits eine kommerzielle Lösung existiert, welche den Anforderungen entspricht. Dabei wurden bestehende Softwarelösungen kurz recherchiert und analysiert.

HP Intelligent Management Center

Das HP Intelligent Management Center (IMC) ist eine Sammlung von Tools, welche die Verwaltung von HP Hardware ermöglicht. Es gibt eine Erweiterung für das IMC, den MPLS-VPN Manager. Dieser ermöglicht die Erstellung und Verwaltung von VPNs über HP Netzwerkgeräte. Der offensichtliche Nachteil hierbei ist die Beschränkung auf HP Hardware.

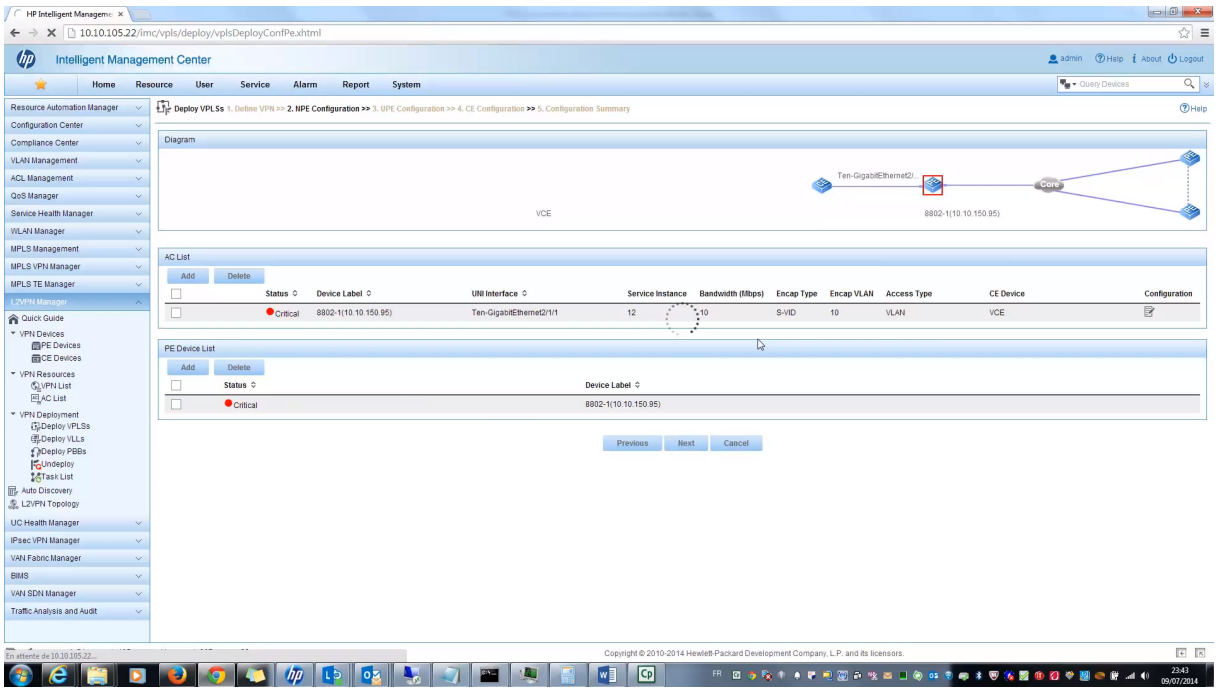

Abbildung 2: HP IMC [Cha15]

CA Spectrum Enterprise Manager

Der CA Spectrum Enterprise Manager ist ebenfalls eine Netzwerkmanagement Software. Hergestellt wird er von der Firma CA Technologies und unterstützt auch mehrere Netzwerkgerätehersteller, unter anderem auch Cisco und Juniper. Der CA Spectrum Enterprise Manager erlaubt jedoch nur die Überwachung von bestehenden Multiprotocol Label Switching (MPLS)-VPNs und bietet keine Möglichkeiten, um neue VPNs hinzuzufügen.

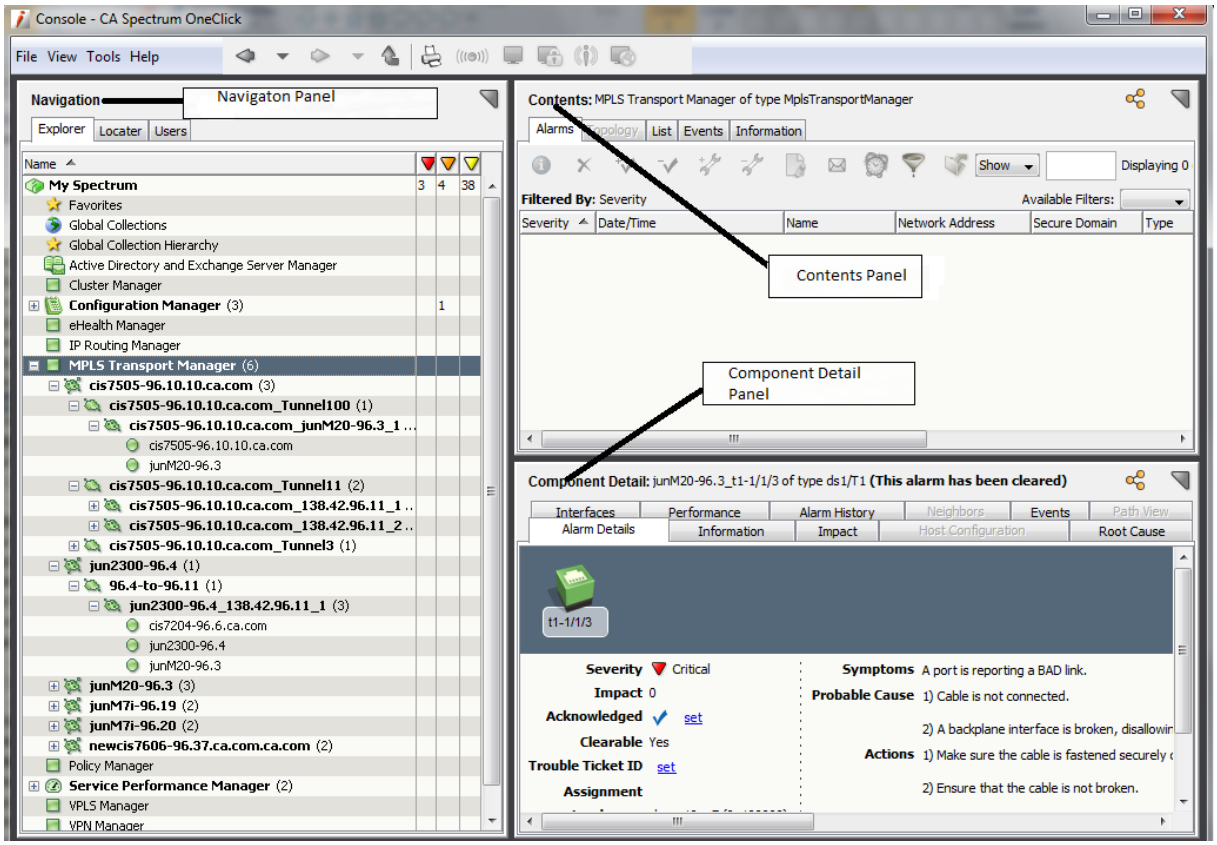

Abbildung 3: CA Spectrum Enterprise Manager [Tec14]

3.2 Anforderungsspezifikation

3.2.1 Einführung

Nach der Analyse der Anforderungen, werden nun sämtliche benötigten Komponenten spezifiziert und Möglichkeiten für die Implementation vorgelegt.

3.2.2 Nicht funktionale Anforderungen

Anhand von nicht funktionalen Anforderungen soll festgelegt werden, welche Qualitätsattribute das Endprodukt in Bezug auf Sicherheit, Skalierbarkeit, Bedienbarkeit und Performance erfüllen muss.

Sicherheit
Unbefugten Drittpersonen darf es nicht ermöglicht werden, Konfigurationen am System vorzunehmen, oder Informationen mitzulesen. Um dies zu gewährleisten werden jegliche Kommunikationskanäle zwischen Benutzer und dem System, sowie den einzelnen Netzwerkkomponenten verschlüsselt und das System wird nur für autorisierte Personen erreichbar sein.

Skalierbarkeit
Da in der Realität ein Netzwerk schnell aus vielen Geräten bestehen kann, muss die Applikation in der Lage sein diese Anzahl zu bewältigen. Die Softwarelösung soll mit 20 bis 30 Netzwerkkomponenten innerhalb der Performancespezifikationen liegen.

Bedienbarkeit
Die Bedienoberfläche, welche dem Benutzer angezeigt wird, soll intuitiv und so weit wie möglich selbsterklärend sein. Falls nötig werden dem Benutzer Hinweise zur korrekten Bedienung eingeblendet und längere Ladezeiten ohne Informationen so weit wie möglich vermieden.

Performance und Stabilität
Das Endprodukt muss komfortabel und effizient benutzbar sein. Dies schliesst ein, dass die Applikation dementsprechend kurze Reaktionszeiten und Ausfallsicherheit besitzt. Die Software muss in einem Zeitfenster von drei Sekunden auf Benutzereingaben reagieren. Kann diese Vorgabe bei bestimmten Aktionen nicht eingehalten werden, muss dies dem Benutzer entsprechend vermittelt werden. Durch Validierung von Benutzereingaben und korrektem Error-Handling werden unerwartete Programmabstürze so weit wie möglich vermieden.

3.2.3 NSO Service

Um die gewünschten Konfigurationen auf die entsprechenden Netzwerkkomponenten anzuwenden, wird ein NSO Service verwendet. Dieser Service wird die Informationen der Topologie entgegennehmen und daraus die benötigten Gerätekonfigurationen erstellen. Er wird die Hauptlogik zum Verwalten der VPNs enthalten. Ein Service wird als NSO Package erstellt, dazu existieren drei verschiedene Möglichkeiten.

YANG only Service
Ein YANG only Service besteht nur aus einem YANG Model, welches die benötigten Informationen definiert und dem Konfigurationstemplate der Geräte. Der Benutzer kann die Informationen, die im YANG Model vorgegeben sind, ins NSO eingeben und diese werden anschliessend in das Template abgefüllt. Das Abfüllen kann durch simple Logik (unter anderem if, else und foreach) beeinflusst werden, dies ist

aber stark beschränkt. Ein YANG only Service ist für diese Arbeit nicht geeignet, da das Verwalten von einem MPLS VPN mehr erfordert als nur einfaches Abfüllen von Konfiguration.

Python Service
Ein Python Service beinhaltet ebenfalls ein YANG Model und ein Template. Er bietet allerdings zusätzlich die Möglichkeit, vor dem Abfüllen der Konfiguration in das Template, Python Code auszuführen und damit die Variablen aus dem YANG Model zu bearbeiten. Dadurch werden auch sehr komplizierte Verarbeitungen möglich.

Java Service
Ein Java Service funktioniert identisch wie der Python Service, nur dass anstatt Python Java verwendet wird.

3.2.4 GUI Komponente

Damit die Bedienung des NSO Services erleichtert werden kann, wird zusätzlich eine entsprechende grafische Oberfläche erstellt. Diese stellt die Eingabemaske der Konfigurationsdaten für den Service zur Verfügung. Um Eingabefehler zu vermeiden, werden nur gültige Auswahlmöglichkeiten angezeigt, oder diese entsprechend validiert. Das grafische Interface bietet zudem die Möglichkeit, den Synchronisierungstatus der Netzwerkkomponenten zu überprüfen und im Fehlerfall zu korrigieren, indem entweder der Stand aus dem NSO auf die Geräte geschrieben wird, oder dieser neu in das NSO geladen werden. Die Konfiguration und Einsicht von Flexalgo Unterstützung, Layer2 Pseudowire und Health Checks wird ebenfalls über das GUI stattfinden.

Die Erstellung einer solchen Oberfläche kann in zwei verschiedenen Varianten erfolgen. Im NSO kann mittels eines Packages eine eigene Benutzeroberfläche integriert werden, welche direkt im NSO verwendet werden kann. Als Alternative bietet sich die Erstellung einer eigenständigen Lösung, welche mit dem NSO mittels einer Programmierschnittstelle kommuniziert.

NSO integriert
Wenn das GUI direkt im NSO integriert wird, bietet das den Vorteil, dass die gesamte Konfiguration an einem zentralen Ort gesammelt ist und keine eigenständige Softwarekomponente entwickelt werden muss. Jedoch kann die Integration im NSO die Entwicklung, durch technische Hindernisse, erschweren.

Eigenständige Lösung
Eine eigenständige Lösung bietet den Vorteil, dass die Technologien für dessen Umsetzung relativ frei gewählt werden können und vermindert das Risiko von unbekannten Restriktionen bei der Entwicklung. Für den Austausch von Informationen und die effektive Verwendung des NSO Service stehen der externen Oberfläche mehrere Schnittstellen vom NSO zur Verfügung.

3.2.5 Infoblox

Als IPAM ist Infoblox schon fix vorgegeben. Es wird benötigt, um freie IP-Adressen für die VPNs zu beziehen und diese dann zu dokumentieren. Das Infoblox ermöglicht die Kommunikation über eine REST-Schnittstelle, durch die die benötigten Daten abgerufen, oder gespeichert werden können. Wenn ein IP Adressbereich konfiguriert wird, werden diese Änderungen anschliessend mittels dieser Schnittstelle auch im IPAM System dokumentiert.

3.2.6 Persistenter Datenspeicher

Es wird für einige Anwendungsfälle nötig sein, gewisse Informationen persistent zu speichern. Ein Beispiel ist eine Zuordnung von CE-Routern zu Standortnamen.

Datenspeicherung in der CDB
Falls die Daten direkt in der Configuration Database (CDB) des NSO gespeichert werden könnten, würde dies die Handhabung der Daten extrem vereinfachen, da sie alle an einem Ort währen. Allerdings ist es noch nicht klar, ob die CDB diese Möglichkeit bietet. Zudem bietet diese Variante ebenfalls einen Vorteil im Punkt Sicherheit, da einerseits alle Daten an einem Ort sind und andererseits die CDB nicht per Netzwerk erreichbar sein muss und somit isoliert werden kann.

Externe Datenspeicherung
Die Daten können auch in einer externen Datenbank gespeichert werden. Dies führt jedoch dazu, dass neben dem NSO und dem Infoblox noch eine dritte Datenquelle verwaltet werden muss. Wenn die grafische Oberfläche als externe Komponente realisiert wird, könnte hierbei die Datenbank angebunden werden.

3.3 Technologieentscheidungen

Für die zahlreichen Aufgaben, die im Rahmen dieser Bachelorarbeit (BA) gelöst werden müssen, stehen verschiedene Lösungsansätze mit unterschiedlichen Technologien zur Verfügung. Nachfolgend werden die verwendeten Technologien pro Komponente und deren Kommunikation mit anderen Komponenten aufgelistet und deren Technologiewahl begründet. Abbildung 4 zeigt eine Übersicht dieser Komponenten.

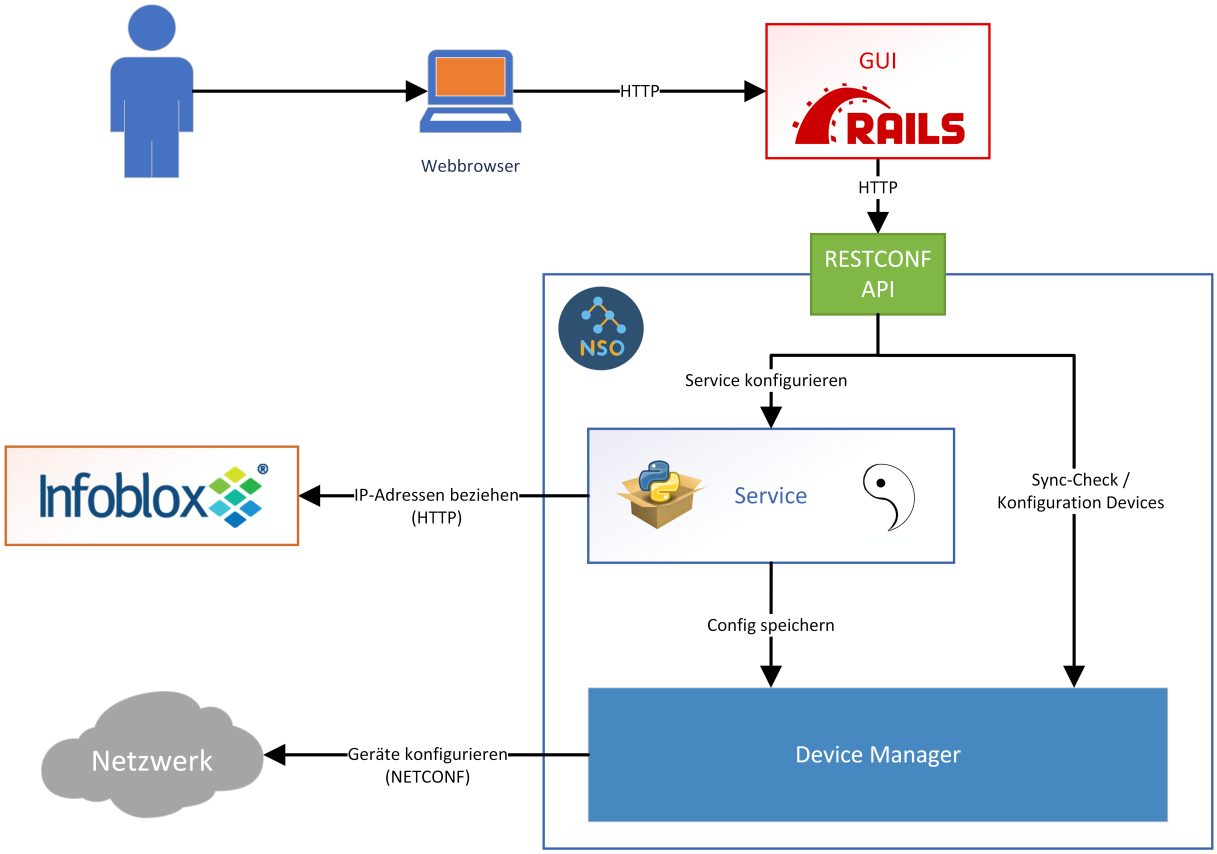

Abbildung 4: Komponentenübersicht

3.3.1 Web-GUI

Der Webserver wird mithilfe von Ruby on Rails entwickelt. Ruby on Rails (kurz Rails) ist ein serverseitiges MVC Web-Framework, welches in der Programmiersprache Ruby implementiert wurde. Grundsätzlich kann Rails auf verschiedenen Plattformen (Windows, Linux, MacOS) verwendet werden. Es bestehen jedoch Einschränkungen unter Windows. Daher wird sich der Fokus dieser BA auf die Unterstützung von Unix basierten Systemen (Linux, MacOS) beschränken. Wir haben Rails bereits für unsere Studienarbeit (SA) verwendet und kennen uns bereits damit aus. Zudem bietet Rails die Möglichkeit, bei Bedarf einfach eine Datenbank zu bedienen, falls dies durch Änderungen in der Architektur zu einem späteren Zeitpunkt nötig wird.

Kommunikation mit dem Web-GUI
Das Web-GUI ist per Webbrowser mittels HTTPS-Verbindung erreichbar. Grundsätzlich ist der Benutzer frei in seiner Browserwahl, für diese BA wird die Oberfläche jedoch nur für Firefox und Chrome getestet.

3.3.2 NSO Service

Für die Entwicklung des Services im NSO stehen Java und Python zur Verfügung. Unsere Wahl ist auf Python gefallen, da diese Sprache für Netzwerkautomation bereits etabliert ist und wir persönliche Vorlieben mit Python haben. Zudem muss Python nicht im voraus kompiliert werden, was den Entwicklungsprozess vereinfacht.

Kommunikation zwischen Rails Applikation und NSO

Das NSO stellt für die Kommunikation mit dem Web-GUI drei verschiedene Schnittstellen zur Verfügung:

REST Proprietäre HTTP-REST Schnittstelle entwickelt von Cisco.

RESTCONF Nach RFC 8040 standardisierte HTTP-REST Schnittstelle für die Übertragung von NETCONF Daten in YANG definierten Formaten.

JSON-RPC Remote Procedure Call Schnittstelle mit JSON Encoding.

Vorerst fiel die Wahl auf die REST Schnittstelle, da diese auf den ersten Eindruck einen vernünftigen Eindruck machte und unsere Anforderungen erfüllte. Das Format für die Anfragen und die Antworten sind für die REST und die RESTCONF Schnittstelle sehr ähnlich, jedoch lieferte die RESTCONF Schnittstelle bei unseren Tests zum Teil massiv lange Antwortzeiten (30 Sekunden oder mehr). Durch entsprechende Query Parameter konnte die Reaktionszeit der RESTCONF Schnittstelle jedoch stark verbessert werden, sodass diese auch innerhalb von 100 bis 1000 Millisekunden antwortete.

Schlussendlich fiel die Wahl dann auf die RESTCONF Schnittstelle, da diese von Cisco für neue Services empfohlen wird und offiziell standardisiert ist. Da mittels der RESTCONF Schnittstelle die Anforderungen gut abgedeckt werden können und bei dessen Verwendung bereits Vorkenntnisse bestehen, wurde die JSON-RPC Schnittstelle nicht weiter evaluiert.

3.3.3 Persistenter Speicher

Als persistenter Speicher wird die CDB des NSO verwendet. Durch den Service wird die Datenstruktur der CDB erweitert, sodass die zusätzlichen Informationen darin gespeichert werden können. In der Web-GUI Applikation hätte ebenfalls ein persistenter Speicher hinzugefügt werden können. Dies wollten wir jedoch explizit vermeiden, sodass die Rails App stateless bleibt und keine Synchronisierungsprobleme zwischen NSO und Rails App auftreten können. Des Weiteren hätte eine weitere Datenbank dem Konzept der Single Source of Truth des NSO widersprochen. Darum werden sämtliche Daten zentral an einem Ort in der CDB des NSO gespeichert.

4 Technologien

Im Rahmen dieser Bachelorarbeit werden verschieden Technologien verwendet. Der folgende Abschnitt befasst sich mit der Analyse dieser Technologien und soll einen kurzen Überblick über deren Funktionsweise bieten.

4.1 NSO

Im Hauptfokus dieser Bachelorarbeit liegt das NSO der Firma Cisco. NSO wurde unter dem Namen Network Control Systen (NCS) von der Firma Tail-f entwickelt. 2014 wurde Tail-f von Cisco aufgekauft und das NCS wurde in NSO umbenannt.

Vor der Übernahme war Tail-f Vorreiter in der standardisierten Konfiguration von Netzwerkgeräten. Dabei arbeiteten sie überwiegend an NETCONF und YANG. Auch das NSO arbeitet mit diesen Technologien. [14b; McG14]

4.1.1 YANG & NETCONF

YANG (Yet Another Next Generation) ist eine modulare Sprache für Datenmodellierung und wurde im Oktober 2010 von der IETF als RFC 6020 veröffentlicht. Die Sprache kann verwendet werden, um Konfigurations und Zustandsdaten zu modellieren, welche anschliessend mittels dem Network Configuration Protocol (NETCONF) übertragen werden. YANG selbst ist protokollunabhängig und beschreibt lediglich Datenmodelle. Für die effektive Instanz eines solchen Modells kann XML oder JSON verwendet werden. Nebst den von der Sprache vordefinierten Datentypen können zusätzliche Typen definiert werden, welche aus den Bestehenden abgeleitet werden.

NETCONF ist ein Protokoll, welches für die entfernte Konfiguration von Netzwerkkomponenten verwendet werden kann und auf dem Remote Procedure Call (RPC) aufbaut. Im Unterschied zu herkömmlichen Methoden mittels CLI, kann NETCONF mithilfe von YANG Modellen geräteunabhängig funktionieren. Sämtliche Konfigurationsänderungen über NETCONF werden in Transaktionen gebündelt. Bei fehlerhaften Eingaben wird die ganze Transaktion rückgängig gemacht. So kann ein konsistenter Stand der Konfiguration garantiert werden.

4.1.2 Network Element Drivers (NED)

Damit NSO mit Geräten von mehreren Herstellern kommunizieren kann, muss bekannt sein, wie es diese Geräte anzusprechen hat. Dafür werden die Network Element Driver (NED)s verwendet. Sie übernehmen die Kommunikation mit den Netzwerkkomponenten und können mehrere Protokolle unterstützen. Unter anderem NETCONF, REST, CLI und SNMP. Falls ein Gerätehersteller eine NETCONF Schnittstelle anbietet, kann der dazugehörige NED sogar automatisch generiert werden. Einfache CLI NEDs für Cisco, Dell und Juniper Geräte sind bereits in der Grundinstallation von NSO integriert. Diese sind jedoch veraltet und dienen nur zu Testzwecken. Weitere CLI NEDs können von Cisco erworben werden.[14a]

NETCONF NEDs

Die Idee von NETCONF NEDs ist simpel. Ein Gerät mit NETCONF Unterstützung gibt seine Konfigurationsmöglichkeiten über YANG Models an. In diesen Models wird beschrieben, welche Datenstruktur das Gerät zur Konfiguration akzeptiert. Die Datenstruktur der CDB wird um die Models erweitert. So kann ein Benutzer diese Daten konfigurieren und NSO übergibt diese eins zu eins per NETCONF dem Gerät.

Durch die Transaktionalität von NETCONF können auch bei fehlerhaften Commits ungültige Konfigurationsreste verhindert werden.

CLI NEDs
Im Gegensatz zu NETCONF NEDs verwenden CLI NEDs das normale Terminal der Geräte, welches meist per SSH angesteuert wird. Diese NEDs können nicht autogeneriert werden. Sie erweitern ebenfalls die CDB, jedoch wird die Datenstruktur nicht vom Netzwerkgerät vorgegeben, sondern von den Entwicklern des NEDs. Da ein solcher NED per normalem Terminal mit den Geräten kommuniziert, ist die Transaktionalität vom Gerät selber nicht gewährleistet. Der CLI NED muss selber bei einer fehlgeschlagenen Transaktion alle unerwünschten Änderungen wieder rückgängig machen.

4.1.3 Funktionalität

NSO ist in der Lage die Konfiguration von mehreren Netzwerkgeräten zu koordinieren und orchestrieren. Dabei fungiert es als „Single Source Of Truth“. Dies bedeutet, dass Konfigurationsänderungen nicht mehr direkt auf den Geräten vorgenommen werden müssen, sondern zentral im NSO verwaltet werden können. Falls nun die Konfiguration eines Geräts aus unbekannten Gründen verändert wurde, kann NSO einfach identifizieren, was geändert wurde und so langwieriges Troubleshooting vermieden werden.[Cis18c]

Durch die zentrale Verwaltung von den Netzwerkgeräten bietet NSO auch zusätzliche Methoden zur Konfiguration. Geräte müssen so nicht mehr einzeln bearbeitet werden, sondern können in Gruppen zusammengefasst werden. [16a]

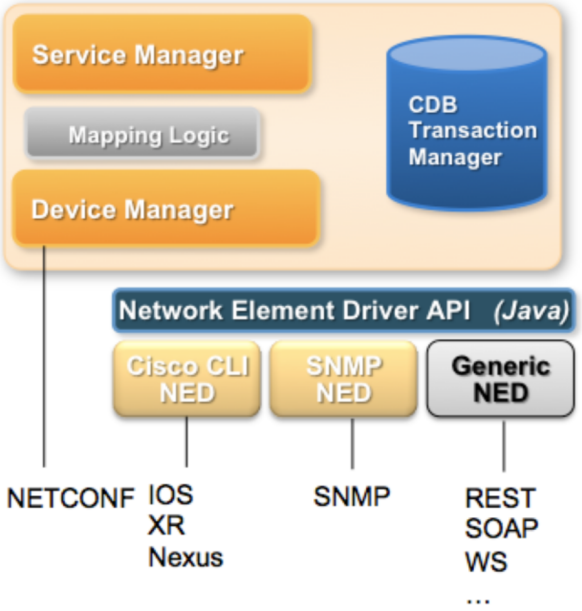

Abbildung 5: NSO Komponenten [Sri18]

NSO Services
Eins der stärksten Features von NSO ist die Organisation der Konfiguration in Services. Anstatt jedes Gerät individuell einzurichten, werden Services definiert, welche auf mehreren Geräten operieren. Ein einfaches Beispiel ist die Konfiguration eines VLAN verteilt über mehrere Switches. Die VLAN Konfiguration wird in den Service integriert und alle Geräte die betroffen sind, werden dem Service hinzugefügt. Nun können die notwendigen Änderungen pro Gerät identifiziert und angewendet werden. Damit dem NSO bekannt, ist welche Daten und in welchem Format diese für einen Service benötigt werden, muss für den Service auch ein entsprechendes YANG Modell erstellt werden.

Transactions
NSO verarbeitet Konfigurationsänderungen auf Transaktionsbasis. Änderungen, welche voneinander abhängig sind können in einer Transaktion gruppiert werden. NSO stellt sicher, dass diese Transaktion vollständig auf die Geräte geschrieben wird. Falls eine Konfiguration in einer Transaktion von einem Gerät abgelehnt wird, macht NSO die ganze Transaktion rückgängig. So wird sichergestellt, dass fehlerhafte Konfigurationen keine Artefakte auf den Geräten zurücklassen. NSO legt für jede erfolgreiche

Transaktion einen Rollback Checkpoint an. Dadurch ist es möglich, nachträglich Änderungen einfach rückgängig zu machen.

Integrierter Datastore
Für die Speicherung der Konfiguration von jeweiligen Geräten, verwendet Cisco NSO intern die sog. CDB. Diese Datenbank verwendet eine Baumstruktur mit YANG Schema. So können die gespeicherten Informationen mittels der YANG Models der entsprechenden Geräten jederzeit validiert werden. Jegliche Änderungen an Komponenten, welche NSO anwendet, werden in dieser Datenbank nach erfolgreicher Transaktion gespeichert.

NSO kann überprüfen, ob die Konfigurationen der Geräte weiterhin mit dem Stand, welcher NSO kennt, synchronisiert (in-sync) sind. Für diesen Sync-Check verwendet NSO zwei Methoden. Die erste verwendet einen einfachen Commit-Hash für den Vergleich. Dies ist zwar sehr schnell, allerdings muss das verwendete Gerät solche Hashes unterstützten. Die zweite Methode liest die komplette Gerätekonfiguration aus und prüft sie gegen die CDB, was mit allen Geräten funktioniert. Im Falle eines Konflikts kann entweder die Konfiguration des Geräts neu in die CDB geschrieben werden, oder der Stand aus Sicht vom NSO auf das Gerät geladen werden. Dieser Datastore bildet somit die Grundlage für die bereits erwähnte Single Source of Truth.

Device-Manager
Der Device-Manager bildet mit dem Service-Manager das Herzstück des NSO. Er kümmert sich um die Transaktionen und Rollbacks von Konfiguration. Zusätzlich kann er die aktuelle Konfiguration auf den Geräten mit den Daten in der CDB vergleichen und so feststellen, ob ein Gerät ausserhalb von NSO geändert wurde.

4.1.4 Services im NSO

Der Zweck eines NSO Services ist es, aus einfachen Vorgaben des Benutzers, Konfiguration für ein oder mehrere Geräte zu generieren. Das Erstellen einer solchen Konfiguration kann zwar herausfordernd sein, ist jedoch noch vergleichsweise simpel. Sobald es darum geht, bestehende Service Konfiguration anzupassen, oder zu löschen, wird es komplizierter. Um den Entwicklern von Services diese Arbeit zu ersparen, besitzt NSO den FASTMAP Algorithmus. Durch die Verwendung von FASTMAP muss ein Service-Entwickler sich nur um die Erstellung eines Services kümmern.

Service Funktionsweise
Ein Service generiert seine Konfiguration in mehreren Schritten. Abbildung 6 zeigt eine Gesamtübersicht für einen Service mit Python Komponente. Die Benutzerdaten kann der Service über mehrere Wege erhalten, unter anderem über externe Schnittstellen wie RESTCONF oder JSON-RPC, über die CLI oder über das eigene GUI. Diese Daten werden zunächst gegen das Service YANG Model validiert. Zu dieser Validierung gehören Datentypen, Value-Ranges und zusätzliche YANG Constraints, wie zum Beispiel die Zugehörigkeit eines Gerätes zu einer Gruppe.

Ist die Validierung erfolgreich, wird die Python Komponente angestossen. In ihr können die Daten bearbeitet und mit zusätzlichen Daten aus der CDB oder aus externen Quellen (hier repräsentiert durch Infoblox) angereichert werden. Wichtig ist, dass die Python Komponente deterministisch sein muss. Das heisst, gleiche Inputdaten müssen immer identischen Output produzieren. Für Daten, die aus externen Quellen bezogen werden, muss sichergestellt werden, dass jedes Mal die gleichen Werte geliefert werden und keine Seiteneffekte auftreten. Ein Beispiel für solche Nebeneffekte ist das Reservieren von

IP-Adressen aus dem Infoblox. Falls Nebeneffekte dennoch gewünscht sind, können diese mit dem Reactive FASTMAP Pattern, beschrieben im Abschnitt 5.4.1, bearbeitet werden.

Die Python Komponente füllt ihre Daten schlussendlich in ein XML-Template ab. Aus diesem wird die Konfiguration generiert, welche in der CDB gespeichert und gegebenenfalls an die Geräte gesendet wird.

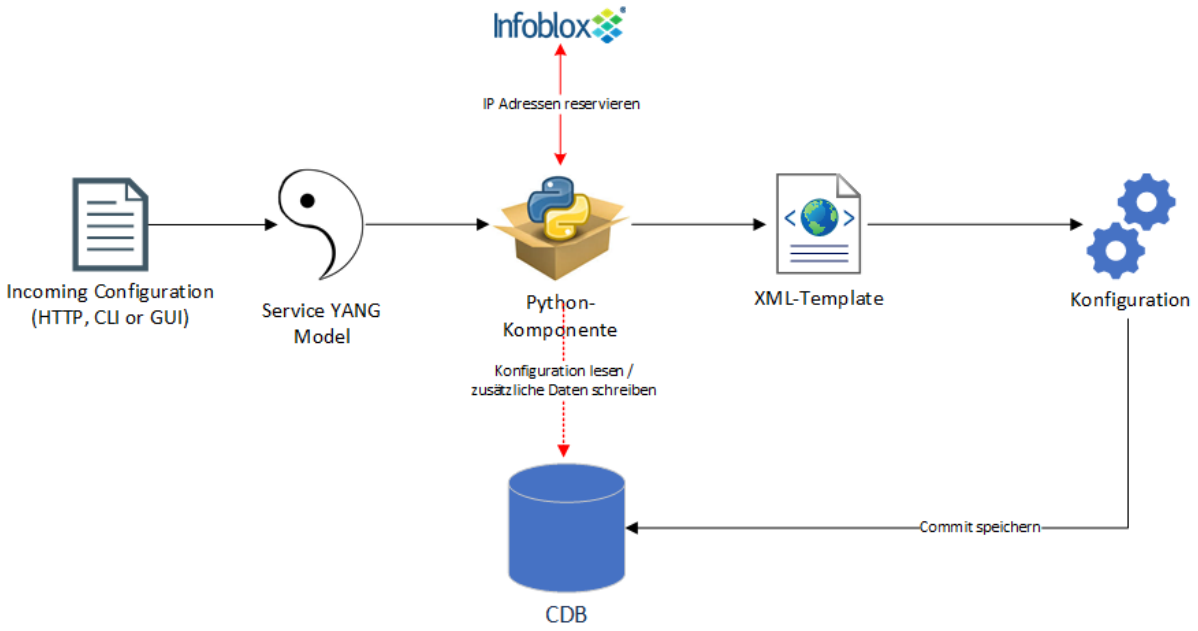

Abbildung 6: Übersicht über die Service Pipeline

FASTMAP Funktionsweise

Wenn ein NSO Service neu instanziiert wird, generiert er seine Konfiguration und FASTMAP schreibt diese in die CDB. Falls eine Instanz geändert wird, übergibt FASTMAP der Python Komponente die geänderten Parameter und lässt sie die Config neu erstellen. Dabei befindet sich die Komponente in einem speziellem Kontext, wo sämtliche vorhergehende Änderungen dieser Instanz ausgeblendet sind. So sieht es für die Python Komponente aus, als ob die Instanz zum ersten Mal erstellt wird. FASTMAP vergleicht anschliessend die neue und alte Konfiguration und schreibt die Änderungen in die CDB. Beim Löschen wird die Python Komponente nicht gestartet. Während dem Entfernen von spezifischer Konfiguration (wie eine IP-Adresse) kann FASTMAP feststellen, ob diese noch anderweitig verwendet wird. In diesem Fall wird sie nicht gelöscht. Durch die Verwendung von FASTMAP muss sich ein Service Entwickler nur um das Erstellen einer Serviceinstanz kümmern. Auf Abbildung 5 wird FASTMAP als Mapping Logic dargestellt. [Cis18b]

Konflikte zwischen Services

Wenn mehrere NSO Services installiert sind, kann es zu Konflikten kommen. Um solche Konflikte aufzuspüren und von Hand zu lösen, bietet der Service-Manager die Möglichkeit ein Sync-Check für eine spezifische Service Instanz durchzuführen. Dabei wird die Konfiguration neu erstellt und mit der Aktuellen verglichen. Bei Bedarf kann ein Service auch auf die Netzwerkgeräte re-deployed werden.

4.1.5 Integration mit einem IPAM

Abbildung 7: Infoblox Logo [Inf]

Jedes Gerät in einem Netzwerk, welches erreichbar sein soll, verfügt bekanntlich über eine im Netz einzigartige IP Adresse, welche Teil eines Subnetzes ist. In einem Netzwerk mit vielen Geräten und Subnetzen kann der Überblick jedoch schnell verloren gehen. Um dem entgegenzuwirken, werden daher IP-Adressen und zugehörige Informationen häufig in sogenannten IP Adress Management (IPAM) Systemen verwaltet. Somit ist an einem zentralen Punkt ersichtlich, wo welche Adressen mit welchen Netzwerkkomponenten verwendet werden. Die Schweizer Armee verwendet ein solches System der Firma Infoblox.

Um das Management der Standorte mittels NSO so einfach wie möglich zu gestalten, sollten die dazugehörigen Konfigurationsschritte möglichst automatisiert werden. Damit dies ermöglicht werden kann, müssen NSO und allfällige Zusatzkomponenten in der Lage sein, mit diesem IPAM System über eine Schnittstelle zu kommunizieren. So können automatisch neue IP-Subnetze im IPAM System beantragt, oder wieder freigegeben werden.

Infoblox bietet für das IPAM System eine REST Programmierschnittstelle an, welche von externen Komponenten verwendet werden kann. [16b] Somit sollte einer Integration von Infoblox und Cisco NSO nichts im Weg stehen.

4.2 Multiprotocol Label Switching (MPLS)

Local tag	Outgoing tag or VC	Prefix or Tunnel Id
16	Pop tag	10.0.5.0/24
17	Pop tag	10.0.2.0/24
18	18	10.0.6.0/24
19	16	10.0.3.0/24
20	17	10.0.4.0/24
21	19	10.0.7.0/24
22	20	10.0.20.1/32
23	Aggregate	10.0.0.0/24[V]
24	Aggregate	10.0.10.0/24[V]

Abbildung 8: Beispiel einer MPLS-LFIB

MPLS ist eine Technologie zum Routen von Datenpaketen in einem Netzwerk. Es entstand in den 1990er Jahren als Alternative zum ressourcenintensiven IP-Routing. Im Gegensatz zu konventionellem Routing, welches bei jedem Paket aufs neue per Longest Prefix Match (LPM) entscheidet, welchen Pfad ein Paket durch das Netzwerk nimmt, entscheidet MPLS schon beim ersten Router des Netzwerks, welchen Pfad ein Paket nehmen wird. Dieser Router sucht per LPM nach einem Pfad. Hat er einen gefunden, fügt er einen MPLS-Header mit einem Label vor das IP-Paket. Die nachfolgenden MPLS-Router im Netzwerk müssen nur noch anhand des Labels überprüfen, welcher Pfad gewählt werden muss. So können aufwendige LPMs bei jedem Router vermieden werden.

4.2.1 VRF

MPLS VPNs verwendet Virtual Routing and Forwarding (VRF) um die einzelnen VPN Instanzen zu separieren. Ein VRF ist ein virtuelle Instanz eines Routers. Es besitzt eine eigene Routingtabelle und einen eigenen Routing-Prozess. So kann es völlig unabhängig von anderen VRFs operieren. Dadurch kann ein Router auch mehrere Netzwerke mit dem gleichen Subnetz verwalten, solange sie sich in unterschiedlichen VRFs befinden.

4.2.2 MPLS VPN

Auch wenn in der heutigen Zeit die Performance von Routern stark gestiegen ist und die geringere Rechenzeit gegenüber IP nicht mehr ganz so relevant ist, bietet MPLS auch noch ein weiteres, mächtiges Feature, den MPLS Layer 3 VPN.

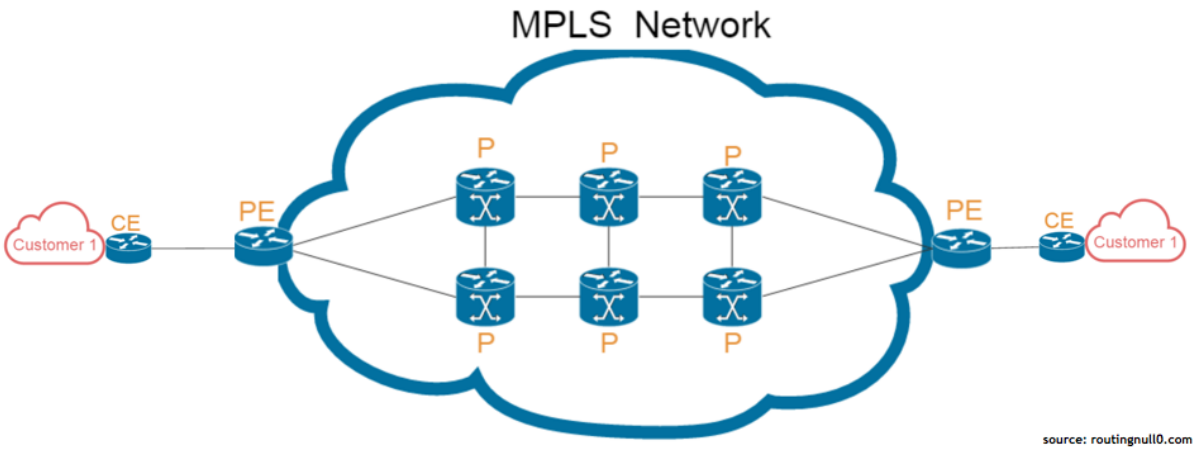

Abbildung 9: Übersicht MPLS VPN Netzwerk[Jac17]

Router Rollen

P (Provider): P-Router befinden sich im Zentrum des MPLS Netzwerkes. Das Layer Distribution Protocol (LDP) teilt jedem Pfad die entsprechenden Labels zu. Sie bilden untereinander die Label Switching Path (LSP), also die Pfade die einem Paket durch das Netzwerk zur Verfügung stehen. Vom VPN selber wissen diese Geräte nichts.

PE (Provider Edge):PE-Router verbinden das MPLS interne Netzwerk mit den aussenstehenden Netzwerken. Sie sind hauptverantwortlich für den Aufbau der VPNs. Für jeden VPN unterhalten sie ein eigenes VRF. Diese Router tauschen per interior Border Gateway Protocol (iBGP) für jeden VPN die entsprechenden Routen aus.

CE (Customer Edge):CE-Router befinden sich ausserhalb des MPLS Netzwerkes und sind immer mit einem PE verbunden. Sie lernen die entsprechenden Routen für den VPN entweder vom PE per Routingprotokoll oder sie werden statisch eingetragen. CEs befinden sich am Standort des externen Betreibers (z.B. eines Kunden).

RR (Route Reflector):Für einen MPLS VPN müssen PEs, die an einem gemeinsamen VPN beteiligt sind per iBGP miteinander kommunizieren. Mit steigender Anzahl PEs wird die Konfiguration immer aufwendiger, da die iBGP Neighborships standartmässig full meshed sind. Der RR löst dieses Problem, da mit ihm jeder PE Router nur noch eine Neighborship mit ihm aufbauen muss und die Routen entsprechend propagiert.

Features

Mit MPLS VPNs können mehrere IP-Netzwerke über ein gemeinsames MPLS Netzwerk verbunden werden, die sich nicht gegenseitig beeinflussen. Aus Sicht des IP-Netzwerks ist der MPLS VPN nur ein „grosser Router“, der sich über mehrere Standorte erstreckt. Im Gegensatz zu einem IPSec VPN, kann bei MPLS durch die LSP garantiert werden, dass alle Pakete den gleichen Pfad nehmen. Dadurch erreicht ein VPN per MPLS eine höhere Stabilität. Jeder PE-Router unterhält pro VPN ein eigenes VRF um zu verhindern, dass sich Routen der einzelnen VPNs vermischen. So können auf einem PE-Router in verschiedenen VPNs auch die gleichen Netzwerke existieren, was mit normalen IPSec VPNs nicht möglich ist. Die P-Router innerhalb des Netzwerks müssen von den einzelnen VPNs nichts wissen, sie arbeiten unabhängig. Dies macht das Hinzufügen eines neuen PE zu einem VPN deutlich einfacher. Ein weiteres Feature von MPLS VPNs ist die Möglichkeit einzelne Pakete in verschiedene „Class of Service“(CoS) einzuteilen und so zu priorisieren.

5 NSO Service Development

Der folgende Abschnitt befasst sich mit der Entwicklung von NSO Services mit Python und der Erstellung von NETCONF NEDs.

5.1 NETCONF NED

Damit das NSO mit den Netzwerkgeräten über NETCONF kommunizieren kann, benötigt es einen NED. Für jedes unterschiedliche Gerätemodell wird ein eigener NED benötigt. NETCONF-NEDs können aus den YANG Files selber kompiliert werden. Ein solcher NED wird dem NSO in Form eines Packages zur Verfügung gestellt.

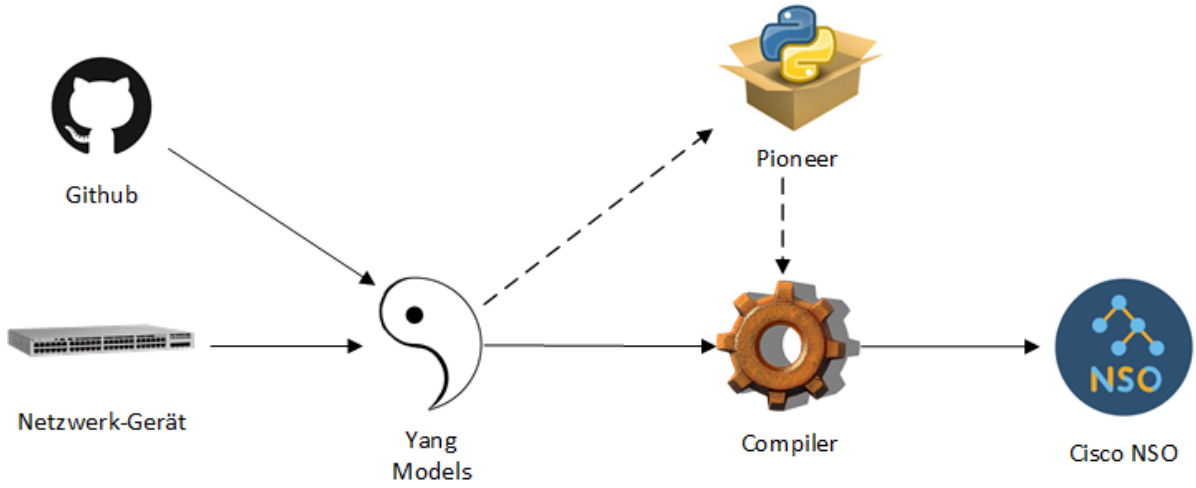

Abbildung 10: NED Build Pipeline

5.1.1 Die YANG Models

Um an die YANG Models eines Gerätes heranzukommen, gibt es zwei Varianten. Einerseits gibt es das YANG GitHub-Repository [1], welches eine Sammlung von standardisierten und auch verkäuferspezifischen Models anbietet. Allerdings, wenn man hier ein Model findet, welches für eine spezifische Betriebssystemversion eines Verkäufers bestimmt ist, heisst das nicht unbedingt, dass jedes Gerät mit der entsprechenden Version die entsprechenden YANG Models unterstützt. Je nach Gerätemodell können andere Funktionen zur Verfügung stehen. Dadurch kann der daraus generierte NED unnötig viele Funktionen anbieten und so das NSO ausbremsen, oder überhaupt nicht funktionieren.

Die andere Möglichkeit, die Models zu bekommen, ist direkt vom Gerät. Für NSO existiert das Paket Pioneer [2], welches die Models per NETCONF vom gewünschten Gerät herunterladen kann. Der Download mit Pioneer funktionierte allerdings nur mit einer globalen NSO Installation (beschrieben in der Installationsanleitung). Ein anderes Tool, welches den Download der YANG-Models anbietet, ist der Advanced Netconf Explorer (ANX) [3]. Der ANX kann die Models nicht nur downloaden, sondern bietet auch eine Benutzeroberfläche an, mit der die Models durchstöbert werden können.

[1] https://github.com/YangModels/yang
[2] https://github.com/NSO-developer/pioneer
[3] https://github.com/cisco-ie/anx

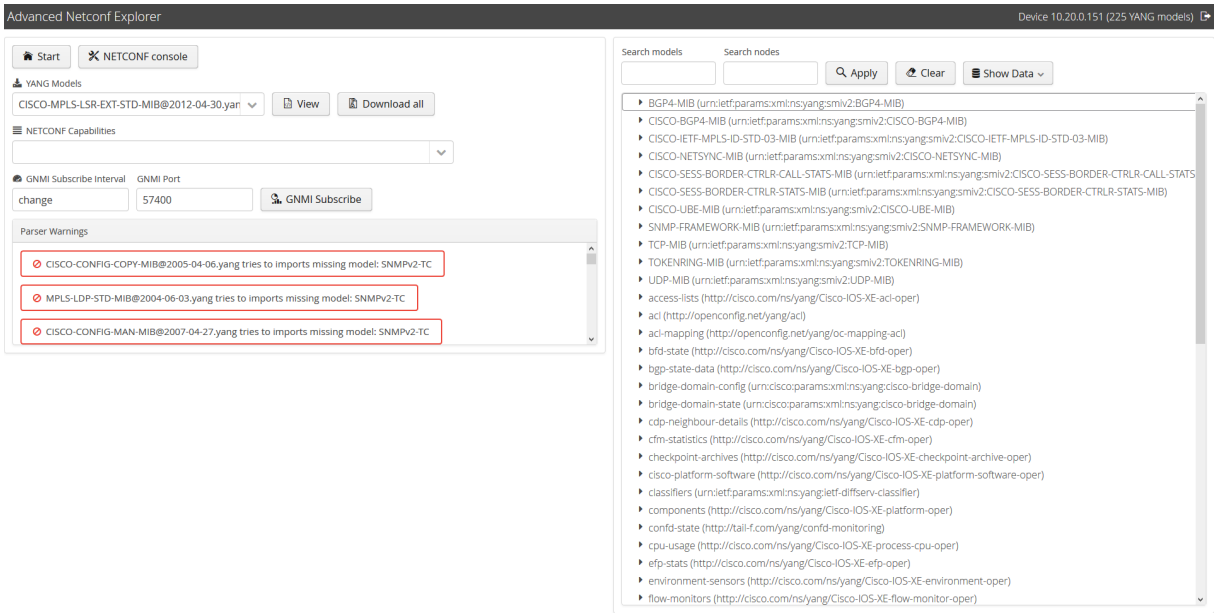

Abbildung 11: Das Benutzerinterface vom ANX

Auf Abbildung 11 sind noch einige Parser-Fehlermeldungen zu sehen. Diese stammen von fehlenden SNMP-MIBS, die nicht auf dem Gerät gespeichert waren. Die fehlenden MIBS können von dem YANG Github Repository bezogen werden. Sie sind in diesem Fall für die erfolgreiche Kompilierung des NEDs vonnöten.

5.2 Kompilieren & Installieren des NEDs

Wie auch bei den YANG Models, kann Pioneer beim Kompilieren des NEDs assistieren. Es liest die Models ein und reicht diese dem Compiler weiter, um das Package für NSO zu erstellen. Alternativ kann auch per ncs-make-package Befehl ein Package manuell erstellt und von Hand kompiliert werden. Falls die Models direkt vom Gerät bezogen wurden, müssen die Fehlenden noch nachgeliefert werden. Es kann sein, dass ein Model nicht für den NED geeignet ist, und der Compiler deswegen abbricht. Bei unseren Cisco-Geräten musste das Model „Tail-f-Conft-Monitoring“ entfernt werden. Pioneer bietet einen einfachen Befehl zum Ausschliessen einzelner YANG Files. Falls der Compiler direkt verwendet wird, muss einfach das entsprechende File gelöscht werden.

Nach dem Kompiliervorgang kann das Package ins NSO importiert werden und erlaubt die Kommunikation mit den Netzwerkkomponenten per NETCONF. Nach der Installation des NEDs hatte unser NSO einige Probleme mit dem Webportal. Die Konfigurationsseiten für die Netzwerkgeräte benötigten merklich länger zum Laden und die Metadaten Konfigurationsseite reagierte gar nicht mehr. Wir vermuten dies liegt dies daran, dass das NSO die Weboberfläche dynamisch aus den ihm verfügbaren YANG Models generiert und es durch die neu hinzugekommenen Models überfordert wurde. Auf die NSO CLI hat dies allerdings keinen Einfluss.

5.2.1 Beispielerstellung eines NEDs

Erstellung mit Pioneer

Sämtliche Commands werden in der NCS-CLI eingegeben und das Pioneer-Package muss dem NSO hinzugefügt worden sein.

```
admin@ncs# devices device device1 pioneer netconf hello #Test Netconf
admin@ncs# devices device device1 pioneer yang fetch-list
admin@ncs# devices device device1 pioneer yang download
admin@ncs# devices device device1 pioneer yang build-netconf-ned
admin@ncs# devices device device1 pioneer yang install-netconf-ned
admin@ncs# packages reload
```

Auflistung 1: Erstellung mit Pioneer

Erstellung ohne Pioneer

Ohne Pioneer müssen die YANG Models über einen anderen Weg bezogen werden. Entweder vom Gerät (z.B. per ANX) oder direkt aus dem YANG-Repository. Mittels dem Make Befehl in einer Bash Shell das Package anschliessend kompiliert werden und mit dem NSO CLI das Package geladen werden.

```
user@host$ ncs-make-package --netconf-ned <YANG_DIR> \
                            --no-java <PACKAGE_NAME>
user@host$ make -C <PACKAGE_NAME>/src
user@host$ ncs-cli -u admin -C
admin@ncs# packages reload
```

Auflistung 2: Erstellung von Hand

5.3 Erstellung eines einfachen Service im NSO

Die Erstellung eines NSO Service beinhaltet mehrere Schritte, welche nötig sind, damit dieser schlussendlich als Package geladen werden kann. Zu Beginn wird ein leeres Service-Package-Skelett generiert. Im nächsten Schritt wird das generierte XML Template und YANG Model mit den gewünschten Änderungen ergänzt. Falls nötig, kann zusätzliche Logik in Form von Python Code hinzugefügt werden. Zum Schluss wird das Package kompiliert, in den `packages` Ordner der NSO Instanz verschoben und geladen.

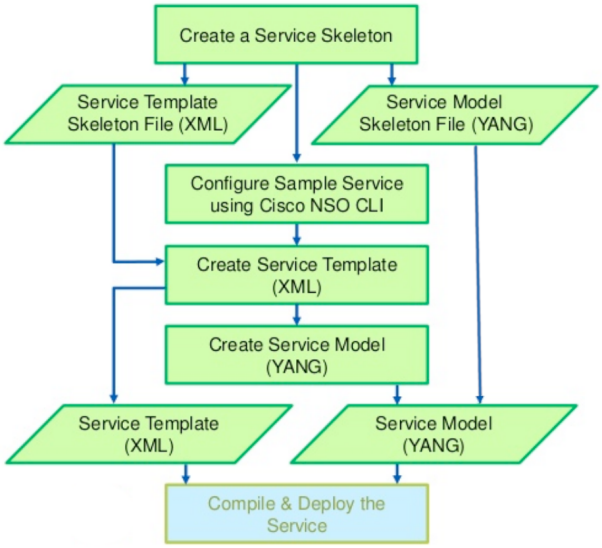

Abbildung 12: Erstellungsablauf eines Services [Can16]

Service Skelett erstellen

Für das Erstellen eines Service-Skeletts wird der Befehl `ncs-make-package` verwendet. Er erstellt sämtliche benötigten Dateien und der Service könnte so bereits ins NSO importiert werden. Beim Erstellen des Skeletts kann angegeben werden, ob ein Service mit Python, Java oder nur mit einem Template erstellt werden soll. Abbildung 13 zeigt ein Beispiel anhand eines Python Service.

```
→  packages ncs-make-package --service-skeleton python-and-template awesomeService
→  packages tree awesomeService
awesomeService
├── package-meta-data.xml
├── python
│   └── awesomeService
│       ├── __init__.py
│       └── main.py
├── README
├── src
│   ├── Makefile
│   └── yang
│       └── awesomeService.yang
├── templates
│   └── awesomeService-template.xml
└── test
    ├── internal
    │   ├── lux
    │   │   ├── Makefile
    │   │   └── service
    │   │       ├── dummy-device.xml
    │   │       ├── dummy-service.xml
    │   │       ├── Makefile
    │   │       ├── pyvm.xml
    │   │       └── run.lux
    │   └── Makefile
    └── Makefile

9 directories, 15 files
→  packages
```

Abbildung 13: Beispiel eines Service Skeletts mit Python

Template im NSO erstellen

Damit der erstellte Service überhaupt etwas bewirken kann, wird in diesem Schritt das generierte Template um die gewünschte Konfiguration ergänzt.

Die Konfigurationsänderungen an einem Gerät werden zunächst über die CLI im NSO getätigt und vor dem Commit mittels `commit dry-run outformat xml` auf die Konsole ausgegeben. Von dieser Ausgabe wird der gesamte Inhalt, welcher zwischen den `<config></config>` Tags vorhanden ist, in das `awesomeService-template.xml` File kopiert. Das `xmlns` Attribut verweist auf das jeweilige YANG-Model, auf welches das Template angewendet wird. Falls diese Models nicht im NSO vorhanden sind, schlägt das Laden des Service-Packages fehl.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{/device}</name>
      <config>
        <interface xmlns="urn:ios">
          <GigabitEthernet>
            <name>0/23</name>
            <switchport>
              <mode>
                <trunk />
              </mode>
              <trunk>
                <allowed>
                  <vlan>
                    <vlans>11</vlans>
                  </vlan>
                </allowed>
              </trunk>
            </switchport>
          </GigabitEthernet>
        </interface>
      </config>
    </device>
  </devices>
</config-template>
```

Abbildung 14: awesomeService Template

```xml
<?xml version="1.0" encoding="UTF-8"?>
<config-template xmlns="http://tail-f.com/ns/config/1.0">
  <devices xmlns="http://tail-f.com/ns/ncs">
    <device>
      <name>{/endpoint/device}</name>
      <config>
        <interface xmlns="urn:ios">
          <GigabitEthernet>
            <name>{interface}</name>
            <switchport>
              <mode>
                <trunk />
              </mode>
              <trunk>
                <allowed>
                  <vlan>
                    <vlans>{/vlan}</vlans>
                  </vlan>
                </allowed>
              </trunk>
            </switchport>
          </GigabitEthernet>
        </interface>
      </config>
    </device>
  </devices>
</config-template>
```

Abbildung 15: Template mit Parameter

Dieses Template könnte bereits verwendet werden, um Einstellungen an Netzwerkkomponenten vorzunehmen. Diese Änderungen sind jedoch komplett statisch und würden bei jedem Gerät die identische Konfiguration vornehmen. Abbildung 14 zeigt den momentanen Stand vom Template.

Service Template Parameter

Damit dieses Template dynamisch verwendet werden kann, wird dieses durch Parameter ergänzt. Diese Parameter werden entsprechend abgefüllt und ermöglichen unterschiedliche Konfigurationen mit dem gleichen Template. Theoretisch kann in dem Template auch einfache Logik wie If-Else Blöcke oder For-Loops verwendet werden.

Nun können Konfigurationswerte dynamisch ins Template eingefügt werden. Abbildung 15 zeigt dafür ein Beispiel. Damit das NSO weiss, welche Variablen mit welchen Werten abgefüllt werden sollen, muss nun das YANG Model des Services entsprechend erweitert werden.

YANG Service Model

Das YANG Model gibt dem NSO die Datenstruktur des Services vor. Damit von dem Service mehrere Instanzen erstellt werden können, wird zuoberst eine Service-Liste mit einem Namen als Key erstellt. Jede Liste benötigt einen Key als einzigartigen Identifier. In dieser Liste können nun die benötigten Variablen definiert werden. Pro Template-Variable wird im Model ein Leaf mit dem entsprechenden Typ und Namen erstellt. Diese Leafs können auch wieder in Listen gruppiert werden um zum Beispiel mehrere Geräte in einem Service verwalten zu können. Abbildung 16 zeigt einen Ausschnitt aus dem

```
list awesomeService {
  key name;
  leaf name {
    tailf:info "Service name";
    tailf:cli-allow-range;
    type string;
  }
  uses ncs:service-data;
  ncs:servicepoint awesomeService-servicepoint;
  list endpoint {
    key device;
    leaf device{
      type leafref {path "/ncs:devices/ncs:device/ncs:name";}
    }
    leaf interface{type string;}
  }
  leaf vlan {type uint16;}
}
```

Abbildung 16: Beispiel eines YANG-Models

YANG-File das dem XML-Template von oben entspricht. Tabelle 1 zeigt eine Übersicht der grundlegendsten YANG-Statements.

Name	Beschreibung
Leaf	Beinhaltet einen Wert eines bestimmten Typs
Leaf-List	Beinhaltet eine Liste von Werten eines bestimmten Typs
Container	Kann keine oder mehrere Werte und andere Statements enthalten
List	Beinhaltet eine Liste von null oder mehr Werten und anderen Statements
Leafref	Beinhaltet eine Verknüpfung auf ein Statement an einem anderen Ort

Tabelle 1: Übersicht der grundlegendsten YANG-Statements

Fertigstellung
Zum Abschluss wird das Package mit dem Make-Tool kompiliert und mittels `packages reload` in der NSO CLI oder der Weboberfläche geladen. Dabei muss beachtet werden, dass sich der erstelle Service-Order im Packages Directory vom NSO befindet. Treten bei der Kompilierung, oder beim Reload der Packages Fehler auf, ist der Service fehlerhaft und muss korrigiert werden.

```
→  packages make -C awesomeService/src
make: Entering directory '/home/lauch/ncs-run/packages/awesomeService/src'
mkdir -p ../load-dir
mkdir -p java/src//
/home/lauch/ncs-4.7/bin/ncsc  `ls awesomeService-ann.yang  > /dev/null 2>&1 && echo "-a awesomeService-ann.yang"` \
              -c -o ../load-dir/awesomeService.fxs yang/awesomeService.yang
make: Leaving directory '/home/lauch/ncs-run/packages/awesomeService/src'
→  packages ncs_cli -u admin -C

admin connected from 192.168.101.25 using ssh on lauch-Virtual-Machine
admin@ncs# packages reload

>>> System upgrade is starting.
>>> Sessions in configure mode must exit to operational mode.
>>> No configuration changes can be performed until upgrade has completed.
>>> System upgrade has completed successfully.
reload-result {
    package awesomeService
    result true
}
reload-result {
    package cisco-ios
    result true
}
admin@ncs#
```

Abbildung 17: Kompilierung des Services und Laden im NSO

5.3.1 Python Komponente

Wenn zusätzlich zu dem Template Logik in Form von Python Code verwendet werden will, muss im `package-meta-data.xml` die Pythonklasse angegeben werden. Nun wird der Service immer zuerst den Python Code ausführen, bevor er das Template abfüllt. Dies bedeutet auch, dass das Template nicht mehr von sich aus gefüllt wird. Das muss nun auch in der Python Komponente geschehen. Dies ist im Beispiel auf Abbildung 18 zu sehen.

Zusätzlich zur Methode, welche für das Abfüllen zuständig ist, gibt es auch die Möglichkeit weitere Hook Funktionen zu definieren, unter anderem für das Starten und Stoppen des Services.

```python
class ServiceCallbacks(Service):

    # The cb_create() callback is invoked inside NCS FASTMAP and
    # must always exist.
    @Service.create
    def cb_create(self, tctx, root, service, proplist):
        self.log.info('Service create(service=', service._path, ')')

        vars = ncs.template.Variables()
        vars.add('DUMMY', '127.0.0.1')
        template = ncs.template.Template(service)
        self.log.info('Template is being applied')
        template.apply('Routing-OSPF-template', vars)
```

Abbildung 18: Beispiel einer Python Funktion

5.3.2 NSO-Model Erweiterungen

Damit unser Service möglichst automatisiert arbeiten kann, müssen die Geräte im NSO um weitere Informationen erweitert werden. Um diese Informationen speichern zu können, muss das Model der Geräte um weitere Felder erweitert werden. Dies kann mit dem YANG-Keyword `augment` erreicht werden, welches bestehende Models erweitern kann. Abbildung 19 zeigt als Beispiel die Erweiterung eines Device-Models um eine Liste.

```
augment /ncs:devices/ncs:device {
  list Connections{
    key Other-Device;
    1 reference
    leaf Other-Device{
      type leafref {path "/ncs:devices/ncs:device/ncs:name";}
    }
   //must "/ncs:devices/ncs:device-group[ncs:name='PE-Devices']/ncs:device-name[.=current()]";
    leaf interface_type {
        type enumeration{
          enum "FastEthernet";
          enum "GigabitEthernet";
          enum "TenGigEthernet";
        }
      }
      leaf interface_number {type string;}
  }
}
```

Abbildung 19: Beispiel eines Augments

5.3.3 Softwaretests für NSO Services

Softwaretests können sich in im Entwicklungsprozess als überaus nützlich erweisen. Sie bieten die Möglichkeit, das Verhalten der Software automatisiert zu überprüfen und detaillierte Berichte zu erstellen. Das kann massiv Zeit sparen und die Erkennung von neuen Fehlern vereinfachen. Jedoch ist die Erstellung solcher Tests nicht immer trivial und kann viel Zeit in Anspruch nehmen.

Für NSO Services bietet sich dafür LUcid eXpect scripting (LUX)[4] an. LUX ist ein in Erlang geschriebenes Testing-Framework und kann verwendet werden, um skriptbasierte Tests zu erstellen. In einem

[4] https://github.com/hawk/lux

Test können Shells gestartet werden, in welchen gewünschte Commands ausgeführt werden. Mittels Regex-Matching kann pro Befehl in der Kommandozeile den Output entsprechend überprüft werden.

Normalerweise sind bei der konventionellen Softwareentwicklung solche Softwaretests durchaus vorgesehen, da sie sich trotz zusätzlichem Zeitaufwand in der Zukunft schnell wieder auszahlen können. Der Fokus dieser Arbeit liegt jedoch auf der Erstellung einer Softwarelösung, welche die Anforderungen erfüllt. Dabei ist die Erstellung von Softwaretests nicht inbegriffen und somit werden im Rahmen dieser Arbeit keine Tests erstellt.

```
→  test git:(First-Implementation) l
total 20K
drwxr-xr-x  3 patrik patrik 4.0K Mar 21 12:46 .
drwxr-xr-x  8 patrik patrik 4.0K Mar 28 09:48 ..
drwxr-xr-x 48 patrik patrik 4.0K Mar 28 09:56 lux_logs
-rw-r--r--  1 patrik patrik  427 Mar 21 11:17 Makefile
-rw-r--r--  1 patrik patrik  504 Mar 21 12:49 nso-load-check.lux
→  test git:(First-Implementation) make
lux *.lux
summary log       : /home/patrik/Desktop/ncs-run/packages/MPLS-L3VPN/test/lux_logs/run_201
9_03_28_08_57_25_74345/lux_summary.log

test case         : nso-load-check.lux
progress          : ..:..:....:.:...:.:.22?:?.z.....:...:.:..:..:.....
result            : SUCCESS

successful        : 1
summary           : SUCCESS

file:///home/patrik/Desktop/ncs-run/packages/MPLS-L3VPN/test/lux_logs/run_2019_03_28_08_57
_25_74345/lux_summary.log.html
→  test git:(First-Implementation)
```

Abbildung 20: Beispiel eines Lux Tests

5.4 FASTMAP und erweiterte Funktionen

Bis hierhin wurde immer nur das Erstellen eines NSO Services beschrieben, was die `create` Methode in der Python Komponente aufruft. Dies liegt daran, dass ein NSO Service sich nur um diesen einen Fall kümmern soll. Bei einer Änderung eines bestehenden Services wird er aufgerufen, als wäre er gerade mit den neuen Parametern erstellt worden.

FASTMAP kann die `create` Methode des NSO Services auch mehrmals hintereinander aufrufen, zum Beispiel bei einem Commit über das interne WebUI des NSO. Deswegen ist es wichtig, dass die `create` Methode bei jedem Durchlauf die gleiche Konfiguration erzeugt. Solange der NSO Service keine Nebeneffekte hat, ist dies kein Problem. Für unsere Arbeit müssen wir aber IP-Adressen aus dem Infoblox beziehen, was die erzeugte Konfiguration davon abhängig macht. Einerseits kann die IP, die uns Infoblox liefert ändern, anderseits wollen wir verhindern, dass für eine Service Instanz mehrere IPs reserviert werden. Um dies mit FASTMAP in Einklang zu bringen, wird ein neues Desing Pattern benötigt: Reactive FASTMAP.

5.4.1 Reactive FASTMAP

Die Grundidee von Reactive FASTMAP ist, die Nebeneffekte in ein anderes Package, hier Ressource-Manager genannt, auszulagern. So können die Aufgaben mit Nebeneffekten unabhängig vom eigent-

lichen Service ausgeführt werden. Dies beschleunigt auch die Ausführung von `create`, da langsame Requests auf externe Ressourcen im Ressource-Manager ausgeführt werden.

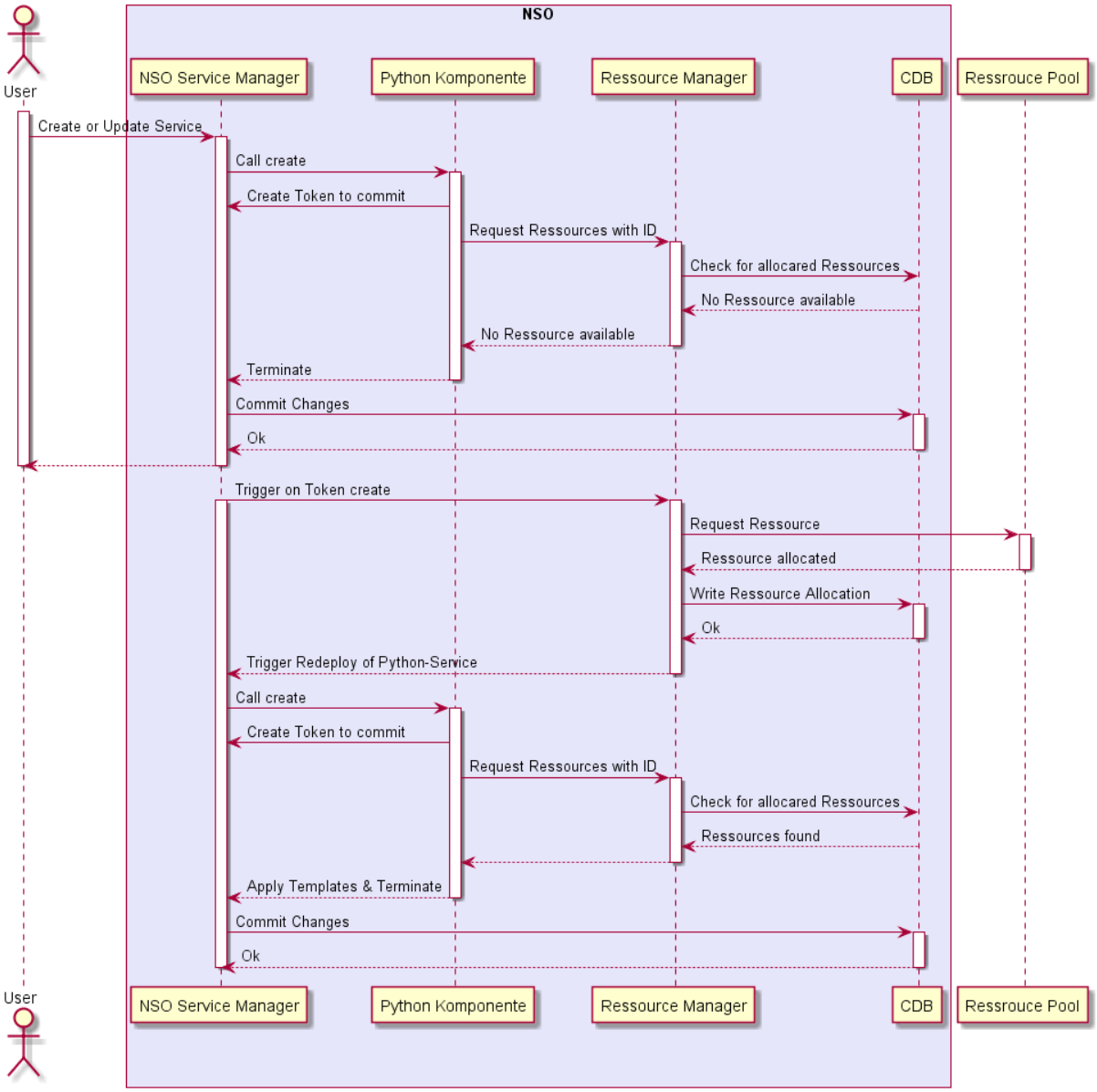

Abbildung 21: Reactive FASTMAP

Der Ressource Request vom Python Service zum Ressource-Manager funktioniert über spezielle Tokens in der CDB. Diese Tokens bestehen aus der eindeutigen ID einer Service-Instanz, welche als Liste in der CDB gespeichert werden. Der Ressource-Manager subscribed auf diese Liste, sodass er bei jeder Änderung benachrichtigt wird. Wird ein neues Token per Commit erstellt, kann er die Ressourcen anfragen, ohne das der Python Service blockiert wird. Die Resultate schreibt er ebenfalls in die CDB. Anschliessend startet der Ressource-Manager ein Redeploy. Der Python Service bekommt nun vom Ressource-Manager die benötigten Daten und kann seine `create` Methode abschliessen. Abbildung 21 zeigt ein einfaches Beispiel.

Ein Problem von Reactive FASTMAP ist, dass der Benutzer keine Informationen über den Verlauf erhält. Falls ein Fehler in der ersten Ausführung der Python Komponente auftritt, wird er noch benachrichtigt,

die anderen beiden Aktionen werden aber im Hintergrund ausgeführt. Um dieses Problem zu lösen, führt Reactive FASTMAP das Konzept von Plans ein. In einem Plan werden die einzelnen Schritte und dessen Status festgehalten.

5.4.2 Python APIs

Um Reactive FASTMAP zu implementieren, werden zusätzliche APIs vom NSO benötigt. Wir werden hier nur diejenigen kurz beschreiben, welche wir für unsere Umsetzung benötigt haben.

CDB API
Die CDB API bietet die Möglichkeit über einen TCP-Socket direkt auf die CDB zuzugreifen. Für die BA haben wir diese API benötigt, um Konfiguration während der Erstellung einer Service Instanz direkt in die CDB zu schreiben, ohne dass sie beim Löschen desselben Instanz durch FASTMAP wieder entfernt wird. Zusätzlich verwenden wir auch die Möglichkeit auf bestimmte Änderungen in der CDB zu subscriben, sodass wir auf diese reagieren können.

DP API
Die DP API ermöglicht eine Reihe an zusätzlichen Callbacks und Hooks. Sie bietet die Möglichkeit das Redeploy einer bestehenden Service Instanz anzustossen, was bei der Implementation von Reactive FASTMAP benötigt wird. Wir verwenden sie für das Erstellen von zusätzlichen eigenen NSO Actions.

MAAPI API
Diese API ist eine High-Level Implementation der CDB API. Sie vereinfacht es Daten zu schreiben und zu lesen. Sie kann in der Create Methode eines NSO Services verwendet werden um einfach Daten aus dem NSO auszulesen. Da sie ein Transaction-Lock verwendet, kann sie nicht verwendet werden, um Daten während eines Commits an FASTMAP vorbei in die CDB zu schreiben. Das Kapitel 6.7.1 beschreibt diese Problematik genäuer.

5.4.3 Log-Level Einstellungen

Die Python Komponente bietet mehrere Log-Levels. Um zwischen diesen umzuschalten muss die NCS-CLI mit dem Parameter -J gestartet werden.

```
user@host$ ncs_cli -J
admin@ncs% config
admin@ncs% set python-vm logging level level-debug
admin@ncs% commit
```

Auflistung 3: Änderung Log-Level

5.4.4 CDB

Es gibt zwei Möglichkeiten, Daten in der CDB zu speichern: Config-Data und Oper-Data (Operational Data). Config-Data sind sämtliche Daten, die vom Benutzer verwendet werden um das NSO oder die verwalteten Geräte zu konfigurieren. Oper-Data hingegen besteht, wie der Name schon sagt, aus Daten die für den Betrieb des NSO und dessen Services wichtig sind. Dazu gehören unter anderem Statistiken oder, in unserem Fall, der State des InfobloxProx. Oper-Data kann von extern nicht verändert werden, weder über das Web-UI noch über die Schnittstellen. Um Daten als Oper-Data zu markieren, muss

im YANG-Model die entsprechende Node mit der Annotation `config: false;` versehen werden. Standardmässig sind Oper-Daten nicht persistent und werden bei einem Neustart des NSO gelöscht. Um sie persistent zu machen, kann `tailf:cdb-oper{tailf:persistent true;}` verwendet werden. Abbildung 22 zeigt ein Beispiel aus unserem Service. Darin werden beide Leafs und die List als Oper-Data behandelt, da ihr Parent-Container als solche markiert wurde.

```
container infoblox_state {
    config false;
    tailf:cdb-oper{
        tailf:persistent true;
    }
    list reservation_tokens {
        key token;
        leaf token {
            type string;
        }
    }

    leaf external_network_reservations {
        type string;
    }

    leaf transit_network_reservations {
        type string;
    }
}
```

Abbildung 22: Beispiel eines YANG-Models mit Oper-Data

5.4.5 NSO Actions

NSO bietet auch die Möglichkeit in einem NSO Service spezielle Actions zu erstellen. Diese Actions können von extern gestartet werden und beliebige Funktionen enthalten, wie zum Beispiel das Testen von Konfiguration oder Resetten von Caches, sie sollen aber nie die Konfiguration des NSO verändern.

Action YANG Model
Damit NSO die Action kennt, muss sie im YANG Model des Services eingetragen sein. Dort können die Inputs und Outputs der Action definiert werden. Zudem wird auch ein Actionpoint definiert. Auf dessen Namen wird später im Python Code die Action registriert. Abbildung 23 zeigt ein Beispiel einer Action im Model.

Action Python Code
Zusätzlich zu der Definition im Model, muss die Action noch implementiert werden. Dies geschieht mit dem Erstellen einer neuen Klasse im Python Code, welche die Funktion `cb_action` besitzt. Abbildung 24 zeigt ein Beispiel. Zusätzlich dazu muss in der `setup` Methode der `main` Klasse der Actionpoint aus dem YANG Model mit der neuen Klasse verbunden werden. Dies geschieht mit der Methode `register_action`. In diesem Beispiel wäre das:

```
self.register_action('MPLS-L3VPN-TestInfoblox', TestInfobloxAction)
```

```
tailf:action test_connection {
    tailf:actionpoint MPLS-L3VPN-TestInfoblox;
    input {
    }
    output {
        leaf status_code {
            type uint16;
        }
        leaf success {
            type boolean;
        }
        leaf body {
            type string;
        }
    }
}
```

Abbildung 23: Beispiel einer Action im YANG Model

```
class TestInfobloxAction(Action):
    @Action.action
    def cb_action(self, uinfo, name, kp, input, output):
        try:
            response = InfobloxProx.test_request()
            output.status_code = response.status_code
            output.body = response.text
            output.success = response.status_code == 200
        except Exception as excep:
            output.success = False
            output.body = str(excep)
```

Abbildung 24: Beispiel einer Action im Python Code

6 Architektur

6.1 Deploymentdiagramm

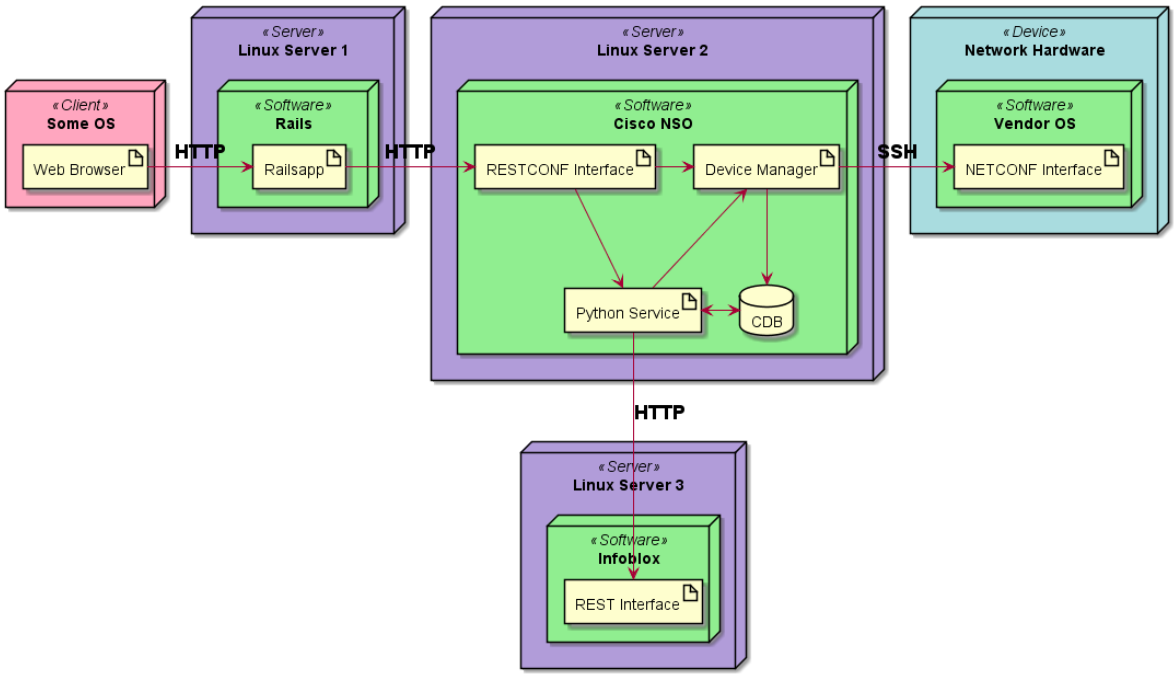

Abbildung 25: Deploymentdiagramm

Abbildung 25 zeigt eine mögliche Variante, wie das gesamte System deployed werden kann. Die drei Softwareteile, „Rails“, „Cisco NSO“ und „Infoblox“ könnten theoretisch auch auf einem Server in Betrieb genommen werden, solange diese untereinander mittels HTTP-Verbindung kommunizieren können.

6.2 Prototyp

Der Prototyp einer Anwendung ist eine minimale Implementierung der gewünschten Funktionen. Er dient dazu, um auszutesten, ob sämtliche Komponenten der Anwendung durch die gewählten Technologien umsetzbar sind und die entworfene Architektur mit den Anforderungen verwendet werden kann. Insbesondere bei der Verwendung von neuen Technologien ist es wichtig, vorerst die erdachten Konzepte in der Praxis zu erproben. Der Prototyp vermittelt dem Auftraggeber zudem einen ersten Eindruck und dieser kann zu einem frühen Zeitpunkt bereits Feedback einbringen.

Im Rahmen dieser Arbeit kann der Prototyp zudem passend verwendet werden, um an der Zwischenpräsentation der Bachelorarbeit den momentanen Stand der entwickelten Software zu zeigen. Dementsprechend besteht die Vorgabe, dass der Prototyp bis zu diesem Datum entwickelt wurde.

6.2.1 Umfang des Prototyps

Unser Ziel ist es, für den Prototyp den Use Case 1.1 zu implementieren, denn dieser eignet sich vom Umfang her perfekt für den Prototyp. Er beinhaltet die grundlegenden Funktionen des gewünschten Endprodukts und ist dennoch simpel genug, um innerhalb der Ellaboration-Phase umgesetzt zu werden. Dieser Use Case beinhaltet kurzgefasst die Erstellung eines einfachen VPN Services im NSO mit GUI Komponente.

6.3 Sequenzdiagramme

Im folgenden Abschnitt werden verschiedene Sequenzdiagramme dargestellt, welche die jeweiligen Interaktionen zwischen den verschiedenen Komponenten wie Service-Manager, Python Komponente, Infoblox, GUI-Komponente und Browser des Benutzers darstellen. Die Frontend Komponenten (Browser und Rails App) werden in den folgenden Diagrammen zusammengefasst als GUI dargestellt, da der Fokus vorerst auf den Backend Komponenten liegt.

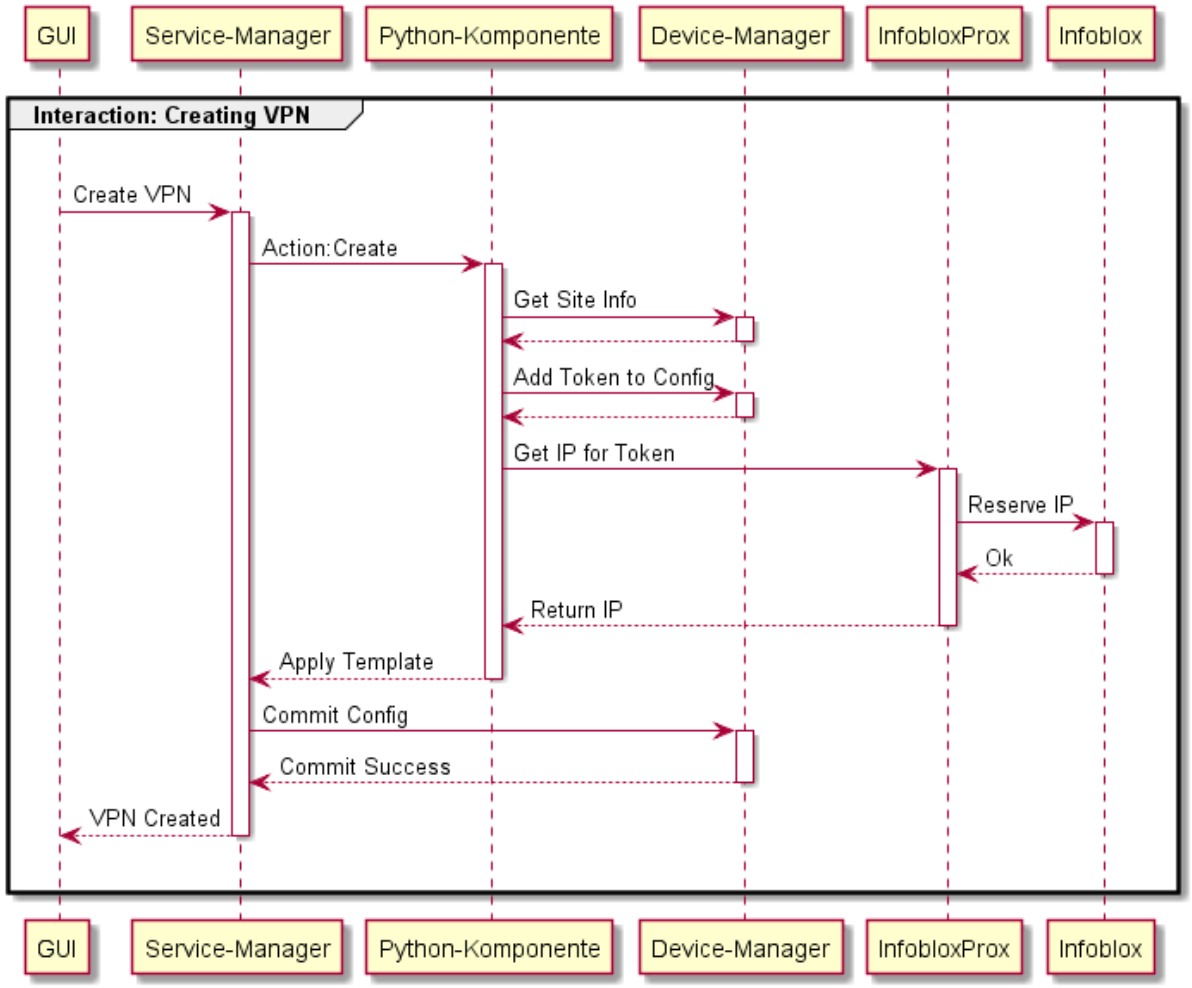

Abbildung 26: Sequenzdiagramm: Creating VPN Service

Create VPN Service

Abbildung 26 zeigt die Interaktion Creating VPN Service, die einfachste Aktion des System. Sie entspricht dem Erstellen eines VPNs im Use Case 1. Ein Benutzer will einen neuen VPN vom GUI aus erstellen. Dabei wird von der GUI Komponente, ausgelöst durch eine Benutzereingabe via Browser, eine Anfrage an das NSO gesendet. Die Python Komponente des Services holt sich zunächst die benötigten Informationen (CE & PE Router, Interfaces, ect.) aus dem Device-Manager und fügt ein einzigartiges Token in die Config ein. Dieses Token wird vom InfobloxProx verwendet, um nicht mehr verwendete Netzwerke zu kennzeichnen. Anschliessend werden die nötigen IP-Netzwerke über den Proxy vom Infoblox bezogen. Nachdem alle benötigten Informationen gesammelt wurden, werden die Templates mit den entsprechenden Konfigurationen angewendet. Als letzter Schritt werden die Änderungen im Device Manager des NSO commitet und auf die physikalischen Geräte angewendet.

Create VPN Service with wrong Config

Abbildung 27 beinhaltet zwei Aktionen. Einmal die Erstellung eines VPN Services mit einer fehlerhaften Konfiguration und anschliessend die erneute Erstellung mit korrigierter Konfiguration. Diese Aktionen zeigen den Zweck des InfobloxProx, welcher die Requests zum Infoblox kapselt. Die erste Aktion schlägt aufgrund der falschen Konfiguration fehl, allerdings hat die Service Instanz bereits die IP-Adressen beim Infoblox reserviert. Der Python Komponente wird vom NSO nicht mitgeteilt, ob der Commit erfolgreich war oder nicht. Deshalb können bei einem fehlerhaften Commit die IP Adressen nicht einfach wieder freigeben werden. Der Proxy merkt sich die IP-Adressen, welche vom Infoblox bezogen wurden und mapped diese mit dem eindeutigen Token der Service Instanz

Die zweite Aktion zeigt einen erneuten Request vom GUI, diesmal mit den korrekten Einstellungen. Die Python Komponente weiss nichts von der vorherigen Aktion und beginnt wie vorhin die Informationen zu sammeln. Der Proxy muss die Adressen nun nicht mehr vom Infoblox reservieren, sondern er kann diejenigen verwenden die er bereits gespeichert hat. So wird verhindert, dass Adressen beim Infoblox reserviert, aber bei keinem Service verwendet werden.

Token Cleanup

Die Freigabe von IPs in Abbildung 27 funktioniert nur, wenn der Benutzer nach einer falschen Anfrage auch noch eine Richtige schickt. Falls dies nicht der Fall ist, wird eine Cleanup-Aktion benötigt, wie in Abbildung 28 dargestellt. Beim Start des NSO wird der erste Token-Check ausgeführt. Dabei wird einfach die Liste der Tokens mit den gespeicherten Netzwerken verglichen. Findet sich für ein Netzwerk kein Token, existiert der korrespondierende VPN nicht mehr und die Netzwerke können beim Infoblox wieder freigegeben werden. Nach diesem ersten Cleanup wird eine Subscription auf die Tokenliste geöffnet, damit diese bei jeder Änderung neu validiert werden kann.

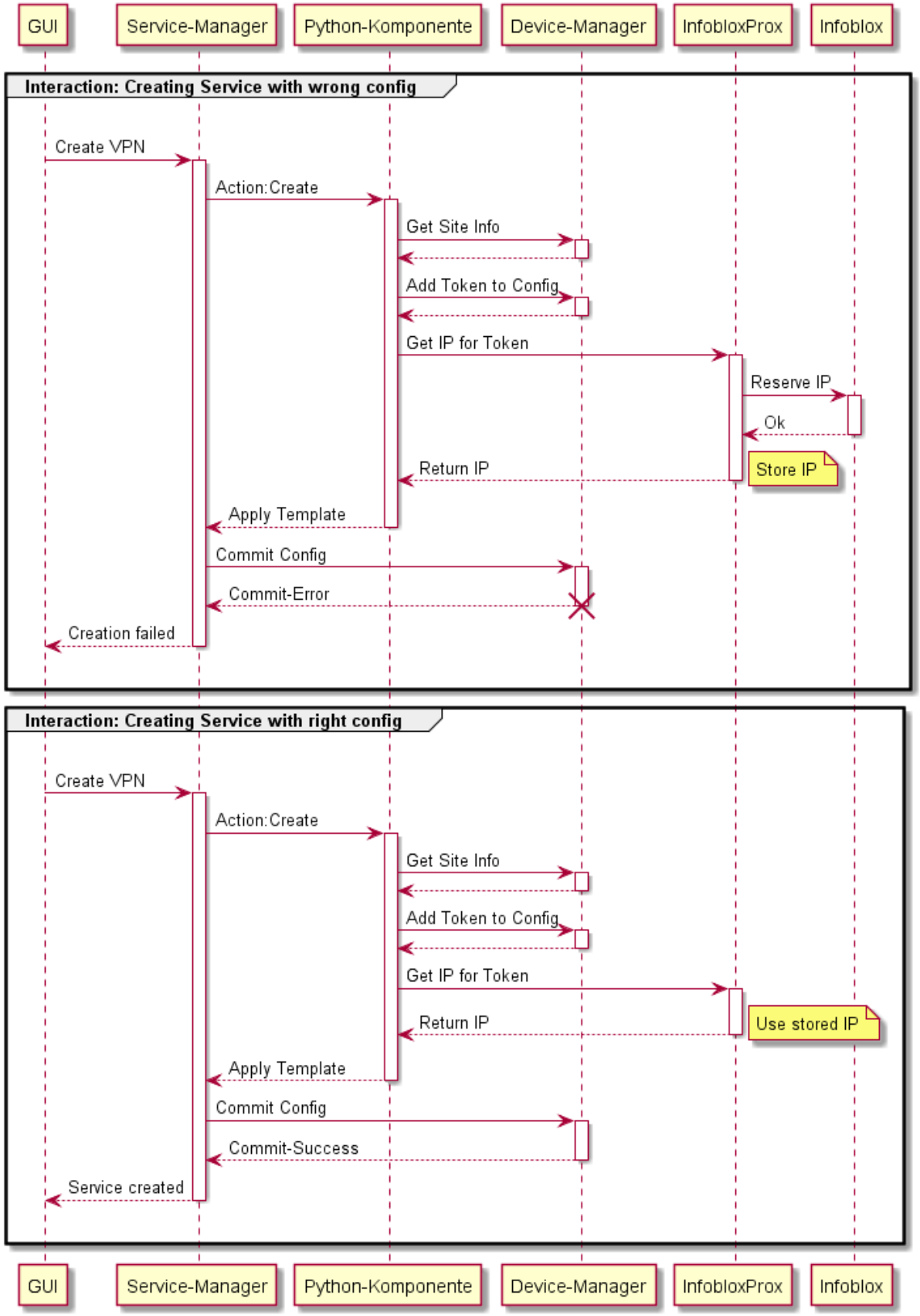

Abbildung 27: Sequenzdiagramm: Creating Service with wrong Config

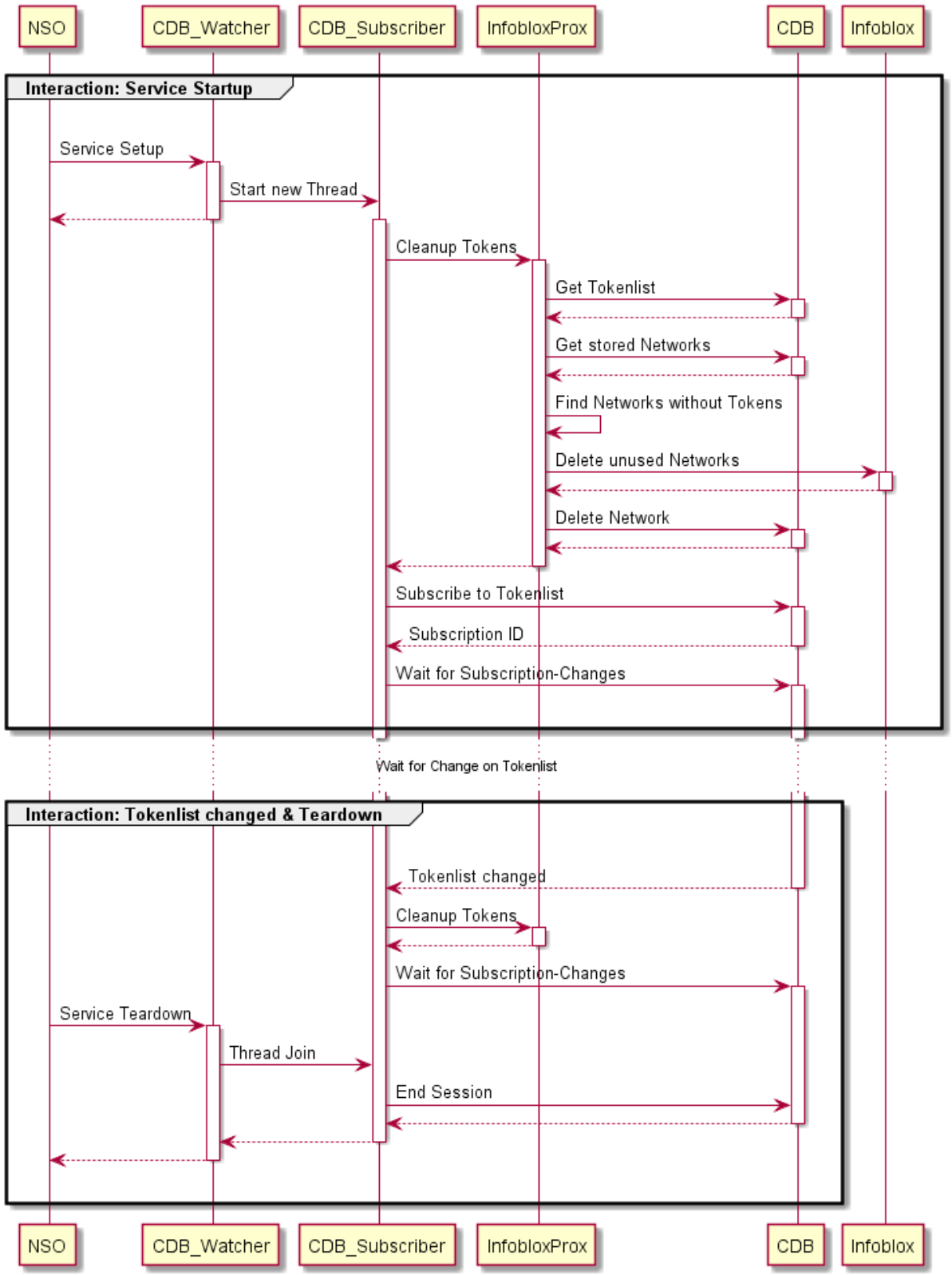

Abbildung 28: Sequenzdiagramm: Cleanup Tokens and Networks

Delete VPN Service

Das Löschen eines VPN Services ist nicht sehr kompliziert. Der Service-Manager entfernt sämtliche Konfiguration der gelöschten Service Instanz. Dabei wird auch das Token aus der Tokenlist entfernt. Der CDB_Subscriber wird darauf benachrichtigt und beginnt einen Token Cleanup. Dadurch werden die Netzwerke des gelöschten VPNs automatisch aus dem Infoblox entfernt. Die Delete-Aktion ist in Abbildung 29 dargestellt.

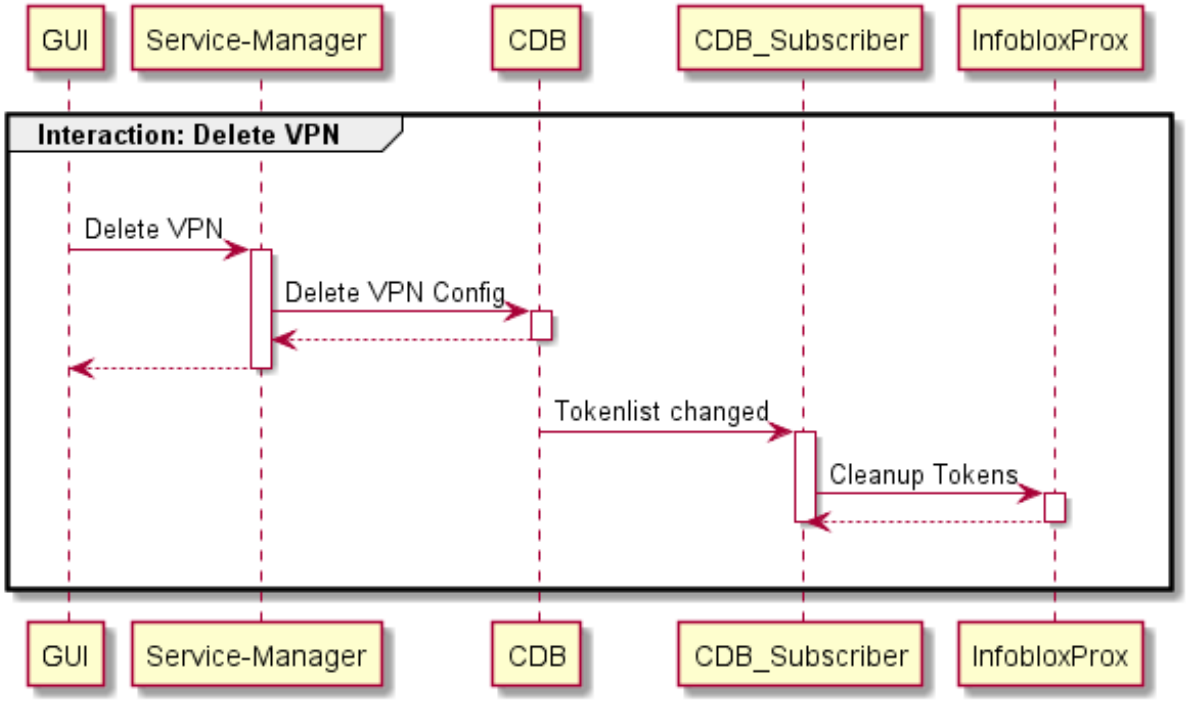

Abbildung 29: Sequenzdiagramm: Delete VPN

6.4 Web-GUI

Das Web-GUI bietet eine vereinfachte Möglichkeit, die Services zu managen. Benötigte Informationen werden vom Benutzer abgefragt, validiert und in aggregierter Form an das NSO weitergeleitet. Eventuelle Fehler, welche das NSO meldet, werden an den Benutzer rückgemeldet. Informationen zu den Services werden beim NSO abgefragt, aufbereitet und dem Benutzer zur Verfügung gestellt. Grundsätzlich ist die GUI-Komponente eine Middleware zwischen dem Benutzer und dem NSO, welche Daten aggregiert, validiert und diese anschliessend in der grafischen Oberfläche darstellt.

Folgende Tabelle beschreibt, welche Models und Services wofür zuständig sind:

Connection	Logik und Daten für Verbindungen zwischen PE und CE Router
Device	Logik und Daten für Management von Geräten im NSO
GlobalSite	Logik und Daten für Globale Sites im NSO
Service	Logik und Daten für eine spezifische Instanz eines MPLS-L3VPN Services
ServiceSite	Logik und Daten für eine spezifische Site in einem MPLS-L3VPN Service

Die folgende Grafik zeigt einen Ausschnitt der implementierten Komponenten der Rails Applikation mit entsprechenden Abhängigkeiten. Zu Gunsten der Übersicht werden generische Komponenten wie Views und Controller nicht dargestellt. Des Weiteren besitzt jeder Service eine Abhängigkeit auf das entsprechende Model mit selbigem Namen.

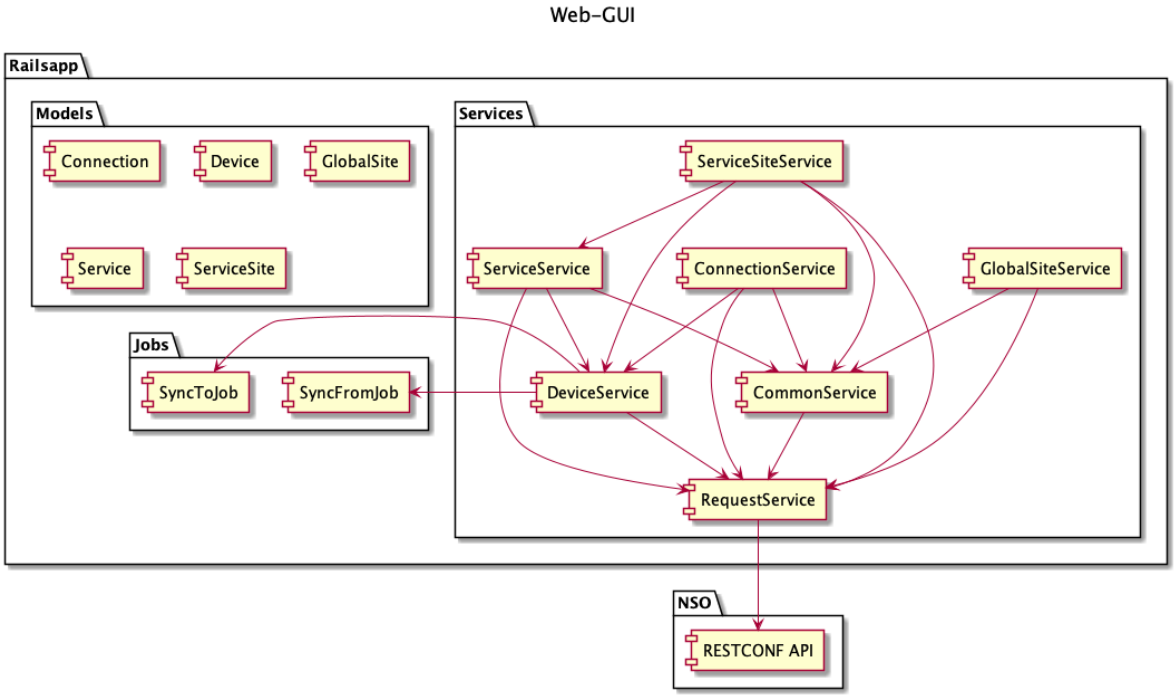

Abbildung 30: Komponenten der Rails App

6.4.1 Models

Um die Handhabung von Daten, welche vom NSO und dem Benutzer abgefragt werden, zu erleichtern, werden diese mittels Modelklassen strukturiert. Diese Models beinhalten die Attributfelder, welche zum Beispiel für einen Service benötigt werden. Daten vom NSO werden mittels der RESTCONF Schnittstelle abgefragt, de-serialisiert und damit eine Modelinstanz erstellt. Diese Modelinstanz kann nun einfach für die Darstellung der Daten in einer Website verwendet werden. Dieses Konzept funktioniert auch in der entgegengesetzten Richtung. Vom Benutzer werden Konfigurationsdaten eines Geräts an die GUI Komponente gesendet. Mittels dieser Parameter wird in der GUI Komponente eine Modelinstanz erstellt. Für die Weiterleitung an das NSO wird diese Instanz in JSON serialisiert und eine entsprechende Anfrage gesendet.

Es wäre möglich, auf diese Models zu verzichten und die einzelnen Parameter jeweils ohne Serialisierung auszutauschen. Diese Models sind jedoch relativ leicht erstellt und bieten grossen Nutzwert. Sie sind vergleichbar mit den YANG Models, welche das NSO für die interne Datenspeicherung, die Kommunikation mit den Netzwerkgeräten und die RESTCONF Schnittstelle für das GUI verwendet.

6.4.2 Services

Um den Programmcode besser zu strukturieren, wird die Programmlogik in sogenannte Services aufgeteilt. Dies sind lediglich einzelne Untermodule in der Rails App, welche spezifische Aufgaben bündeln. So wird unter anderem die gesamte Logik für die Netzwerkkommunikation mit der RESTCONF Schnittstelle im RequestService verpackt. Nur von diesem Service aus wird eine Verbindung zum NSO aufge-

baut. Im gleichen Stil wird die Logik für die weiteren Funktionalitäten der Benutzeroberfläche gekapselt. Mehrfach verwendete, oder unabhängige Logik wird im CommonService gebündelt.

Die Verlagerung von Anwendungslogik in Services bringt somit Struktur in den Programmcode und verbessert dessen Wartbarkeit. Weniger Code muss unnötig dupliziert werden und die Controller in der Rails App bleiben schlank und spezifisch.

6.4.3 Jobs

Damit gewisse Anfragen an das NSO parallel ausgeführt werden können, wurden für diese einzelne Jobs erstellt. So kann ein Job mit unterschiedlichen Eigenschaften mehrfach gestartet werden und mehrere Requests an das NSO gesendet werden. Das NSO bearbeitet diese Anfragen zwar sequentiell, dennoch zeigt sich durch die Parallelisierung ein Zeitvorteil. Auch die Jobs kommunizieren mittels RequestService mit dem NSO. Dies wurde aus Übersichtsgründen jedoch nicht im obigen Diagramm eingetragen.

Ein Beispielszenario für die Verwendung dieser Jobs ist, wenn mehrere Geräte mittels Sync-From mit dem NSO synchronisiert werden sollen. Die Rails Applikation sendet pro Gerät, welches synchronisiert werden soll, eine Anfrage an das NSO. Das NSO synchronisiert jedes Gerät sequentiell. Durch die Parallelisierung muss jedoch nicht immer die Antwort vom NSO abgewartet und eine neue Anfrage gesendet werden. Stattdessen kann das NSO bereits die nächste, wartende Anfrage bearbeiten.

6.4.4 Interaktionen der Komponenten

Abbildung 31 zeigt die Interaktionen der einzelnen Komponenten im GUI bei der Erstellung einer Service Instanz. Treten bei der Erstellung Fehler auf, werden diese entsprechend an den Benutzer rückpropagiert.

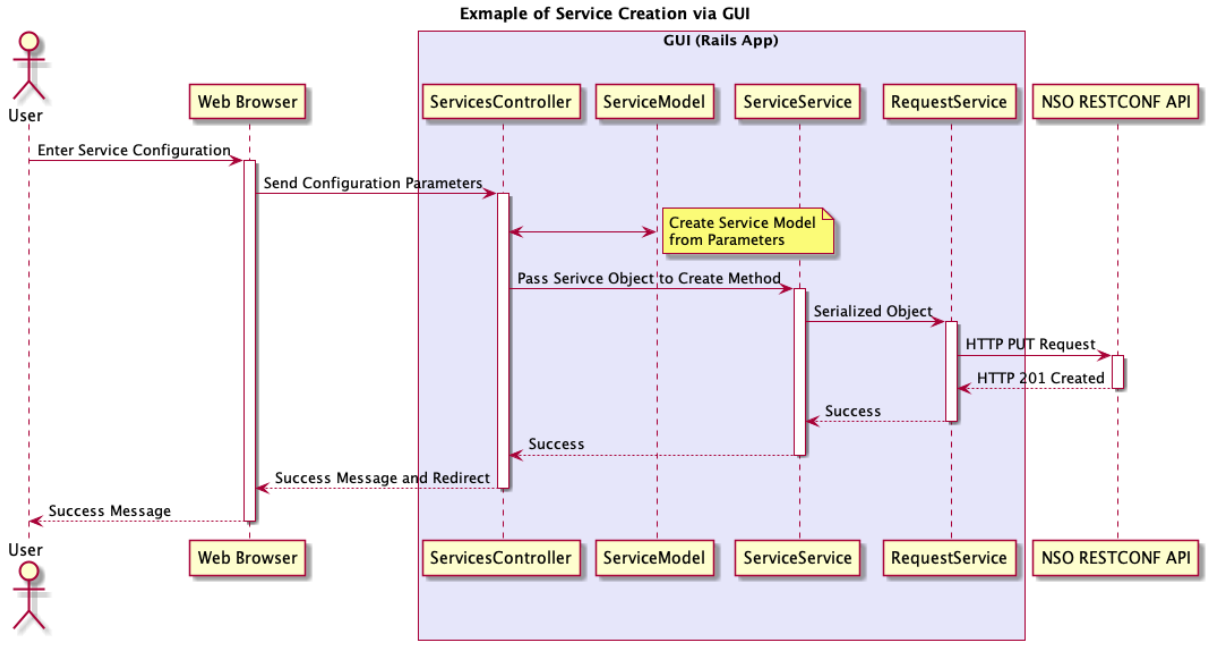

Abbildung 31: Sequenzdiagramm Erstellung eines Services

6.4.5 Verwendete Technologien

Bei der Umsetzung der GUI Komponente werden nebst Ruby on Rails weitere erwähnenswerte Technologien eingesetzt:

Bootstrap
Um die Entwicklung der einzelnen Views zu vereinfachen, verwenden wir Bootstrap. Bootstrap ist ein freies CSS-Framework, welches Gestaltungsvorlagen für Typografie, Formulare, Buttons, Grid-Systeme und viele weitere Benutzeroberflächenelemente mit optionalen JavaScript Erweiterungen zur Verfügung stellt.

Rest-Client
Für die Erstellung der HTTP Anfragen an das NSO verwenden wir eine Bibliothek namens rest-client für Ruby. Rest-client ermöglicht es uns, spezifische HTTP Requests mit entsprechenden Headern, HTTP Verben und SSL Optionen zu erstellen, welche einen reibungslosen Nachrichtenaustausch mit dem NSO ermöglichen.

Sucker Punch
Sucker Punch ist ein Ruby Gem für Rails, welches die Möglichkeit bietet, Jobs in parallel auszuführen, ohne dass diese persistent in einer Datenbank gespeichert werden müssen. Andere Bibliotheken verwenden für das Queue Management der gestarteten Jobs einen persistenten Datenspeicher wie Redis, oder PostgreSQL, welcher einen Schutz gegen verlorene Jobs bei einem unerwarteten Neustart, oder Crash der Rails App bietet. Wie bereits erwähnt, wollten wir jedoch der GUI Applikation keinen persistenten Speicher zusätzlich hinzufügen. Sucker Punch speichert die Queue der Jobs stattdessen in-Memory. Dafür können gestartete Jobs im schlimmsten Fall jedoch verloren gehen, bzw. nicht abgeschlossen werden. Für unser Verwendungsszenario ist dies jedoch kein gravierendes Problem.

6.4.6 Konfiguration der Rails Applikation

Die Rails Applikation kann über mehrere Konfigurationsswitches an die entsprechenden Bedürfnisse angepasst werden. Diese sind als Environment Variablen realisiert, welche beim Start der Anwendung aus einer Datei ausgelesen werden. Im `config` Verzeichnis der Rails App befindet sich die Datei `application.yml`, welche diese Variablen setzt:

```
NSO_LOCATION: 'http://localhost:8080'
NSO_USERNAME: 'user'
NSO_PASSWORD: 'pass'
NSO_NO_NETWORKING: 'false'
NSO_SYNC_FROM_AFTER_COMMIT: 'true'
NSO_SYNC_FROM_DELAY: '6'
NSO_CONNECT_TIMEOUT: '15'
LOG_NSO_REQUESTS: 'true'
RAILS_LOG_TO_STDOUT: 'true'
RAILS_SERVE_STATIC_FILES: 'true'
RAILS_MASTER_KEY: 'aabbccddeeff'
```

Auflistung 4: Beispielinhalt von ./config/application.yml

NSO_LOCATION
URL, welche verwendet wird, um die RESTCONF Schnittstelle des NSO zu kontaktieren

NSO_USERNAME
Benutzername, welcher für die Authentifizierung beim NSO verwendet wird

NSO_PASSWORD
Passwort, welches für die Authentifizierung beim NSO verwendet wird

NSO_NO_NETWORKING
Einstellung, ob die Commits im NSO mit dem Flag no-networking durchgeführt werden sollen, sodass keine Änderungen auf die Netzwerkgeräte geschrieben werden. Praktisch fürs Debugging.

NSO_SYNC_FROM_AFTER_COMMIT
Einstellung, ob nach einem Commit das NSO automatisch ein Sync-From von den betroffenen Netzwerkgeräten durchführt.

NSO_SYNC_FROM_DELAY
Verzögerung in Sekunden, welche vor einem automatischen Sync-From abgewartet wird.

NSO_CONNECT_TIMEOUT
Timeout in Sekunden, welches abgewartet wird, bevor eine Verbindung ohne Antwort mit dem NSO terminiert wird.

LOG_NSO_REQUESTS
Einstellung, ob Anfragen an das NSO im Log protokolliert werden sollen. Nützlich fürs Debugging, sollte aber im Produktionsbetrieb ausgeschaltet werden, da sensitive Informationen geloggt werden.

RAILS_LOG_TO_STDOUT
Einstellung, ob Loginformationen von Rails auf die Konsole (STDOUT) ausgegeben werden sollen. Andernfalls werden die Informationen in das entsprechende Logfile im `log` Verzeichnis geschrieben. Bei Verwendung der App in Kombination mit Systemd kann so direkt in das entsprechende Journal geloggt und mit `journalctl` eingesehen werden.

RAILS_SERVE_STATIC_FILES
Einstellung, ob die Asset (CSS und Javascript) Files direkt von Rails ausgeliefert werden sollen. Da wir bei unserem Setup keinen Reverse Proxy wie Nginx davor verwenden, welcher die Assets ausliefert, muss diese Einstellung aktiviert sein.

RAILS_MASTER_KEY
Master Key, welcher für die Entschlüsselung der Datei `config/credentials.yml.enc` verwendet wird. Dieser Eintrag ist in unserem Fall der Ersatz für die Datei `config/master.key`, welche dafür normalerweise verwendet wird.

6.5 Python Komponente

Die Python Komponente nimmt die Informationen des NSO Services entgegen und bereitet diese auf. Dazu gehören unter anderem die IP-Adressen, die vom Infoblox abgerufen werden. Für unseren Service haben wir zwei Komponenten erstellt: Eine Hauptkomponente namens MPLS_L3VPN und die Zweitkomponente CDB_Watcher. Die Klassen beider Komponenten wurden nochmals in Module gruppiert. Mit Ausnahme der Proxy-Klassen sind alle Module in einem eigenen File gebündelt. Die Proxy-Klassen wurden, der Übersicht halber, in einzelne Files aufgeteilt. Abbildung 32 zeigt das Dependency-Diagramm der Python-Klassen.

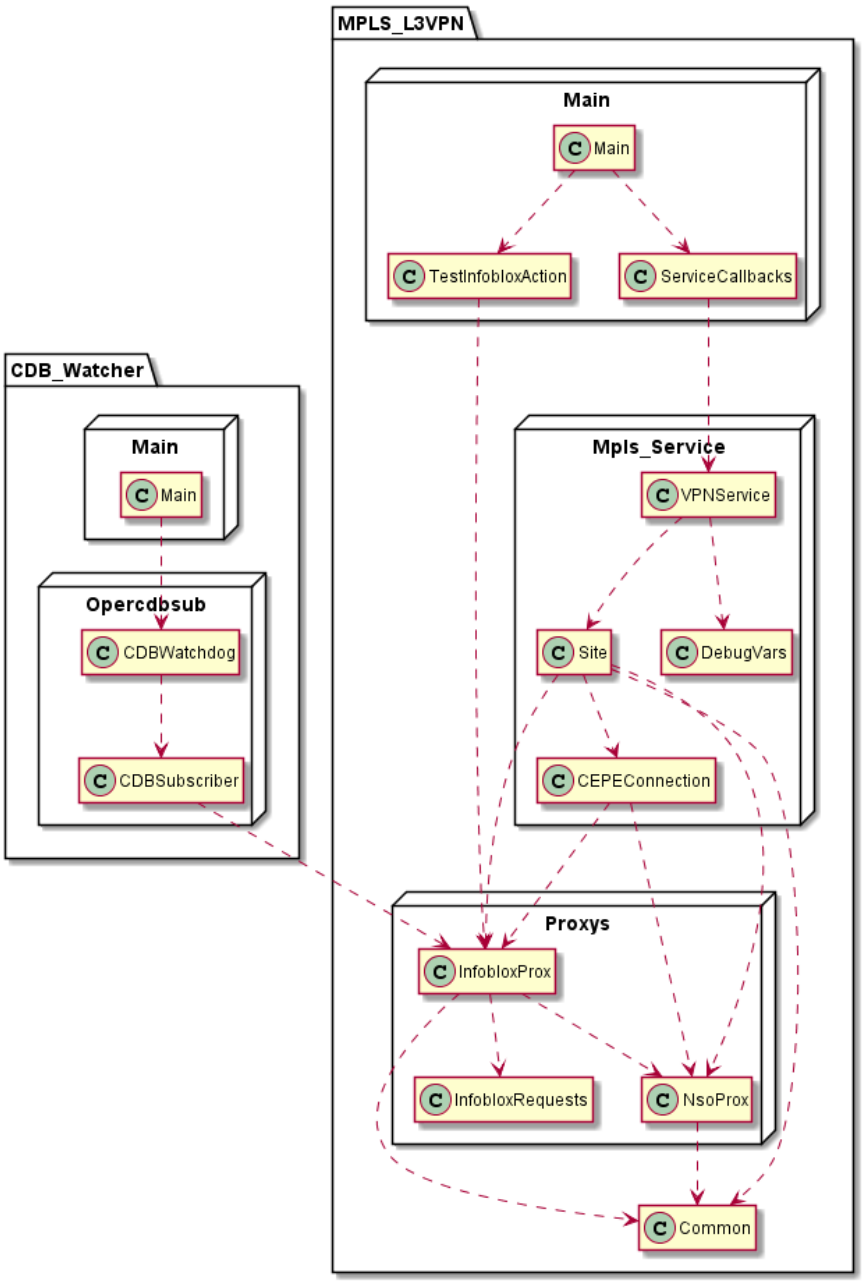

Abbildung 32: Python Dependency Diagramm

Die MPLS_L3VPN Komponente besitzt den Hauptteil der Servicelogik. Sie registriert die nötigen Callbacks für das NSO, registriert IP-Adressen und füllt die Service-Templates ab.

6.5.1 Modul: Main

Das Main Modul wurde vom NSO erstellt und ist der Einstiegspunkt des NSO Services.

Klasse: Main
Die Main Klasse ist für das Setup und Teardown des ganzen Services verantwortlich. Sobald das Package geladen wird, registriert sie die ServiceCallbacks und die TestInfobloxAction beim NSO und aktiviert so den Service.

Klasse: ServiceCallbacks
Wie schon im Name beschrieben, enthält diese Klasse die Callbacks, welche das NSO aufruft, sollte ein VPN erstellt, gelöscht oder geändert werden. Für unsere Arbeit ist aber nur das `create` Callback vonnöten.

Klasse: TestInfobloxAction
In dieser Klasse wird eine Action definiert, mit der die Verbindung zum Infoblox getestet werden kann. Mithilfe des InfobloxProx wird über ein HTTP-GET überprüft, ob das konfigurierte Infoblox die benötigte API-Version unterstützt.

6.5.2 Modul: Mpls_Service

Dieses Modul beinhaltet die Logik zum Abfüllen der Templates und zum Aggregieren der benötigten Daten.

Klasse: VPNService
Diese Klasse beinhaltet sämtliche VPN-spezifischen Daten (wie VPN-ID) und eine Sammlung von einzelnen Sites. Das Abfüllen der Templates wird hier gestartet.

Klasse: Site
Eine Site beinhaltet alle Informationen, die spezifisch für jede Site sind (wie externe Gateway-IP) und eine Liste von CEPEConnections. Informationen, welche nicht von dem VPNService zur Verfügung gestellt werden, holt sie die Site aus externen Quellen, entweder mithilfe des InfobloxProx oder des NsoProx.

Klasse: CEPEConnection
Sämtliche Daten, welche spezifisch zu einem CE sind, werden in dieser Klasse gespeichert. Hierzu gehören der entsprechende PE, die Transit-Adressen und eventuelle HSRP Konfiguration. Zusätzliche Informationen werden ebenfalls von den beiden Proxys beschafft.

Klasse: DebugVars
Da die XML-Templates nur sehr dürftige Fehlermeldungen über die Variablen, die sie bekommen, liefern, gibt es diese Wrapper-Klasse. Sie loggt jede Variable, die abgefüllt wird, und erleichtert so die Fehlersuche.

6.5.3 Modul: Proxys

Die Proxys maskieren Zugriffe auf externe Ressourcen und speichern gegebenenfalls Daten.

Klasse: InfobloxProx
Der InfobloxProx ist für Abfragen an das Infoblox zuständig. Er kümmert sich auch darum, dass IP Adressen nicht mehrmals reserviert werden und dass unbenutzte Adressen wieder freigegeben werden. Er speichert die aktuell reservierten IP-Adressen als Oper-Data in der CDB.

Klasse: NsoProx
Dieser Proxy beinhaltet sämtliche Methoden, die Informationen aus dem NSO lesen. Dazu verwendet er die MAAPI- und die CDB-API.

Klasse: InflobloxRequests
Um den InfobloxProx nicht mit Methoden zu überfüllen, wurden sämtliche Methoden, die HTTP-Requests starten oder Wrapper dafür sind, in diese Klasse ausgelagert.

Klasse: Common
In dieser Klasse sind einige Konfigurationsvariablen gebündelt, die über mehrere Klassen verwendet werden.

6.6 Komponente: CDB_Watcher

Diese Nebenkomponente kümmert sich um das Aufräumen des InfobloxProx. Dazu subscribed sie auf ein bestimmtes Feld in der CDB und löscht bei Änderungen überflüssige IP-Adressen aus dem Infoblox. Die genaue Vorgehensweise ist im Kapitel 6.7 beschrieben.

Klasse: Main
Genau wie in der Hauptkomponente ist diese Klasse autogeneriert. Nach dem Laden des Packages instanziiert sie den CDBWatchdog und startet ihn. Da der CDBSubscriber mit einem blockierendem Loop implementiert ist, wird er in einem separaten Thread gestartet.

Klasse: CDBWatchdog
Der CDBWatchdog kümmert sich um das Setup für den CDBSubscriber. Er bereitet die nötigen Parameter vor und registriert die Callbacks für das Beenden des Subscribers beim Shutdown des NSO.

Klasse: CDBSubscriber
Im CDBSubscriber ist die Subscriber-Logik der CDB implementiert. Bei jeder Notification startet er die Cleanup-Methode im InfobloxProx.

6.7 Reactive Fastmap Alternative

Da unser NSO Service IP-Netzwerke aus dem Infoblox reserviert, wäre das Reactive FASTMAP Pattern geeignet. Leider fehlt uns für die Umsetzung die benötigte Dokumentation. Die DP-API, welche für Reactive FASTMAP benötigt wird um das Redeploy auszulösen, ist in Python zwar vorhanden, aber nicht ausreichend dokumentiert. Deswegen haben wir das Pattern so abgewandelt, dass das Redeploy nicht mehr nötig ist. Dadurch wird auch das Deploy des Services in einem Durchgang wieder möglich und dem Benutzer kann eine klare Fehlermeldung geliefert werden. Im Gegenzug dauert das Erstellen eines Services ein paar Momente länger.

6.7.1 InfobloxProx (Ressource-Manager)

Die Rolle des Ressource-Manager wird durch den InfobloxProx übernommen. Er bekommt Anfragen für IP-Netzwerke und bearbeitet diese. Er führt eine Liste von sämtlichen Netzwerken und deren ID, nachfolgend State genannt. Durch den State ist sichergestellt, dass eine Service Instanz bei jeder Ausführung der `create` Methode die gleichen Netzwerke vom Proxy bekommt. Zusätzlich ermöglicht der State die Löschung von nicht mehr benötigten Netzwerken aus dem Infoblox.

Identifikation der Anfragen
Der Proxy kann Anfragen für Transit-Netzwerke oder für Externe-Netzwerke von verschiedenen Service Instanzen bekommen. Damit der Proxy weiss, welcher Service die Anfragen gestellt hat, speichert dieser zusätzlich zu den Netzwerkinformationen auch noch eine ID im State. Diese ID ist abhängig vom Namen der Service Instanz und der entsprechenden Site. Da Namen für Sites und Instanzen immer einzigartig sein müssen, ist die Uniqueness der ID so sichergestellt.

Reservations-Tokens
Um zu wissen, ob ein Netzwerk noch verwendet wird, werden Reservations-Tokens benutzt. Diese werden von der Service Instanz innerhalb ihrer `create` Methode erstellt. Dadurch sind sie im FASTMAP Prozess mitinbegriffen und werden automatisch entfernt, sobald der Service sie nicht mehr benötigt. Der Proxy kann durch einen Vergleich des States mit der Token-Liste herausfinden, welche Netzwerke gelöscht werden müssen.

Persistenz des States
Der State des InfobloxProx muss persistent sein und auch ein Neustart des NSO überleben, anderenfalls würden im Infoblox Reste von alten Services übrig bleiben. Da die ganze Logik des Proxys innerhalb der `create` Methode ausgeführt wird, kann nicht direkt in die CDB geschrieben werden. Es wird zwar eine MAAGIC-Session innerhalb des `create` bereitgestellt, allerdings unterliegt diese den Regeln von FASTMAP. Dies heisst, falls die Service Instanz gelöscht oder geändert wird, werden sämtliche Änderungen dieser MAAGIC-Session automatisch rückgängig gemacht. Die Session besitzt natürlich auch die Write-Locks der CDB, weshalb die Öffnung einer zweiten Write-Session in der `create` Methode zu einem Deadlock führt.

Die Alternative ist die Nutzung der low-level CDB-API, die direkt per TCP in die CDB schreiben kann. So ist es ohne Notwendigkeit eines Commits möglich, persistent Daten zu speichern. Leider ist auch hier die Dokumentation wieder sehr spärlich und die Fehlermeldungen wenig hilfreich. Deshalb mussten wir uns damit zufriedengeben einfache String-Values zu schreiben und zu lesen und konnten die Daten nicht in Listen organisieren. Sämtliche Daten des States werden als persistente Oper-Data gespeichert.

6.7.2 CDBWatcher

Der InfobloxProx ist in der Lage die Mehrfachreservation von Netzwerken zu unterbinden. Gemäss FASTMAP muss ein NSO Service sich nur um das `create` Szenario kümmern, und bekommt deshalb nicht mit, falls ein Service gelöscht oder geändert wird. Um nicht mehr benötigte Netzwerke von gelöschten oder geänderten Service Instanzen auch aus dem Infoblox wieder zu entfernen, muss er deshalb von extern benachrichtigt werden. Für diese Aufgabe wurde der CDBWatcher als zusätzliche Service-Komponente entwickelt.

Der CDBWatcher subscribed auf die Liste mit den Reservation-Tokens. Die CDB-API bietet eine Funktion, die blockiert bis eine Änderung am Subscriber-Objekt auftritt. Damit diese Funktion nicht die ganze

Komponente blockiert, wird sie in einem eigenen Thread ausgeführt. Bei einer Änderung der Tokenliste wird der InfobloxProx benachrichtigt, und dieser kann so nicht mehr benötigte Netzwerke beim Infoblox wieder entfernen.

6.8 NSO

Die Abbildung 33 zeigt, welche Daten wie im NSO angelegt werden. Die selbstdefinierten Daten sind in drei Kategorien aufgeteilt: Globale Konfiguration (Grün), Oper-Data (Grau) und Service Instanz spezifische Daten (Rot). Die blaue Klasse wurde vom NSO erstellt und ist nur vorhanden um die Abhängigkeiten zu veranschaulichen.

6.8.1 Globale Konfiguration

Die globale Konfiguration speichert Daten die sich auf die Grundkonfiguration des Systems berufen. Sie werden benötigt, damit bei der Erstellung eines Services nur minimale Informationen angegeben werden müssen.

6.8.2 Oper-Data

In den Oper-Data Klassen werden State-Informationen für den InfobloxProx gespeichert. Dies sind read-only Daten, die nur vom NSO Service selber verändert werden können.

6.8.3 Instanzdaten

Diese Klassen werden vom NSO verwendet um einen instanziierten Service (als Beispiel ein VPN zwischen zwei Sites) zu speichern. Alle Informationen, die zwar zum Erstellen des VPN benötigt werden, aber nicht in den Instanz-Klassen vorhanden sind werden vom Service aus den anderen beiden Klassentypen, oder sogar aus externen Quellen wie dem Infoblox extrahiert.

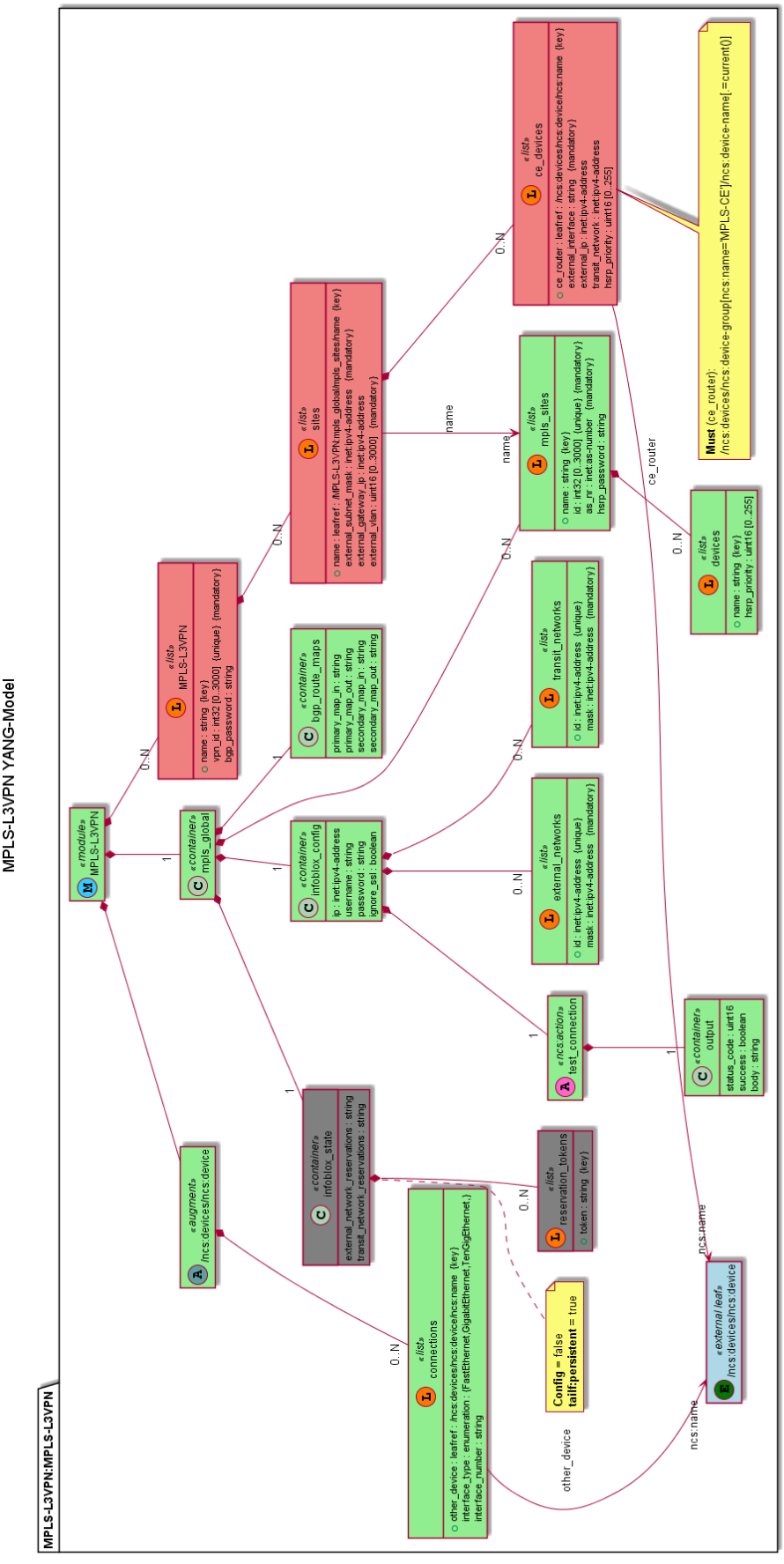

Abbildung 33: Klassendiagramm NSO

6.9 IP-Ranges

Damit IP-Adressen automatisch aus dem Infoblox abgerufen werden können, muss der NSO Service wissen, aus welchen Ranges er Adressen anfragen kann. Diese Netzwerke können in der Infoblox_Config in den beiden Listen external_networks und transit_networks eingetragen werden.

Zusätzliche Informationen im Infoblox
Zu jedem Netzwerk, welches reserviert wird, werden noch zusätzlich Informationen im Infoblox gespeichert. Die VLAN-ID wird als extra Attribut im Feld `extattrs` gespeichert. Zusätzlich wird im Kommentar beschrieben, für welchen VPN die IP reserviert wurde.

6.9.1 Externe Netze

Externe Subnetze sind die Netze, die von dem VPN schlussendlich verbunden werden. Der CE bietet dieses Netzwerk auf seinem Externen Interface per VLAN an. Je nach VPN-Konfiguration können sie unterschiedliche Subnetzgrössen haben. Der Benutzer gibt die gewünschte Grösse ein, und der NSO Service fragt beim Infoblox eine entsprechend grosses Subnetz an. Der CE bekommt anschliessend die erste verfügbare IP aus diesem Subnetz für das externe Interface zugewiesen. Falls mehrere CE-Router für eine Site konfiguriert wurden, wird die erste Adresse für HSRP verwendet und die CE-Router bekommen die aufsteigend die nächsten Adressen zugewiesen.

6.9.2 Transit Netze

Transit Subnetze stellen die Kommunikation zwischen CE und PE sicher. Sie besitzen immer eine Grösse von /30. Der CE bekommt immer die kleinere, ungerade IP und der PE die grössere, gerade. Bei einem Netzwerk von 192.168.0.0/30 wäre das 192.168.0.1 für den CE und 192.168.0.2 für den PE.

6.9.3 VPN-IDs

Sämtliche Adressen werden stets mit einem VLAN verwendet. Diese VLAN-ID wird nicht vom Infoblox abgefragt, sondern vom NSO Service selbst definiert. Bei jeder reservierten IP wird allerdings noch die ID des verwendeten VLANs mitgespeichert.

7 Resultat

Folgender Abschnitt setzt sich mit dem erreichten Resultat auseinander und vergleicht dieses mit den anfänglich gestellten Anforderungen.

7.1 Abdeckung der Use Cases

Im Verlauf dieser Arbeit konnten leider nicht alle der gewünschten Use Cases umgesetzt werden. Unerwartete Probleme sorgten für Zeitverzögerungen, sodass verschiedene Anforderungen gestrichen werden mussten.

7.1.1 Use Case 1

Der primäre Use Case mit den fundamentalen Anforderungen wurde fast komplett umgesetzt. Der Support für Juniper Geräte wurde bereits nach zwei Drittel der Projektzeit in Absprache mit dem Betreuer gestrichen, damit der Fokus auf Cisco Geräte gerichtet werden kann. Ein XR Router stand relativ spät erst als Testgerät zur Verfügung. Bei dessen Verwendung mit dem NSO sind schwerwiegende Probleme aufgetreten. Für die Lösung dieser Probleme bestand keine Zeit mehr, sodass der Support von XR Router schlussendlich auch gestrichen wurde. Somit unterstützt unser Produkt im momentanen Auslieferungszustand nur Cisco XE Router.

Die Konfigurationsmöglichkeit für QoS Policies wurde niedrig priorisiert und musste schlussendlich aus Zeitgründen ausgelassen werden. Eine Implementation dieser Funktionalität sollte aber nicht allzu aufwändig ausfallen.

Use Case 1.1 - 1.4
Die Sub Use Cases 1-4 vom ersten Use Case sind alle komplett umgesetzt.

7.1.2 Use Case 2

Auf der grafischen Oberfläche ist einsehbar, auf welchen Sites ein VPN deployed ist. Das Gegenstück dazu, welche VPNs bei einer Site deployed sind, ist ebenfalls einsehbar. Die Benutzerführung wurde jedoch ein wenig angepasst und weicht von der exakten Aufgabenstellung ab. Dennoch ist dieser Use Case komplett umgesetzt worden.

7.1.3 Use Case 3

Mittels der grafischen Oberfläche kann der Synchronisationsstatus der Geräte eingesehen werden. Sync-To oder Sync-From Aktionen können auf den Geräten ausgeführt werden. Unterschiede in der Konfiguration von Gerät und NSO können in einer Diff-Darstellung eingesehen werden. Dementsprechend ist dieser Use Case auch komplett umgesetzt.

7.1.4 Use Case 4 bis 6

Diese drei Use Cases sind alle aufgrund von mangelnder Zeit komplett ausgelassen worden. Bereits von Beginn der Arbeit an war klar, dass diese mit grosser Wahrscheinlichkeit nicht umgesetzt werden. Jeder einzelne Use Case hätte dazu geführt, dass wir uns in neue Technologie einarbeiten müssten, was zusätzliche Zeit in Anspruch genommen hätte.

7.1.5 Use Case 7

Der letzte Use Case wurde nach Absprache mit dem Betreuer partiell umgesetzt. Neue Geräte müssen von Hand im NSO eingetragen und einer Device-Group hinzugefügt werden. Alle fortlaufenden Konfigurationsschritte sind aber in unserer grafischen Oberfläche implementiert und können mittels dieser getätigt werden.

7.2 Nicht funktionale Anforderungen

Zu Beginn dieser Arbeit wurden nicht funktionale Anforderungen definiert. Nun wird überprüft, wie vollständig diese eingehalten wurden. Diese Anforderungen wurden sehr stark durch das Verhalten vom NSO beeinflusst.

7.2.1 Sicherheit

Jegliche Kommunikationskanäle, welche verwendet werden, sind verschlüsselt, um das unerlaubte Abhören von Informationen zu unterbinden. Eine Art von Benutzerauthentifizierung, um das System von unautorisierten Personen zu schützen, wurde nicht implementiert, da dies durch die Aufgabenstellung nicht vorgesehen war. Schlussendlich sind wir der Meinung, dass diese Anforderung gut erfüllt wurde.

7.2.2 Skalierbarkeit

Durch die Workarounds für Probleme, welche aufgetreten sind, wurde die Skalierbarkeit der gesamten Architektur stark negativ beeinflusst. Dennoch wurde darauf geachtet, dass die Skalierbarkeit nicht unnötig beeinträchtigt wurde. Bei der Entwicklung der GUI Komponente wurde Wert darauf gelegt, dass keine N+1 Queries implementiert werden. Im Entwicklungsumfeld dieser Arbeit wurden bis zu sechs Geräte angesteuert. Dies hat den Umständen entsprechend gut funktioniert. Steigt die Anzahl der bedienten Geräte stark an, ist aber nicht garantiert, dass sich das System einwandfrei verhalten wird. Dadurch ist die Anforderung an die Skalierbarkeit nur teilweise erfüllt.

7.2.3 Bedienbarkeit

Bei der Erstellung der Benutzeroberfläche wurde darauf geachtet, diese so simpel wie möglich zu gestalten. Insbesondere in Kombination mit der Bedienungsanleitung sollte demnach die Oberfläche einfach zu bedienen sein und längere Ladezeiten werden entsprechend dargestellt. Treten unerwartete Probleme im NSO oder auf den Netzwerkgeräten auf, müssen diese aber von Hand untersucht und behoben werden. Dieser Punkt ist dadurch unserer Meinung nach grösstenteils erfüllt.

7.2.4 Performance und Stabilität

Dieser Punkt deckt sich teilweise mit dem der Skalierbarkeit und Bedienbarkeit. Die aufgetauchten Probleme wurden mittels Workarounds behoben, mit Kosten auf die Performance und Stabilität. Zudem ist die momentane Lösung stark an die Testinfrastruktur mit den entsprechenden Firmware Versionen der Router und die verwendete NSO Version gebunden. Die angestrebte Reaktionszeit der grafischen Oberfläche von drei Sekunden wurde grundsätzlich erfüllt, hängt jedoch vollkommen von der Reaktionszeit vom NSO ab. Bestimmte Aktionen dauern NSO-bedingt im Schnitt bis zu 20 Sekunden oder mehr. Daraus ist unserer Meinung nach dieser Punkt nur mangelhaft erfüllt.

7.3 Ausblick

Ein offensichtlich nächster Schritt wäre die Implementation der momentan fehlenden Funktionalität (QoS, FlexAlgo, L2 Pseudowire, Health Checks). Der Support für XR und Juniper Router kann hier ebenfalls dazugezählt werden.

Momentan wird für die Verwendung des NSO Service eine Grundkonfiguration auf den Geräten vorausgesetzt, welche gewisse Voraussetzungen erfüllen muss. Genauere Details dazu sind der Installationsanleitung zu entnehmen. Diese Grundkonfiguration könnte bis auf die Anbindung ans NSO ebenfalls als Service in entwickelt werden, um das Gesamtsetup zu erleichtern.

Ein Austausch der NETCONF NEDs mit CLI NEDs wäre ebenfalls ein möglicher Schritt, um die Performance und Stabilität des Produkts zu steigern.

Des Weiteren ist in der Zwischenzeit eine neue Hauptversion vom NSO erschienen. Wir arbeiteten mit der Version 4.7. Zum momentanen Zeitpunkt (7. Juni 2019) ist die Version 5.1 zum Download verfügbar. Eventuell werden durch die neue Version auch Probleme behoben, auf welche wir gestossen sind. Ein weiterer Schritt wäre eine Aktualisierung auf die neuste Version.

8 Fazit

8.1 Technische Probleme

Im folgenden Abschnitt werden die grössten technischen Probleme genauer erläutert, welche uns im Verlauf dieser Arbeit begegnet sind. Die meisten Probleme konnten entweder minimiert, oder komplett behoben werden. Für gewisse Probleme mussten spezifische Workarounds implementiert werden, wodurch zum Teil auch eine verminderte Performance in Kauf genommen wurde.

8.1.1 Performance

Während der Bachelorarbeit traten Performanceprobleme auf, welche sich unter anderem durch lange Ladezeiten bemerkbar gemacht haben. Meistens konnten diese jedoch durch entsprechende Massnahmen vermindert werden oder spielten keine grosse Rolle.

NSO GUI
Nach der frischen Installation des NSO konnte dessen grafische Oberfläche ohne starke Verzögerungen und mit flüssigem Workflow verwendet werden. Sobald wir jedoch unseren NED installierten, wirkte sich das negativ auf die Performance aus. Zu einem gewissen Grad ist dies auch berechtigt, da durch den NED die interne Datenstruktur des NSO dezent komplexer wird und entsprechend mehr Ressourcen für dessen Verwaltung benötigt wird. Das Hauptproblem zeigte sich jedoch darin, dass die Konfigurationsseite für ein Gerät nicht mehr korrekt geladen wurde. Entweder wurde für länger als 10 Minuten eine Ladeanimation dargestellt, oder die Seite fälschlicherweise leer angezeigt. Einen Fehler bei der Installation des NSO, Kompilation des NEDs oder ungenügende Systemressourcen konnten ausgeschlossen werden, da das Problem bei allen Installation des NSO zeigte und mit dem Experte von Cisco abgeklärt wurde. Auch zwischen lokaler und globaler Installation konnte kein Unterschied festgestellt werden.

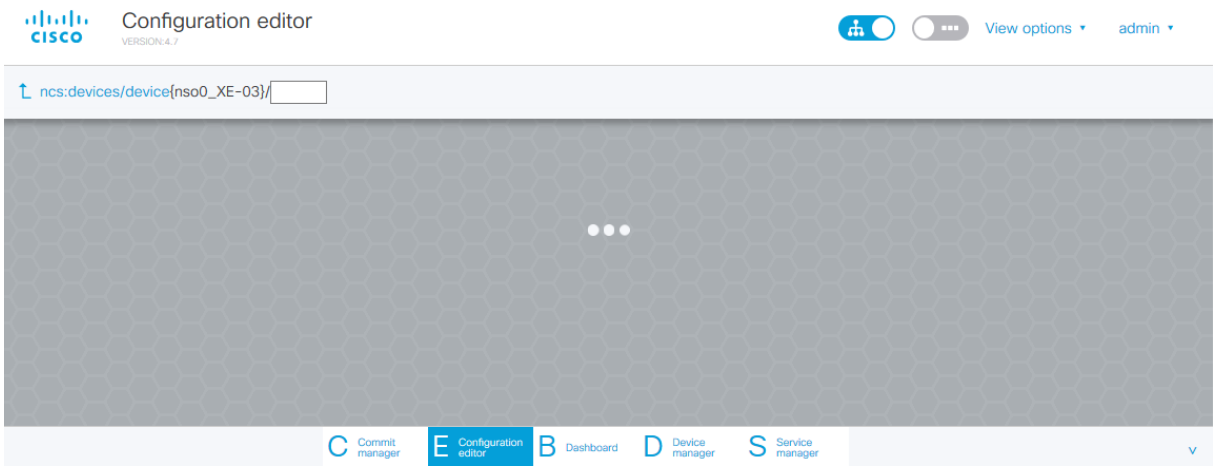

Abbildung 34: Festgefahrene Ladeanimation des NSO

Da wir für die Arbeit jedoch nicht auf das NSO GUI angewiesen sind und sich dieses Problem bei der Verwendung der CLI nicht zeigte, konnte dieses Problem ignoriert werden. Eine vermutete Lösung für dieses Problem könnte sein, bei der Kompilation des NEDs für die Netzwerkkomponenten, statt allen YANG Models des Geräts nur die benötigten Models einzubinden. Anfangs waren uns die konkreten Models jedoch noch nicht bekannt und zusätzlich erkennt das NSO dadurch nur noch Konfigurationsänderungen von Models, welche im NED vorhanden sind.

RESTCONF Antwortzeiten

Bei der Evaluation der Schnittstelle, welche von der Rails App für die Kommunikation mit dem NSO verwendet wird, sind uns bei der Verwendung von RESTCONF ebenfalls lange Antwortzeiten aufgefallen. Dabei muss jedoch erwähnt werden, dass dies nur bei Anfragen der Fall war, welche viele geschachtelte Daten abgefragt haben. Mittels zusätzlichen Query-Parametern kann die Anfrage eingeschränkt werden. Zum Beispiel werden nur gewisse Felder angefragt, oder die Tiefe der Datenschachtelung auf zwei Ebenen limitiert. Diese Justierungen wirken sich wieder sehr positiv auf die Antwortzeiten aus.

Die Anfragen, welche schlussendlich die GUI Komponente an das NSO sendet, werden so vom NSO meist innerhalb von wenigen 100 Millisekunden und im Worst-Case innerhalb von ca. zehn Sekunden beantwortet. Dies ist kein perfektes Resultat, aber innerhalb den von uns definierten nichtfunktionalen Anforderungen.

8.1.2 NSO und NETCONF

Bei der Verwendung von NETCONF mit NSO hatten wir ebenfalls mit einigen Problemen zu kämpfen, welche mehr oder weniger einfach zu lösen waren.

Kompilation des NETCONF NED

Als einer der ersten Schritte für die Inbetriebnahme von NSO muss ein NETCONF NED für die Geräte erstellt und installiert werden. Wie bereits erwähnt werden bei diesem Prozess mittels Pioneer die YANG Models von einem Gerät heruntergeladen und zu einem NED kompiliert, welcher anschliessend vom NSO als Package geladen werden kann. Bei diesem Prozess wurden wir zu unterschiedlichen Zeiten im Projekt mit Problemen konfrontiert:

- Das Herunterladen der YANG Models mittels Pioneer funktioniert nur mit einer globalen NSO Installation.
- Das YANG Model `tailf-confd-monitoring@2013-06-14.yang` enthielt fehlende Referenzen und musste entsprechend entfernt werden. (Abbildung 35)
- Openconfig YANG Models wurden alle entfernt, da sie zum Teil ebenfalls Fehler enthielten und nicht verwendet wurden.
- Der kompilierte NED für Cisco XR Router konnte im NSO nicht geladen werden. Es wurde jeweils nur der Fehler `Error: Internal error: internal` angezeigt. Mittels Logfiles konnte ein fehlerhaftes YANG Model ausgemacht werden. Wurde dieses entfernt, konnte der NED dann geladen werden.
- Mit dem XR NED traten beim Pushen von Einstellungen Fehler auf. Entweder wurden die Änderungen mittels generischer Fehlermeldung abgewiesen, oder vom Gerät nicht übernommen.
- Die NEDs der XE und XR Geräte verwenden zum Teil die gleichen Models, welche zu Konflikten beim Laden der NEDs führte.

```
augmented/Cisco-IOS-XE-ospf@2018-06-28.yang:7975: warning: The 'must' expression should have a tailf:dependency. If is doesn't, it will be checked for every commit.
augmented/tailf-confd-monitoring@2013-06-14.yang:65: error: grouping 'live_ncs_common-monitoring-objects' in module 'tailf-common-monitoring' not found
Makefile:25: recipe for target 'ncsc-out/.done' failed
make: *** [ncsc-out/.done] Error 1
make: Verzeichnis „/mnt/c/dev/ncs/packages/TESTNED/src“ wird verlassen
```

Abbildung 35: Fehler beim Builden eines NEDs

Da wir erst spät im Projekt Zugriff auf die XR Router bekommen haben, sind die Probleme mit dem XR NED auch spät erkannt worden. Diese Probleme hätten mehr Zeit in Anspruch genommen und daher wurde aus Zeitgründen der Support für XR Geräte nicht umgesetzt.

Gleiche Config in mehreren YANG Models

Durch die Struktur der YANG Models kann es vorkommen, dass eine Konfiguration mit mehreren Models angewendet werden kann. Dies ist zum Beispiel bei der Konfiguration von Interfaces der Fall. Hierfür wird das YANG Model von IOS Native, sowie das Model IF-Interfaces der IETF verwendet. Im Template unserer Konfiguration wird nur das IOS Native Model verwendet. Wird dieses Template nun angewendet, um Änderungen auf das Gerät zu schreiben, funktioniert dies auch einwandfrei. Wenn aber vom Gerät anschliessend die Konfiguration ausgelesen wird, werden beide Models für dessen Beschreibung verwendet. Dies führt dazu, dass das Gerät im NSO out of sync ist, da nur die Konfiguration eines Models bekannt ist, während das andere leer bleibt. Weitere Commits werden erst wieder zugelassen, wenn das Gerät in sync ist.

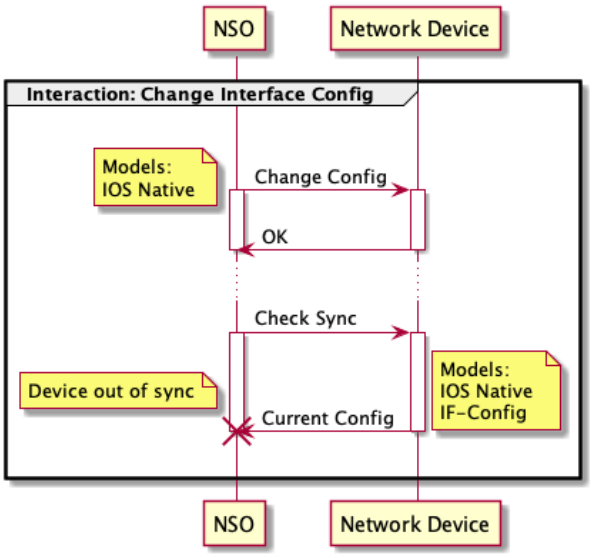

Abbildung 36: Check-sync mit mehreren Models

Ein weiteres Symptom dieses Problems ist, dass es unter Umständen die Sync-To Aktion fehlschlagen lässt. Die ist ist der Fall, wenn auf einem Gerät ein unerwünschtes Subinterface erstellt wird und NSO dieses durch Sync-To wieder löschen will. Die Interfaces werden vom NSO mit beiden Models gelöscht. Dieser Vorgang schlägt jedoch bei der Netzwerkkomponente während der Verarbeitung der Änderungen im zweiten Model fehl, da das Sub-Interface bereits vom Ersten gelöscht wurde.

Um das Subinterface auf dem Gerät wieder zu entfernen, muss zunächst durch ein Sync-From die Konfiguration des Subinterfaces ins NSO geladen werden. Nun muss das Subinterface in einem, und nur in einem, Model gelöscht werden. Beim anschliessenden Commit pusht NSO nur das geänderte Model. Nach erfolgreichem Commit muss noch ein Sync-From ausgeführt werden um das Subinterface auch noch aus dem anderen Model zu löschen. Dieser letzte Schritt ist besonders wichtig, da ansonsten das zweite Model nicht in sync ist. Bei der nächsten Sync-To Aktion würde das Subinterface dann wieder erstellt.

Ein Lösungsansatz für dieses Problem wäre, einfach eines dieser Models aus dem NED zu entfernen. Da nur mit dem IOS Native Model alle gewünschten Änderungen vorgenommen werden können, kommt nur das zweite Model für eine Löschung in Frage. Dies ist jedoch durch weitere Abhängigkeiten von anderen YANG Models auf das Model IF-Interfaces nicht möglich.

Des Weiteren wurde versucht das Problem zu beheben indem explizit beide Models mit dem Template entsprechend abgefüllt wurden. Beim Erstellen von z.B. Sub-Interfaces funktioniert dies problemlos. Beim Löschen von Sub-Interfaces tritt jedoch wieder das Problem auf, dass das Subinterface zweimal gelöscht wird.

Der Lösungsansatz, welcher schlussendlich umgesetzt wurde, sieht so aus, dass nach jedem Commit mit Konfigurationsänderungen auf ein Netzwerkgerät anschliessend ein Sync-From von diesem durchgeführt wird. So bleiben die Geräte nach einem Commit in sync. Jedoch wird ein weiteres Problem durch diese Lösung hervorgerufen. Denn im Zeitraum zwischen einem Commit und dem darauffolgenden Sync-From könnten theoretisch Out-of-Band Änderungen am Gerät vorgenommen werden, welche dann in die NSO Datenbank „geschmuggelt“ werden. Dieser Zeitraum ist mit ein paar Sekunden jedoch

relativ klein.

Dies ist keine schöne Lösung, aber sie funktioniert. Gemäss Experte von Cisco besteht dieses Problem bei der Verwendung eines CLI-NEDs nicht. Ein Wechsel stand für uns jedoch ausser Frage, da diese CLI-NEDs kostenpflichtig erworben werden müssen und wir erst nach zwei Drittel der Arbeitszeit darauf hingewiesen wurden. Ursprünglich wurde uns vermittelt, dass diese CLI-NEDs veraltet sind und mittlerweile NETCONF verwendet werden soll.

Dieses Problem ist relativ bedenklich, da es sich der Grundidee vom NSO widersetzt. Obwohl sich die Konfiguration auf dem Netzwerkgerät in der Zwischenzeit nicht verändert, sind die Daten welche vom NSO geschrieben werden und anschliessend wieder ausgelesen werden, unterschiedlich. Um dieses Problem zu eliminieren und eine korrekte Funktionalität des NSO zu gewährleisten, müsste das NSO in der Lage sein, zu erkennen, dass eine Konfiguration in zwei verschiedenen YANG Models gleichzeitig vorhanden sein kann.

Verzögerung bis Geräte NETCONF Config übernommen haben

Dadurch, dass wir direkt nach einem Commit des NSO wieder ein Sync-From vom Netzwerkgerät durchführen, haben wir ein weiteres NETCONF relevantes Problem festgestellt. Die Geräte welche uns vom Institute for Networked Solutions (INS) zur Verfügung gestellt wurden, benötigen einige Sekunden, um eine NETCONF Konfigurationsänderung zu übernehmen. Das heisst, wenn nun zu schnell nach einem Commit ein Sync-From durchgeführt wird, ist das Gerät trotz dem Sync-From wenige Sekunden danach nicht mehr in sync mit dem NSO, dargestellt in Abbildung 37.

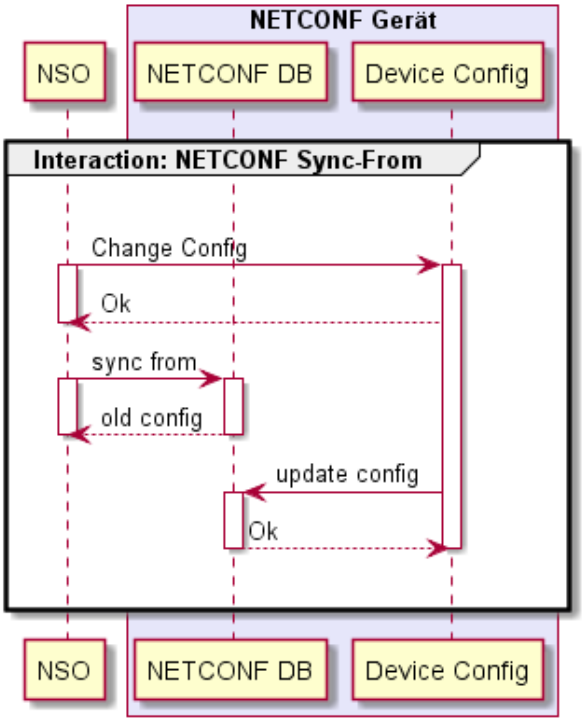

Abbildung 37: Sync-From mit alter Config

Um dieses Problem zu umgehen, wurde eine künstliche Verzögerung implementiert. Sprich es wird nun standartmässig sechs Sekunden nach einem Commit gewartet, bis die Sync-From Aktion für die betroffenen Geräte gestartet wird. Leider kann nicht festgestellt werden, wann die Geräte ihre Konfiguration intern synchronisiert haben und die Zeitverzögerung ist daher nur ein Schätzwert. Dieser Wert kann im `application.yml` File der Rails App angepasst werden, falls die Geräte länger als sechs Sekunden brauchen. Durch diese Verzögerung muss in der GUI Komponente eine entsprechend längere Ladezeit in Kauf genommen werden.

Eine Möglichkeit dieses Problem zu lösen wäre, wenn das NETCONF-Gerät bei einem Request solange blockiert, bis sämtliche Daten im Gerät intern sauber synchronisiert wurden. Dies würde zwar die Antwortzeiten von NETCONF negativ beeinflussen, aber immerhin wäre die Konsistenz gewährleistet.

Out-of-sync nach fehlgeschlagenem Commit

Schlägt ein Commit auf einem Netzwerkgerät fehl, ist das NSO fälschlicherweise der Meinung, das Gerät sei out-of-sync. Wird die Compare-Config Aktion auf dem Gerät aufgerufen, ist die Diff-Ausgabe

der Konfigurationsunterschiede leer und das Gerät plötzlich wieder in sync. Dieses Problem ist nicht unbedingt funktionsbeeinträchtigend, aber vermindert die Benutzerfreundlichkeit.

Leere Listen im NSO GUI
Manchmal hatten wir das Szenario, dass Listen (z.B. Inhalt der Device-Groups) in der NSO Benutzeroberfläche fälschlicherweise als leer dargestellt wurden, obwohl mit Elementen gefüllt waren. Mittels der CLI oder RESTCONF Abfragen wurden die Listen mit korrektem Inhalt dargestellt. Ein Neustart des NSO hat das Problem jeweils behoben.

8.1.3 Dokumentation

Eines der grössten Probleme, welches uns viel Zeit gekostet hat, war der Mangel an ausführlicher Dokumentation. Die mit dem NSO mitgelieferte Dokumentation ist erst nach dessen Installation einsehbar und zum Teil veraltet und fehlerhaft. Somit musste das NSO vorerst installiert werden, bevor die entsprechende Installationsdokumentation eingesehen werden konnte. In der Dokumentation wird oft von noch NCS anstelle von NSO gesprochen, was anfänglich für Verwirrung sorgte.

Über die Ciscowebsite, einem Community Forum von Cisco und in einzelnen GitHub Repositories, ist je nachdem Dokumentation mit Beispielen auffindbar. Die Beiträge im Forum sind teilweise sehr hilfreich gewesen. Manchmal fanden wir jedoch existierende Beiträge von anderen Personen mit der gleichen Frage, ohne dass diese beantwortet wurde, oder zu Dokumentation verlinkte welche nicht mehr existierte (Links auf Ciscowebsites mit Fehler 404). Generell ist die Doku meist eher spärlich oder wieder veraltet. Über das Cisco DevNet sind kurze Videokurse verfügbar, welche jedoch relativ kurz und eher marketingorientiert sind. Schlussendlich mussten viele Informationen aus verschiedenen Orten zusammengetragen werden, oder durch zeitaufwändiges Trial und Error ermittelt werden.

Developer Guide
Es existiert für das NSO einen Developer Guide, sprich ein Dokument, welches Entwicklerinformationen über verfügbare Schnittstellen und weiteres enthält. Für die Entwicklung eines NSO Services, ist dieses Dokument sehr praktisch. Leider ist dieses Dokument nicht öffentlich zugänglich und wir haben erst nach zwei Drittel der Arbeitszeit davon erfahren und darauf Zugriff bekommen. Zudem sind viele APIs, welche in Java verwendet werden können zwar dokumentiert, das Äquivalent in Python jedoch nicht.

Generische Fehlermeldungen
Während der Entwicklung des NSO Services sind immer wieder verschiedene Fehlermeldungen aufgetreten. Diese waren zum Teil sehr generisch, was das Debugging erschwerte. Zum Teil konnten erweitere Informationen aus zusätzlich aktivierten Logfiles extrahiert werden. Dennoch wäre es hilfreich wenn die Fehlermeldungen des NSO aussagekräftiger sind.

Beispiele der generischen Fehlermeldungen:

- `Error: Internal error: internal`
- `The built-in function returned NULL without setting an error`
- `inconsistent value: Device refused on or more commands`

Beispiele von NSO Services
Mittels Recherche im Internet und der beigefügen Dokumentation findet man Beispiele von NSO Services, welche für den Einstieg sehr praktisch sind. Viele dieser Beispiele waren auch in Python imple-

mentiert, was für uns sehr hilfreich war. Zum Teil waren diese jedoch leider veraltet und mit unserer NSO Version nicht mehr funktionsfähig.

8.1.4 RESTCONF

Überschreibung bei Erstellung von neuen Objekten
Wenn mittels der RESTCONF Schnittstelle ein neues Objekt (z.B. eine globale Site) im NSO erstellt wird, sendet die GUI Komponente einen HTTP PUT Request mit entsprechender Payload an das NSO. Wenn im NSO bereits ein solches Objekt mit gleichem Name besteht, wird dieses einfach überschrieben. Beim Ändern von bestehenden Objekten ist dies kein Problem, wird jedoch ein neues Objekt mit gleichem Name erstellt, kann so ein bestehendes komplett überschrieben werden. Nebst vereinzelten Forenbeiträgen mit Beispielen konnten wir keine Dokumentation finden, welche weitere Informationen über dieses Problem lieferte.

Um dieses Problem vorzubeugen wird in der Rails Applikation vor dem Erstellen eines neuen Objekt überprüft, ob ein gleichnamiges Objekt besteht. Ist dies der Fall, wird die Aktion abgebrochen und der Benutzer entsprechend informiert.

Umlaute in URL Parameter
Das NSO verwendet den Namen des Objekts als Parameter in der URL. Wenn nun zum Beispiel eine globale Site mit dem Namen Zürich verwendet wird, ist das Wort Zürich in der URL enthalten. Nicht-ASCII-Zeichen in URLs sind problematisch, können aber mit entsprechendem Encoding gelöst werden. Dies wird auch für die RESTCONF Kommunikation mit dem NSO verwendet. Aus uns unbekannten Gründen funktioniert dies jedoch mit Umlauten nicht:

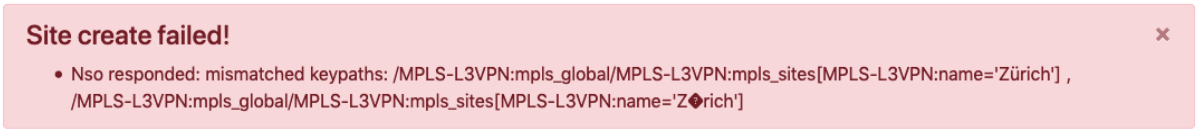

Abbildung 38: Fehler mit Umlauten in der URL

Wir haben das Problem so gelöst, dass nur alphanummerische Zeichen, Underscores und Bindestriche für Namen verwendet werden können.

8.2 Organisatorische Probleme

Aus organisatorischer Sicht sind ebenfalls kleinere Schwierigkeiten aufgetreten, welche kurz erläutert werden. Es sei zu beachten, dass dies lediglich Reflexionen aus Sicht der Teammitglieder sind. Sie sollen nicht als Schuldzuweisungen dienen, sondern bloss Punkte aufzeigen, welche wir als Teammitglieder in Zukunft verbessern können. Des Weiteren soll erwähnt werden, dass sich diese Probleme kaum negativ auf die Qualität der Arbeit ausgewirkt haben. Allenfalls ist dadurch lediglich ein wenig Zeit verloren gegangen.

8.2.1 Verfügbarkeit von Ressourcen

Im Verlauf dieser Arbeit haben wir zu verschiedenen Zeitpunkten unterschiedliche Arten von Ressourcen benötigt. Sei es die schriftliche Aufgabenstellung, eine Testinstanz von Infoblox, die virtuellen Router, oder die entsprechende Gerätekonfigurationen für die Router. Zum Teil mussten wir mehrere Tage

warten, oder unsere Arbeit anders priorisieren, bis die entsprechenden Ressourcen uns zur Verfügung standen.

Eine Möglichkeit um diese Problematik in der Zukunft zu vermeiden wäre, dass die Dringlichkeit solcher Ressourcen von unserer Seite klarer kommuniziert wird. Solche Deadlines müssen in der Projektplanung besser miteingeplant werden, sodass im Team selbst bekannt ist, ab wann welche Ressourcen fix benötigt werden.

8.2.2 Verfügbare Zeit von Betreuer, Experte und Industriepartner

Zum Teil konnten wir unsere Zeit nicht effektiv nutzen, da wir auf Antworten von Fragen warten mussten. Antworten auf unsere Mails oder Slack-Nachrichten dauerten manchmal mehrere Tage, oder blieben komplett aus. Manche Besprechungen konnten für entsprechende Fragen nicht genutzt werden, da die benötigte Ansprechperson nicht anwesend war, obwohl dies teilweise im Voraus entsprechend angekündigt wurde.

Dieses Hindernis könnte eventuell vermieden werden, wenn die Dringlichkeit dieser Fragen besser kommuniziert wird und entsprechende Deadlines für diese gesetzt werden. Wird eine Deadline erreicht, ohne dass die entsprechende Frage geklärt werden konnte, wird im Team intern eine Entscheidung gefällt und umgesetzt.

8.3 Fazit zum NSO

Das Grundkonzept vom NSO, die Verwaltung von mehreren Netzwerkgeräten über eine Single Source of Truth, ist für Netzwerkadministratoren sehr vielversprechend. Die Verwaltung von mehreren Netzwerkgeräten, ob vom selben Hersteller oder nicht, wird enorm erleichtert. Durch die Transaktionalität werden Konfigurationsüberreste vermieden und gleichzeitig auch eine Versions History angeboten.

8.3.1 Schnittstellen

Das Webinterface des NSO ist zwar praktisch, aber für den produktiven Einsatz nicht fehlerfrei genug. Dies war jedoch nicht unsere primäre Nutzungsweise des NSO. Es bietet mit RESTCONF, REST und JSON-RPC drei weitverbreitete northbound Schnittstellen, welche eine technologieunabhängige Steuerung des NSO ermöglichen. Die von uns verwendete RESTCONF Schnittstelle deckt den vollen Funktionsumfang des NSO ab und ist unkompliziert zu verwenden. RESTCONF bietet einen grossen Mehrwert für die Bedienung des NSO über selbstentwickelte Tools. Die erwähnten Probleme konnten mit Workarounds entschärft werden. Als einziger Nachteil könnte die langen Antwortzeiten genannt werden, die aber mehr der allgemeinen Performance des NSO zuzuweisen sind.

8.3.2 NETCONF NEDs

Die Verwendung von NEDs ermöglicht es dem NSO theoretisch beliebige Geräte von verschiedenen Herstellern zu unterstützen. Die Möglichkeit, mit Pioneer selbst NEDs für Geräte zu kompilieren, welche NETCONF unterstützen, ist ein grosser Vorteil und auch sehr einfach gehalten. Leider funktionieren NETCONF NEDs nicht so reibungslos wie sie sollten. Wenn sich die Probleme nur auf die Kompilierung und Installation beschränken würden, wäre unser Fazit bedeutend positiver. Vor allem die Komplikationen mit der Transaktionalität werden im alltäglichen Gebrauch zu Inkonsistenzen und viel unnötigem Mehraufwand führen. Aus diesen Gründen müssen wir von der Verwendung von NETCONF NEDs mit den getesteten Geräten im produktiven Umfeld abraten.

Als Alternative könnten die proprietären CLI-NEDs von Cisco dienen, diese wurden im Rahmen dieser Arbeit aber nicht getestet. Bei der Verwendung der Standard CLI NEDs, welche mit dem NSO mitgeliefert werden, konnten wir keine Probleme feststellen. Allerdings wurden diese mit einem kleineren Funktionsumfang und veralteten Geräten getestet.

8.3.3 Services

Die Services sind wohl das stärkste Argument für NSO. Zusammen mit der Transaktionalität bilden sie ein starkes Framework zur Verwaltung eines Netzwerkes. Durch FASTMAP wird Service Entwicklern viel Arbeit abgenommen, sodass sie sich auf die Konfiguration konzentrieren können und nicht allzu komplizierten Code schreiben müssen. Mit Python und Java stehen zwei weitverbreitete Sprachen zur Verfügung, auch wenn die Dokumentation von Python noch ausbaufähig ist. Ein grosser Vorteil von Python ist die Möglichkeit, einfach neue Packages zu installieren, dafür ist das Debugging, mit Ausnahme von Logfiles, leider nicht ohne weiteres möglich.

Durch Services wird die Konfiguration eines Netzwerkes immens vereinfacht, was auch die Bedienung durch weniger erfahrene Benutzer ermöglicht. Zudem wird durch Automatisierung die Wahrscheinlichkeit auf Konfigurationsfehler gemindert.

8.3.4 Schlussfolgerung

Cisco NSO ist ein Produkt, welches Initialaufwand benötigt, um voll ausgeschöpft zu werden. Zu Beginn müssen für die häufigsten Use Cases eigene Services entwickelt werden. Kleine Konfigurationsänderungen können natürlich auch ohne Service vorgenommen werden, dies ist aber nicht die Stärke des NSO. Sobald NSO mit verschiedenen Services bestückt wird, zeigt es sein volles Potential. Bei der Entwicklung dieser Services ist der FASTMAP Algorithmus sehr hilfreich, sofern man versteht wie man mit ihm umgehen muss.

Für unser Szenario können wir den Einsatz von NSO dennoch nicht empfehlen. Die Vorteile werden durch die Probleme mit NETCONF leider ausgehebelt. Vor allem die Verletzung der Transaktionalität, die ein Pushen der Konfiguration auf Geräte unter Umständen unmöglich macht, ist eine verheerende Einschränkung der NSO Funktionalität. Zudem bremst das teilweise Fehlen von Dokumentation die Entwicklung von Services stark ab, was durch stärkere Zusammenarbeit mit Cisco aber beseitigt werden könnte.

8.4 Projektfazit

In dieser Arbeit sollte Cisco NSO evaluiert und ein MPLS Service dafür erstellt werden. Es wurde viel Zeit benötigt für die Einarbeit in NSO und dessen Ökosystem, weswegen schlussendlich nicht alle gewünschten Funktionen umgesetzt werden konnten. Speziell die Implementierung des Reactive-FASTMAP benötigte einiges an Nachforschung. Dennoch sind wir mit dem Resultat zufrieden.

Die verwendete Architektur mit Service und Website, ist solide und besitzt auch potential für einen zukünftigen Ausbau. Durch die NETCONF Probleme mussten wir zwar an einigen Stellen etwas unschöne Workarounds einfügen, die sich auch auf die Architektur auswirkten. Doch trotz allen Problemen ist es uns gelungen die grundlegenden Funktionen, mit Performanceeinbussen, umzusetzen.

Der Projektplan wurde von Anfang an simpel gehalten. Insbesondere zu Beginn der Arbeit war vieles noch unklar und eine feingranulare Planung gar nicht möglich. Es wurden nicht viele Meilensteine oder fest definierte Ziele gesetzt. Dafür wurden, mit Einbussen einiger Features, die gesetzen auch erreicht.

Zudem ist bei einer solchen Arbeit, wo zu Beginn die benutzen Technologien und Frameworks noch nicht gut bekannt sind, eine genaue Abschätzung eine grosse Herausforderung. Wir sind stets davon ausgegangen, das wir für die optionalen Use Cases nicht genügend Zeit zur Verfügung haben werden.

Es ist dennoch bedauerlich, wieviel Zeit durch fehlende oder unvollständige Dokumentation verloren gegangen ist. Vor allem am Anfang sind wir dadurch eher langsam vorangekommen und mussten teils auch bereits erdachte Konzepte wieder verwerfen. Der NSO Developer Guide, der uns vom Experten zur Verfügung gestellt wurde, brachte uns schon ein grosses Stück weiter. Leider erhielten wir den erst in der elften Woche, also nach zwei Drittel des Projekts. Als wir dann ein gutes Verständnis für die NSO Serviceentwicklung erlangt hatten, stand der Feature-Freeze schon vor der Tür.

8.4.1 Zeitauswertung

Gesamtzeit	330h
Schlusspräsentation	15h
Anzahl Wochen	17h
Stunden pro Woche	19h

Für die BA sind pro Student 12 ECT Punkte an Zeitaufwand vorgesehen. Ein ECTS Punkt entspricht einem Arbeitsaufwand von 25-30 Stunden, dies ergibt für jeden Student 300 bis 360 Stunden. Für die Schlusspräsentation, inkl. Vorbereitung sind 15 Stunden pro Student geplant. Gemäss der Rechnung auf der Tabelle sind das 19 Stunden pro Woche und Student während dem Semester. Auf der Abbildung 39 als blaue Linie eingezeichnet.

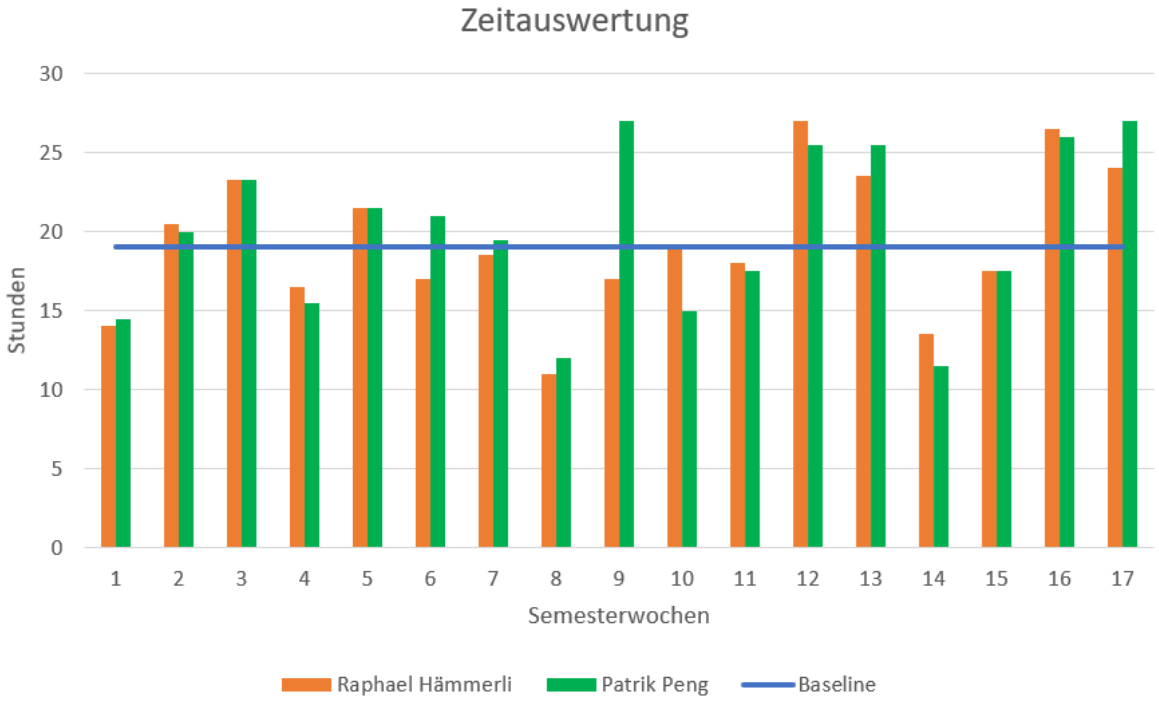

Abbildung 39: Auswertung Zeitdaten

In der Zeitauswertung sind die Oster- und Auffahrtswoche (8 und 14) gut zu erkennen. In der ersten Woche hatten wir noch nicht allzu viele Informationen, weswegen die Arbeitszeit in dieser Woche auch beschränkt war. Zum Abgabetermin der schriftlichen Arbeit wird Patrik Peng insgesamt 339 und Raphael Hämmerli 328 Stunden für diese Arbeit aufgewendet haben. Mit den 15 Stunden für die Präsentation währen das 354 und 342. Somit befinden sich beide Teammitglieder im Zeitrahmen von 300-360 Stunden.

8.4.2 Umgang mit Risiken

Sämtliche Risiken, die im Projektplan genannt wurden, werden in diesem Abschnitt nochmals beleuchtet. Es wird angeschaut ob, und in welchem Mass sie eingetroffen sind und ob bessere Massnahmen ergriffen hätten werden können.

R1 Ausfall durch Krankheit
Keines der Teammitglieder ist krank geworden, somit ist dieses Risiko nicht eingetroffen.

R2 Unklarheiten oder Verzögerungen der Aufgabenstellung
Dieses Risiko ist teilweise eingetroffen. Die vollständige Aufgabenstellung war nicht direkt beim Projektstart vorhanden, allerdings musste am Anfang sowieso einige Zeit in die Einarbeitung der Technologien gesteckt werden. Zudem mussten wegen Unklarheiten zusätzliche Informationen angefragt werden. Auf einige dieser Anfragen erhielten wir keine Rückmeldung, weswegen wir eigene Annahmen getroffen haben. Als verbesserte Massnahme wäre hier aufzuführen, falls fehlende Informationen nicht rechtzeitig geliefert werden, das Projektteam selber entscheidet.

R3 Ausfall der Infrastruktur / Datenverlust
Die Infrastruktur war nie unerwartet nicht erreichbar. Bei Wartungsarbeiten wurden wir stets vorgewarnt. Datenverlust ist auch keiner aufgetreten. Durch diese Risiko wurden wir nicht negativ beeinflusst.

R4 Benötigte Infrastruktur verspätet
Dieses Risiko ist eingetroffen und die aufgeführte Massnahme konnte leider nur teilweise umgesetzt werden. Für die Einarbeitung in NSO und MPLS konnte ein eigenes GNS3 mit älteren virtuellen Routern eingerichtet werden. Da wir keine Images für aktuelle XR und JUNOS Geräte hatten, konnten wir diese auf unserem GNS3 nicht einrichten. Durch die Verspätung konnte der Support für diese beiden Systeme nicht umgesetzt werden.

R5 Schwierigkeiten durch neue Technologien
Durch das Fehlen von Entwicklungserfahrung mit NSO ist definitiv einige Zeit verloren vergangen. Als künftige Massnahme könnte die Hilfe beim Experten, falls möglich, früher angefragt werden.

R6 Mangelnde Funktionalität der NSO Northbound APIs
Durch RESTCONF konnten wir alle benötigten Funktionen des NSO ansprechen, dieses Risiko ist somit nicht eingetreten.

Quellen

[14a] *Network Services Orchestrator Network Element Drivers*. Cisco Systems Inc. Nov. 2014. URL: https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/network-services-orchestrator/datasheet-c78-734669.html (besucht am 05. 03. 2019).

[14b] *Tail-f Systems is now part of Cisco*. Cisco Systems Inc. 2014. URL: https://www.cisco.com/c/en/us/about/corporate-strategy-office/acquisitions/tail-f.html (besucht am 04. 03. 2019).

[16a] *Cisco Network Services Orchestrator Enabled by Tail-f*. Cisco Systems Inc. 2016. URL: https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/network-services-orchestrator/datasheet-c78-734576.pdf (besucht am 04. 03. 2019).

[16b] *Infoblox REST API Referemce Guide*. Infoblox. Okt. 2016. URL: https://www.infoblox.com/wp-content/uploads/infoblox-deployment-infoblox-rest-api.pdf (besucht am 05. 03. 2019).

[Bar17] David Barroso. *YANG for dummies*. 14. Aug. 2017. URL: https://napalm-automation.net/yang-for-dummies/ (besucht am 05. 03. 2019).

[Can16] Cisco Canada. *NSO: Network Service Orchestrator enabled by Tail-f Hands-on Lab (Slide 77, Angepasst)*. 19. Mai 2016. URL: https://de.slideshare.net/CiscoCanada/nso-network-service-orchestrator-enabled-by-tailf-handson-lab (besucht am 14. 03. 2019).

[Cha15] Airheads Broadcasting Channel. *L2VPN VPLS management with iMC*. 20. Okt. 2015. URL: https://youtu.be/uy1s1wp5zFE?t=618 (besucht am 12. 03. 2019).

[Cis18a] Cisco. *NSO Core Concepts*. 2018. URL: https://developer.cisco.com/docs/nso/?utm_sq=fvimn9eeu5#!core-concepts (besucht am 05. 03. 2019).

[Cis18b] Inc. USA Cisco Systems. *NSO 4.7 Development [PDF vom Experten]*. 21. Juni 2018.

[Cis18c] Inc. USA Cisco Systems. *NSO 4.7 Manual Pages [PDF, generiert aus dem NSO Installer]*. 21. Juni 2018. URL: https://developer.cisco.com/docs/nso/#!getting-nso/getting-nso (besucht am 02/2019).

[Inf] Infoblox. *Infoblox Logo*. URL: https://www.exclusive-networks.com/be/wp-content/uploads/sites/3/2017/12/infoblox_TRAINING_PAGE_header_logo-1.png (besucht am 05. 03. 2019).

[Jac17] Don Jacob. *A Guide to MPLS VPN Fundamentals*. 5. Mai 2017. URL: https://www.packetdesign.com/blog/a-guide-to-mpls-vpn-fundamentals/ (besucht am 05. 03. 2019).

[McG14] Shamus McGillicuddy. *Cisco buys YANG wizard Tail-F Systems for telco network orchestration*. TechTarget. 17. Juni 2014. URL: https://searchnetworking.techtarget.com/news/2240222890/Cisco-buys-YANG-wizard-Tail-F-Systems-for-telco-network-orchestration (besucht am 04. 03. 2019).

[NÅ19] Håkan Niska und Martin Åkerström. *LABSPG-2442 Lab Guide*. Feb. 2019. URL: https://clnv.s3.amazonaws.com/2019/eur/pdf/LABSPG-2442-LG.pdf (besucht am 19. 03. 2019).

[Sri18] Vikas Srivastava. *Network Services Orchestrator Value Proposition*. 8. Mai 2018. URL: http://blog.devopssimplified.com/Cisco_NSO (besucht am 05. 03. 2019).

[Tec14] CA Technologies. *CA Spectrum - Main Windows for MPLS*. 23. Aug. 2014. URL: https://docops.ca.com/ca-spectrum/10-0/en/managing-network/mpls-transport-manager/managing-your-lsp-data/viewing-lsp-details/main-window-for-mpls (besucht am 12. 03. 2019).

[Wik19] Wikipedia. *YANG Artikel*. 20. Feb. 2019. URL: https://en.wikipedia.org/wiki/YANG (besucht am 05. 03. 2019).

Glossar

ANX Advanced Netconf Explorer.

BA Bachelorarbeit.

CDB Configuration Database.

eBGP exterior Border Gateway Protocol.

FUB Führungsunterstützungsbasis.

HSR Hochschule für Technik Rapperswil.

iBGP interior Border Gateway Protocol.
IFS Institut für Software.
IMC Intelligent Management Center.
INS Institute for Networked Solutions.
IPAM IP Adress Management.

LDP Layer Distribution Protocol.
LPM Longest Prefix Match.
LSP Label Switching Path.
LUX LUcid eXpect scripting.

MIB Management Information Base.
MPLS Multiprotocol Label Switching.

NCS Network Control Systen.
NED Network Element Driver.
NSO Network Services Orchestrator.

RPC Remote Procedure Call.
RUP Rational Unified Process.

SA Studienarbeit.
SAD Software Architecture Document.
SNMP Simple Network Management Protocol.
SVM support vector machine.

VRF Virtual Routing and Forwarding.

FASTMAP Ein Algorithmus im NSO, der die Konfiguration von verschiedenen NSO Services auf die Konfiguration im NSO Device Manager mappt.

Infoblox Ein Tool für das Management von IT-Netzwerken mit einem eingebautem IPAM.
IOS-XE Betriebssystemsoftware für Cisco Geräte, welches auf einem Linux basiert und IOS darauf ausführt.
IOS-XR Betriebssystemsoftware für Cisco Geräte. Es basiert ebenfalls auf einem Linux, im Gegensatz zu IOS-XE teilt es sich aber keinen Code mit dem Original-IOS.

JUN-OS Betriebssystemsoftware für Juniper Netzwerk Geräte.

LFIB Label Forwarding Information Base, Tabelle die alle möglichen lokalen Pfade eines Routers in einem MPLS-Netzwerk beinhaltet.

MAAGIC Pythonspezifische API für Zugriff auf die CDB.
MAAPI Python und Java API um per Transaktionen auf die CDB zu schreiben.

NETCONF Protokoll zur geräteunabhängigen Konfiguration von Netzwerkgeräten.
NSO Service Ein NSO Package, welches spezifische Konfigurationen von Netzwerkgeräten vereinfacht, z.B. Die Erstellung eines MPLS-VPNs.
NSO Package Eine Erweiterung für das NSO, z.B. ein NSO Service oder ein NED.
NSO NED Eine Erweiterung für das NSO um mit bestimmten Geräten kommunizieren zu können.

Oper-Data Eine Art um Daten in der CDB zu speichern. Sie kann über externe Schnittstellen nur ausgelesen und nicht verändert werden.

Python Komponente Ein Teil eines NSO Services der Daten aggregiert und aufbereitet, meist das Herzstück.

Reactive FASTMAP Design-Pattern, das NSO Services das Verwenden von Nebenwirkungen erlaubt.
RESTCONF Nach RFC 8040 standardisierte HTTP Schnittstelle welche Zugriff auf einen Datastore mit YANG-spezifizierten Daten ermöglicht.

Service Instanz Eine spezifische Instanz eines NSO Services, z.B. ein MPLS-VPN zwischen Bern und Luzern.

YANG Sprache zur Beschreibung von Datenmodellen.

9 Anhang

A Projektplan

A.1 Zweck

Der folgende Abschnitt im Dokument erläutert das Vorgehen in Bezug auf das Projektmanagement der Bachelorarbeit „Intent Based Networking with NSO“ im Frühlingssemester 2019 an der Hochschule Rapperswil.

A.2 Einführung

Ziel dieser Bachelorarbeit ist es, eine Erweiterung für die Cisco Software „NSO“ zu erstellen, welche das Management von MPLS-VPNs erleichtern soll. Dies wird in Form eines Services für NSO und einer zusätzlichen externen Komponente realisiert, welche eine grafische Oberfläche für den Service zur Verfügung stellen soll.

Die Arbeit wurde vom INS der Hochschule für Technik Rapperswil (HSR) ausgeschrieben und findet in Zusammenarbeit mit der FUB der Schweizer Armee, als Industriepartner statt.

A.3 Organisation

Für die Projektplanung wird nach den Phasen vom Rational Unified Process (RUP) (Inception, Ellaboration, Construction, Transition) vorgegangen.

Der momentane Fortschritt und das weitere Vorgehen werden wöchentlich jeden Donnerstagnachmittag in einem Meeting mit dem Betreuer besprochen. Anschliessend wird im Team für die Nachbearbeitung und Arbeitsplanung noch ein Meeting abgehalten. Dieses Meeting ist vergleichbar mit einer Kombination aus Sprintreview und Sprintplanning aus dem SCRUM Prozess.

A.3.1 Organisationsstruktur

Teammitglieder
Das Team setzt sich zusammen aus Patrik Peng und Raphael Hämmerli, beides Informatikstudenten der HSR im sechsten Semester (FS19). Da das Team aus zwei Personen besteht, werden keine spezifischen Rollen den entsprechenden Mitgliedern zugeteilt. Dies ermöglicht höhere Flexibilität und soll unnötigen Zeitaufwand für die Koordination vermindern.

Betreuer und Co-Betreuer
Betreut wird die Arbeit von Prof. Laurent Metzger und Urs Baumann vom INS der HSR.

Industriepartner
Industriepartner dieser Bachelorarbeit ist die FUB der Schweizer Armee. Dabei stehen Herr Serge Pidoux und Herr Laurent Billas als Ansprechpersonen zur Verfügung

Experte
Marcel Witmer der Firma Cisco Schweiz wird die Arbeit als externer Experte begleiten und bewerten.

Gegenleser
Die Arbeit wird von Prof. Stefan F. Keller vom Institut für Software (IFS) gegengelesen.

A.4 Arbeitsphasen

Der grobe Projektplan inklusive Meilensteine orientiert sich am RUP mit den oben erwähnten Phasen. In der Inception-Phase wird der Projektplan ausgearbeitet, sowie die Anforderungen genauer analysiert und definiert. In der Ellaboration-Phase wird der Fokus stärker auf dem Erstellen der Softwarearchitektur liegen. Um sicherzustellen, dass die erstellte Architektur mit den gewählten Technologien die Anforderungen erfüllen kann, wird auf das Ende der Ellaboration ein Prototyp erstellt. Diese Konzepte werden anschliessend in der Construction-Phase konkret in die Realität umgesetzt. Als letzte Phase befasst sich die Transition-Phase mit der Übergabe des Produkts, dessen Einführung und dem Abschluss der Arbeitsdokumentation.

A.5 Infrastruktur

Für die Durchführung dieser Arbeit wird vom INS und der HSR Infrastruktur zur Verfügung gestellt und betreut. Dabei wird diese in diesem Abschnitt in Entwicklungs- und Betriebsinfrastruktur unterteilt.

A.5.1 Entwicklungsinfrastruktur

Arbeitsplatz
Die HSR stellt jedem Gruppenmitglied ein zugewiesener Arbeitsplatz im Gebäude 1 zur Verfügung, welcher für die Arbeit an der Bachelorarbeit verwendet werden kann. Dieser Arbeitsplatz ist zudem mit Monitoren und Peripherie ausgestattet.

Zeiterfassung
Für die Zeiterfassung wird eine Redmine-Instanz verwendet, welche auf einem Linux Server bei Teammitglied Patrik gehostet ist. Als zusätzliche Sicherheitsmassnahme wird diese Instanz täglich gesichert.

Dateiablage & Versionsverwaltung
Als Dateiablage und Versionsverwaltung wird ein git-Repository verwendet, welches bei dem Provider GitHub eingerichtet wurde. Dies bringt den zusätzlichen Vorteil, dass Änderungen einfach zwischen den Teammitgliedern ausgetauscht werden können, ohne dass diese verloren gehen.

Entwicklungsumgebung
Für die Entwicklung der Software werden die jeweiligen Privatgeräte der Teammitglieder verwendet. Um Datenverlust zu vermeiden, werden Änderungen an Dokumentation und Code möglichst rasch in das remote git-Repository hochgeladen.

Testgeräte
Das INS stellt virtuelle Router als Testgeräte zur Verfügung, welche den korrekten Geräten entsprechen, wo schlussendlich das Produkt verwendet werden soll. Mittels VPN Zugriff, können diese Geräte jedem Standort aus erreicht werden.

A.5.2 Betriebsinfrastruktur

Server
Das INS stellt zwei Virtuelle Maschinen zur Verfügung, welche für die Installation von NSO und dem Hosting der virtuellen Router verwendet werden. Diese Geräte sind auch via VPN Zugriff jederzeit erreichbar.

Die virtuelle Maschine mit der NSO Installation wird zudem für das Hosting der externen Komponente mit der grafischen Oberfläche verwendet.

GNS3
Bevor die Infrastruktur des INS zur Verfügung steht, wird für die Einarbeitung in Cisco NSO eine Kombination aus privaten Virtuellen Maschinen mit installiertem NSO und einer geteilten GNS3 Instanz mit Cisco Geräten verwendet.

A.6 Zeitplan & Meilensteine

Die folgende Tabelle zeigt die wichtigsten Termine und Meilensteine dieser Bachelorarbeit.
SW = Semesterwoche, KW = Kalenderwoche

Woche	Wochenstart	Meilensteine
Inception		
SW 1 / KW 8	18.02.2019	
SW 2 / KW 9	25.02.2011	
SW 3 / KW 10	04.03.2019	
SW 4 / KW 11	11.03.2019	14.03.2019: **M1 Anforderungsanalyse und Projektplan erstellt**
Ellaboration		
SW 5 / KW 12	18.03.2019	
SW 6 / KW 13	25.03.2019	
SW 7 / KW 14	01.04.2019	
SW 8 / KW 15	08.04.2019	
SW 9 / KW 16	15.04.2019	18.04.2019: **M2 Prototyp und SAD fertiggestellt**
Construction		
SW 10 / KW 17	22.04.2019	
SW 11 / KW 18	29.04.2019	
SW 12 / KW 19	06.05.2019	09.05.2019: **M3 UC-1 Implementiert**
SW 13 / KW 20	13.05.2019	
SW 14 / KW 21	20.05.2019	
SW 15 / KW 22	27.05.2019	27.05.2019: **M4 Alle erforderlichen Use Cases implementiert**
Transition		
SW 16 / KW 23	03.06.2019	
SW 17 / KW 24	10.06.2019	14.06.2019: **M5 Abgabe (12:00)**

Zusätzliche Termine

KW 17	25.04.2019	Zwischenpräsentation BA
KW 24	14.06.2019	16.00 - 20:00 Präsentation und Ausstellung BA
KW 27	19.06.2019	10.00 - 11:00 Schlusspräsentation BA

M1 Anforderungsanalyse und Projektplan erstellt (14.03.2019)
Bis zu diesem Meilenstein sollten folgende Punkte definiert worden sein:

- Projektplanung inklusive Meilensteinen
- Analyse und Spezifikation der Anforderungen

M2 Prototyp und SAD fertiggestellt(18.04.2019)
Zusätzlich zur Implementierung des Prototyps wird ein Software Architecture Document (SAD) erstellt. Sämtliche Architekturentscheidungen werden in diesem Dokument festgehalten. Wenn das Programm in den weiteren Meilensteinen um eine Funktion erweitert werden soll, wird immer zuerst die Architektur im SAD angepasst. So ist sichergestellt das die Architektur immer vollständig dokumentiert ist.

M3 UC-1 Implementiert(09.05.2019)
Zu diesem Zeitpunkt sollte der Use Case 1 komplett implementiert sein. Der erste Use Case beinhaltet die Mehrheit des Implementationsaufwandes. Daher wird für diesen einen Grossteil der verfügbaren Zeit eingeplant.

M4 Alle erforderlichen Use Cases implementiert(27.05.2019)
Die fundamentale Implementierung wurde abgeschlossen. Alle nicht-optionalen Use Cases sind zu diesem Zeitpunkt umgesetzt. Nach diesem Meilenstein herrscht Feature-Freeze für unser Projekt. Dies bedeutet, dass ab diesem Zeitpunkt keine weiteren Features mehr implementiert werden. Lediglich zur Behebung von Fehlern werden, wenn nötig, noch Anpassungen am Code vorgenommen.

M5 Abgabe (14.06.2019)
Die restliche Zeit des Projekts wird für die Erstellung der Dokumentation verwendet. An diesem Meilenstein ist die Arbeit komplett fertig und abgabereit. Bis am Mittag um 12:00 wird sie dem Betreuer abgegeben.

A.7 Risiken

Nachfolgend sind sie identifizierten Risiken für die Arbeit aufgelistet. Die Risiken sind klassifiziert in die Eintrittswahrscheinlichkeit und den Schaden, beides von niedrig bis hoch. Der klassifizierte Schaden ist der Restschaden der nach der Anwendung der Massnahmen übrig bleibt. Für die bekannten Risiken werden so weit wie möglich Vorsichtsmassnahmen getroffen, um die Eintrittswahrscheinlichkeit und die Auswirkungen zu minimieren.

ID	Risiko	Auswirkungen	Massnahmen	Wahrsch-einlichkeit	Schaden
R1	Ausfall durch Krankheit	Verzögerung des Projektes	Keine	Niedrig	Hoch
R2	Unklarheiten oder Verzögerungen in der Aufgabenstellung	Erstellung des Zeitplans wird erschwert, Zeit wird für falsche Ziele verwendet	Weiteres Vorgehen im Protokoll erfassen und absegnen lassen, fehlende Informationen anfordern	Hoch	Mittel -Hoch
R3	Ausfall der Infrastruktur / Datenverlust	Verzögerung in der Entwicklung oder sogar Verlust des Fortschritts	Sicherung sämtlicher Infrastruktur & Speichern der Daten auf gesicherten Medien (Privat oder Hoster)	Niedrig	Hoch
R4	Benötigte Infrastruktur kann nicht rechtzeitig bereitgestellt werden	Verzögerung in der Entwicklung	Ausweichen auf private Infrastruktur	Mittel	Niedrig
R5	Schwierigkeiten durch Umgang mit neuen Technologien	Zeitverlust oder erarbeiten von nicht optimalen Lösungen	Erforschen der Technologien in der Inception Phase	Hoch	Mittel
R6	Northbound-APIs von NSO bieten nicht die erforderliche Funktionalität	Umsetzung von einigen Features wird erschwert oder verunmöglicht	Keine	Niedrig	Hoch

A.8 Qualitätsmassnahmen

Um eine gute Qualität der Arbeit sicherzustellen, werden folgende Massnahmen ergriffen:

A.8.1 Git Flow

Sämtlicher Code und alle Dokumente werden in einem Git-Repository gespeichert. Jeder Pull-Request muss vom anderen Teammitglied gegengelesen werden, bevor dieser gemerged wird. So wird sichergestellt, dass alle Änderungen von beiden Teammitgliedern überprüft wurden.

A.8.2 Dokumentation

Für die Dokumentation der Applikation, sowie für die Bachelorarbeit wird LaTeX verwendet. Die Dokumente werden im Git Repository eingecheckt und werden, wie der Code, nur mit entsprechendem Review akzeptiert.

A.8.3 Zwischenpräsentation

In der Mitte der Arbeit wird eine Zwischenpräsentation gehalten, in der der momentane Fortschritt der Arbeit vorgestellt wird. Sie bietet dem Industriepartner die Möglichkeit Feedback einzubringen und es wird sichergestellt dass wir weiterhin auf Kurs sind.

A.8.4 Linting Tools

Auf den Entwicklungsumgebungen werden Linting Tools eingesetzt, welche den Code auf Style-Verletzungen und weitere Probleme überprüft. Somit soll verhindert werden, dass schlechter Code in das Projekt fliesst. Zudem hilft dies, den Code-Style zwischen den Teammitgliedern einheitlich zu halten.

B Installations- und Bedienungsanleitung

B.1 Inbetriebnahme des NSO

Folgender Abschnitt beschreibt die Inbetriebnahme des NSO und die Vorbereitungen, welche dafür getroffen werden müssen.

B.1.1 Installation und Start des NSO

Installationsschritte für das NSO sind in der offiziellen Dokumentation einsehbar und werden hier nicht nochmals explizit aufgelistet. Das NSO kann entweder global auf dem System, oder lokal z.B. im Benutzerordner installiert werden. Für einen produktiven Einsatz wird eine globale Installation empfohlen.

Zusätzlich zu den offiziell aufgelisteten Anforderungen wird für den Betrieb des MPLS-L3VPN Packages Python mit der Mindestversion 3.7.2 benötigt.

Lokale Installation

Nach einer lokalen Installation kann mit folgenden Befehlen ein NSO Arbeitsverzeichnis erzeugt und das NSO darin gestartet und gestoppt werden:

```
ins@sr-000034:~$ ncs-setup --dest $HOME/ncs-run
ins@sr-000034:~$ cd ~/ncs-run
ins@sr-000034:~/ncs-run$ ncs
ins@sr-000034:~/ncs-run$ ncs --stop
```

Auflistung 5: Lokale Installation und Start vom NSO

Globale Installation

Wird das NSO global auf dem System installiert, kann es mittels Systemd Service gestartet, oder gestoppt werden:

```
ins@sr-000034:~$ sudo systemctl start ncs.service
ins@sr-000034:~$ sudo systemctl stop ncs.service
```

Auflistung 6: Start und Stop mit Systemd

Configdatei

Unter dem Pfad `/etc/ncs/` für globale Installationen, oder im Arbeitsverzeichnis bei einer lokalen Installation befindet sich die Konfigurationsdatei `ncs.conf`, welche weitere Möglichkeiten zur Konfiguration bietet. Darin muss beachtet werden, dass die RESTCONF Schnittstelle aktiviert ist.

B.1.2 Installation des NED und MPLS-L3VPN Package im NSO

Für die Kommunikation mit den Cisco XE Routern musste ein eigener NED kompiliert werden, wie zu einem früheren Zeitpunkt in dieser Arbeit bereits beschrieben wurde. Unser NED wurde mit der NSO Version 4.7 für Cisco IOS XE 16.09.02 kompiliert. Für andere Versionen wurde der NED nicht getestet und wird sehr wahrscheinlich nicht funktionieren.

Die Installation des NED ist relativ simpel. Dafür muss lediglich die Archivdatei `nso0_XE-01.7z` in das `packages` Verzeichnis des NSO entpackt werden. Bei einer globalen Installation ist dieses Verzeichnis unter `/var/opt/ncs/packages` zu finden.

Mit gestartetem NSO kann in dessen Konsole anschliessend der NED mittels dem Befehl `packages reload` geladen werden.

Für das MPLS-L3VPN Package kann gleich vorgegangen werden. Dabei wird stattdessen die Archivdatei `MPLS-L3VPN.7z` in das Verzeichnis `packages` entpackt. Auch das MPLS-L3VPN Package wurde nur mit den bereits erwähnten Versionen vom NSO und Cisco IOS getestet.

```
/var/opt/ncs/packages
|-- MPLS-L3VPN
\-- nso0_XE-01

2 directories, 0 files
```

Auflistung 7: Beispielinhalt des Packages Verzeichnis bei einer globalen Installation

```
ins@sr-000034:~$ ncs_cli -C

ins connected from 10.0.5.137 using ssh on sr-000034
ins@ncs# packages reload
reload-result {
    package MPLS-L3VPN
    result true
}
reload-result {
    package nso0_XE-01
    result true
}
```

Auflistung 8: Ausgabe des Befehls packages reload

B.1.3 Vorbereitung der NSO Konfiguration

Authgroup

Zu Beginn muss im NSO eine Authgroup erstellt werden, welche für die Geräte verwendet wird. Dies kann mittels NSO Kommandozeile folgendermassen durchgeführt werden:

```
admin@ncs# config
admin@ncs(config)# devices authgroups group <netconf>
admin@ncs(config-group-netconf)# default-map remote-name <username>
admin@ncs(config-group-netconf)# default-map remote-password \
<password>
admin@ncs(config-group-netconf)# default-map remote-secondary-password \
<password>
admin@ncs(config-group-netconf)# commit
```

Auflistung 9: Erstellung der Authgroup

Eingaben in <Klammern> sind Beispielinformationen und können entsprechend angepasst werden.

Device Groups

Damit die Geräte später entsprechend als PE oder CE markiert werden können, werden im NSO dafür Device Groups erstellt. Die Namen der Device Groups werden von der Python Komponente des NSO Services verwendet und können nicht geändert werden. Mit der NSO CLI wird dies so gemacht:

```
admin@ncs# config
admin@ncs(config)# devices device-group MPLS-CE
admin@ncs(config-device-group-MPLS-CE)# devices device-group MPLS-PE
admin@ncs(config-device-group-MPLS-PE)# commit
```

Auflistung 10: Erstellung der Device Groups

Infoblox Config

Für die Verwendung mit Infoblox wird im NSO ebenfalls zusätzliche Konfiguration benötigt. Dies beinhaltet Zugangsinformationen und IP-Netzranges welche für die Reservationen verwendet werden sollen. Dies kann mit folgenden Befehlen in der CLI konfiguriert werden:

```
admin@ncs# config
admin@ncs(config)# mpls_global infoblox_config ip <123.123.123.123>
admin@ncs(config)# mpls_global infoblox_config username <admin>
admin@ncs(config)# mpls_global infoblox_config password <password>
admin@ncs(config)# mpls_global infoblox_config ignore_ssl <true/false>
admin@ncs(config)# mpls_global \
infoblox_config external_networks <10.0.0.0> mask <255.0.0.0>
admin@ncs(config-external_networks-10.00.0.0)# mpls_global \
infoblox_config transit_networks <172.16.0.0> mask <255.240.0.0>
admin@ncs(config-transit_networks-172.16.0.0)# commit
```

Auflistung 11: Globale Infoblox Konfiguration

Mittels dem Befehl `mpls_global infoblox_config test_connection` kann in der CLI die Infoblox Verbindung validiert werden.

BGP Route Maps

Im NSO müssen die Namen der BGP Route Maps Namen konfiguriert werden, sodass diese mit denjenigen in der MPLS Konfiguration übereinstimmen.

```
admin@ncs# config
admin@ncs(config)# mpls_global bgp_route_maps primary_map_in \
<BGP_PRIMARY_IN_RMP>
admin@ncs(config)# mpls_global bgp_route_maps primary_map_out \
<BGP_PRIMARY_OUT_RMP>
admin@ncs(config)# mpls_global bgp_route_maps secondary_map_in \
<BGP_SECONDARY_IN_RMP>
admin@ncs(config)# mpls_global bgp_route_maps secondary_map_out \
<BGP_SECONDARY_OUT_RMP>
admin@ncs(config)# commit
```

Auflistung 12: Konfiguration der Route Maps

B.1.4 Vorbereitung der Netzwerkgeräte

Grundkonfiguration

Die Netzwerkgeräte müssen bereits über eine Grundkonfiguration verfügen, damit diese vom NSO verwendet werden können.

- IP Konnektivität zwischen Routern und mit dem NSO
- MPLS Setup für PE Router (inklusive Route-Maps)
- Loopback0 IP muss auf PE und CE Routern bereits gesetzt sein
- NETCONF Zugang via SSH
- Zugangsdaten

Geräte im NSO erfassen

Wenn die Geräte und das NSO entsprechend vorbereitet sind, können die zu steuernden Geräte nun im NSO hinzugefügt werden.

```
admin@ncs# config
admin@ncs(config)# devices device <my-device>
admin@ncs(config-device-my-netconf-device)# address <10.20.0.151>
admin@ncs(config-device-my-netconf-device)# port <830>
admin@ncs(config-device-my-netconf-device)# authgroup <netconf>
admin@ncs(config-device-my-netconf-device)# device-type netconf
admin@ncs(config-device-my-netconf-device)# state admin-state unlocked
admin@ncs(config-device-my-netconf-device)# commit
```

Auflistung 13: Neuerfassung eines Geräts

Der Default Port für NETCONF ist 830 und muss nur konfiguriert werden, wenn der verwendete Port davon abweicht. Nachfolgend können nun die SSH Host Keys mit folgendem Befehl vom Gerät abgerufen werden:

```
admin@ncs(config)# devices device <my-device> ssh fetch-host-keys
```

Auflistung 14: SSH Keys Fetchen

Anschliessend muss das hinzugefügte Gerät lediglich noch der korrekten Device Group hinzugefügt werden:

```
admin@ncs(config)# devices device-group <MPLS-CE/MPLS-PE> \
device-name <my-device>
admin@ncs(config-device-group-MPLS-PE) commit
```

Auflistung 15: Zuweisung der Device Group

B.1.5 Betrieb des NSO

Nach diesen Schritten sollte das NSO und die dazugehörigen Netzwerkgeräte entsprechend für den Einsatz bereit sein. Falls Probleme auftreten sind unter `/var/log/ncs` bei einer globalen Installation oder im `log` Ordner des Arbeitsverzeichnisses bei einer lokalen Installation Logfiles zu finden, welche bei der Problemlösung helfen können.

B.2 Inbetriebnahme der Rails Applikation

In diesem Abschnitt werden die benötigten Schritte für eine Inbetriebnahme der GUI Applikation aufgelistet.

B.2.1 Vorbereitungen auf dem Host

Da die GUI Komponente als Ruby on Rails Applikation realisiert wurde, muss auf dem Host Ruby in der Mindestversion 2.6.3 installiert sein. Für die Installation von Ruby wird `rbenv` empfohlen. Je nach System und Distribution müssen noch benötigte Systembibliotheken installiert werden, damit Ruby kompiliert werden kann.

Des Weiteren müssen Nodejs und der Package Manager Yarn auf dem Host installiert sein.

B.2.2 Installation von Rails

Als Nächstes kann nun das Archiv der GUI Applikation in ein gewünschtes Verzeichnis entpackt werden. Bei der Installation von Rails Dependencies (Gems) im nächsten Schritt können eventuell Fehler durch fehlende Systembibliotheken auftreten. Je nach OS und Distribution sind das unterschiedliche Bibliotheken, welche noch nachinstalliert werden müssen.

Wird die Applikation in einer Entwicklungsumgebung installiert, können mit folgendem Befehl die Abhängigkeiten installiert werden.

```
bundle install --jobs 4
```

Auflistung 16: Installation der Gems

Bei der Verwendung in einer Produktionsumgebung sind nicht alle Dependencies nötig. Dafür müssen jedoch die Assets (CSS und Javascript Files) vorkompiliert werden. Dies kann mit folgenden Befehlen bewerkstelligt werden:

```
bundle install --without development test --jobs 4
RAILS_ENV=production rails assets:precompile
```

Auflistung 17: Vorbereitungen für die Produktionsumgebung

B.2.3 Konfiguration von Rails

Mittels der Datei `application.yml` im `config` Verzeichnis können verschiedene Parameter der Rails Applikation entsprechend konfiguriert werden. Diese Parameter sind im Architekturkapitel genauer beschrieben.

B.2.4 Betrieb der Applikation

Wenn die obigen Schritte befolgt worden sind, sollte die Applikation nun mittels `rails s` gestartet werden können. Mittels `CTRL-C` in der Shell wird der Server wieder gestoppt. Bei Problemen sind im `log` Verzeichnis Logfiles vorhanden, welche bei der Problembehebung hilfreich sind.

Produktiver Betrieb

Bei einem produktiven Einsatz der Rails App empfiehlt es sich, einen Systemd Service zu erstellen, damit der Server einfach gestartet/gestoppt werden kann und bei einem Neustart des Hosts automatisch gestartet wird. Um dies zu bewerkstelligen, wird das Textfile `/etc/systemd/system/ba-gui.service` mit folgendem Inhalt erstellt:

```
[Unit]
Description=BA-GUI Rails application
After=network.target

[Service]
Type=simple
User=ins
Group=ins
WorkingDirectory=/opt/BA-GUI
ExecStart=/usr/bin/zsh -c "/home/ins/.rbenv/shims/bundle exec \
rails server -e production -b \
'ssl://0.0.0.0:8443?key=./key.pem&cert=./cert.pem'"


[Install]
WantedBy=multi-user.target
```

Auflistung 18: ba-gui.service

Mittels `sudo systemctl daemon-reload` wird der erstellte Service geladen. Nun kann die Rails Applikation wie andere Systemd Services mittels `systemctl` gemanagt werden. Der Logoutput kann mittels `journalctl -u ba-gui.service` abgerufen werden.

Die Datei `bin/deploy` im Applikationsverzeichnis ist ein rudimentäres Deployment Skript, welches während der Entwicklung der GUI Komponente verwendet wurde und bei Unklarheiten zusätzlich konsultiert werden kann.

B.3 Bedienung der grafischen Oberfläche

In diesem Abschnitt wird die Verwendung der grafischen Oberfläche kurz erklärt und anhand eines Beispiels ein VPN Service erstellt. Um nach einer frischen Installation einen VPN Service zu erstellen, soll nach folgedem Schema vorgegangen werden:

- Globale Einstellungen prüfen
- Gerätestatus kontrollieren
- Logische Verbindungen zwischen PE und CE Routern erstellen
- Globale Sites erstellen
- VPN Service erstellen

Je nach Environment, welches für den Betrieb der Applikation gewählt wird, sind die URLs für den Zugriff auf die grafische Oberfläche unterschiedlich. Wird die Applikation im Development Environment auf dem eigenen Gerät gestartet, ist die Website unter `http://localhost:3000` erreichbar. Im Production Environment ist die Website via HTTPS unter der URL `https://<IP-DES-HOSTS>:8443` zu erreichen.

B.3.1 Globale Einstellungen

Über die Navigationsleiste am oberen Bildschirmrand können die Globalen Einstellungen erreicht werden. Auf dieser Seite können konfigurierte Einstellungen eingesehen werden, Zugangsinformationen fürs Infoblox angepasst und dessen Verbindung überprüft werden.

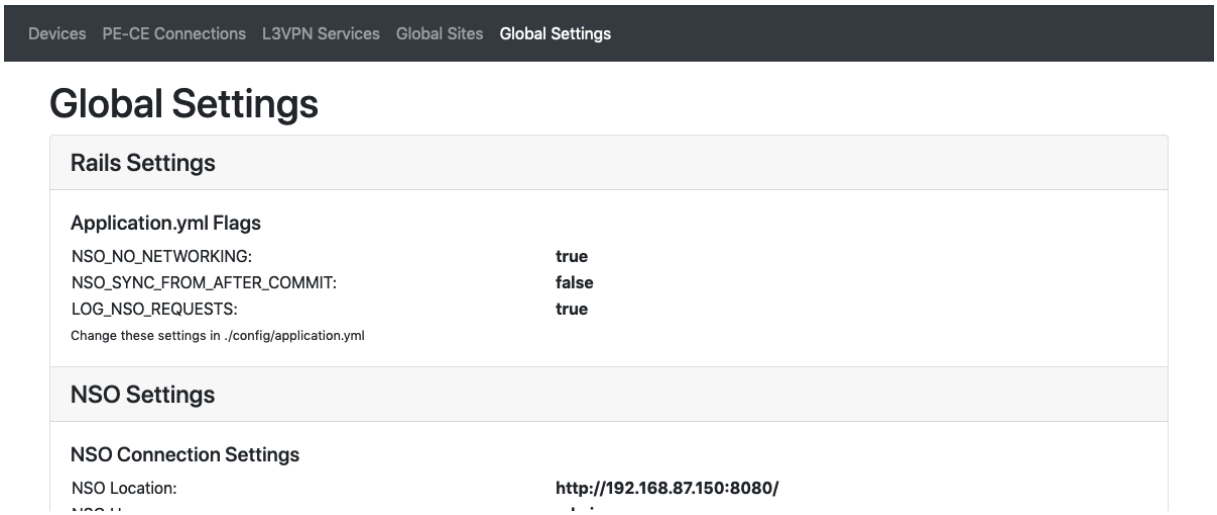

Abbildung 40: Seite mit globalen Einstellungen

B.3.2 Gerätestatus

Wenn alle Einstellungen korrekt sind, kann zum nächsten Punkt fortgefahren werden. Über den Menüpunkt Devices kann die Liste der im NSO eingetragenen Netzwerkgeräte angezeigt werden. Die Geräte werden anhand der zugeteilten Device-Group im NSO entsprechend gruppiert und deren Sync-Status abgefragt. Pro Gerät können mittels der Schaltfläche `Show Diff` die Unterschiede zwischen Konfiguration im NSO und im Gerät dargestellt werden. Mittels Selects und der `Selected Devices` Schaltfläche können Sync-To oder Sync-From Operationen auf den ausgewählten Geräten ausgeführt werden.

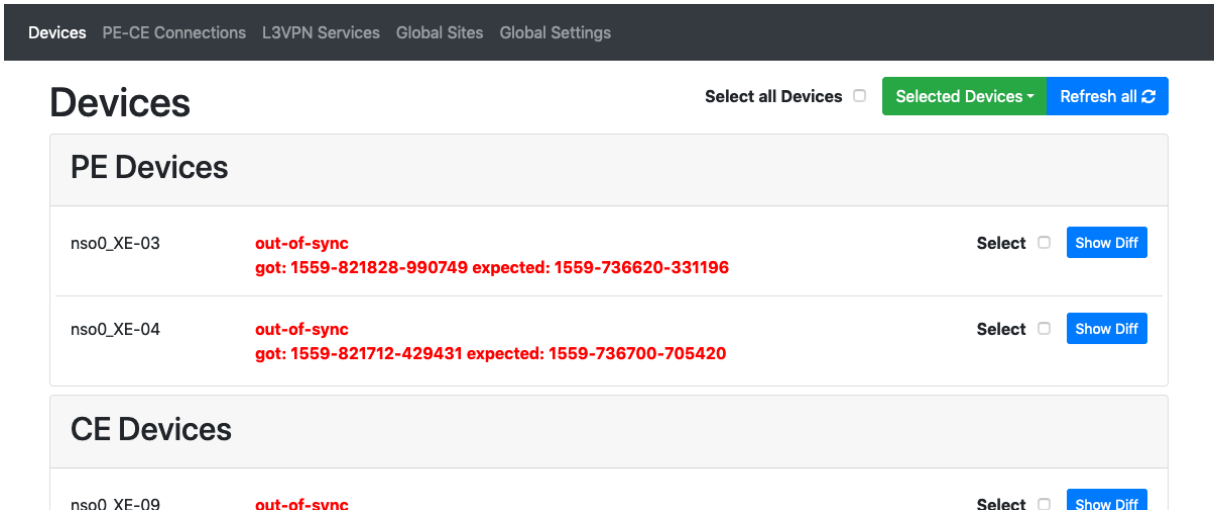

Abbildung 41: Auflistung der Geräte

B.3.3 PE-CE Verbindungen

Sind alle Geräte in sync, können nun logische Verbindungen zwischen PE und CE Routern erstellt werden. Unter dem Menüpunkt PE-CE Connections werden diese aufgelistet. Bestehende Verbindungen können zur Bearbeitung und Löschung in der Liste angeklickt werden. Das Formular zur Erstellung von neuen Verbindungen kann über die Schaltfläche `Create new PE-CE Connection` erreicht werden.

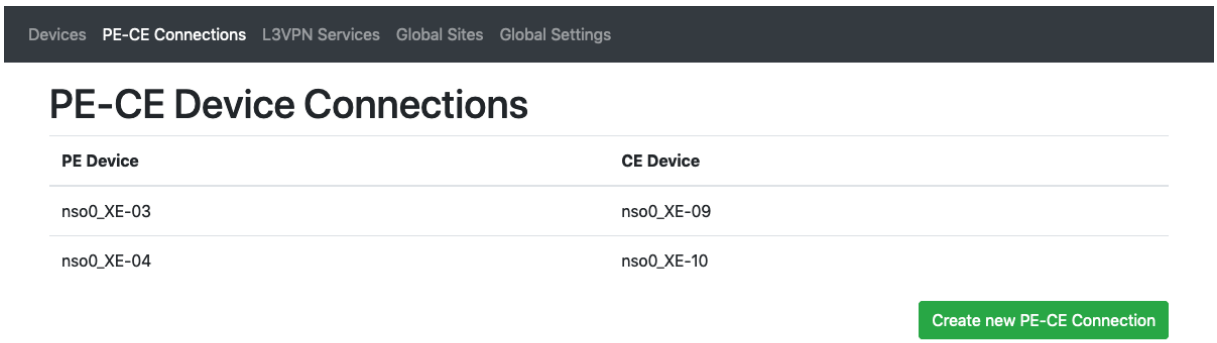

Abbildung 42: Auflistung der PE-CE Connections

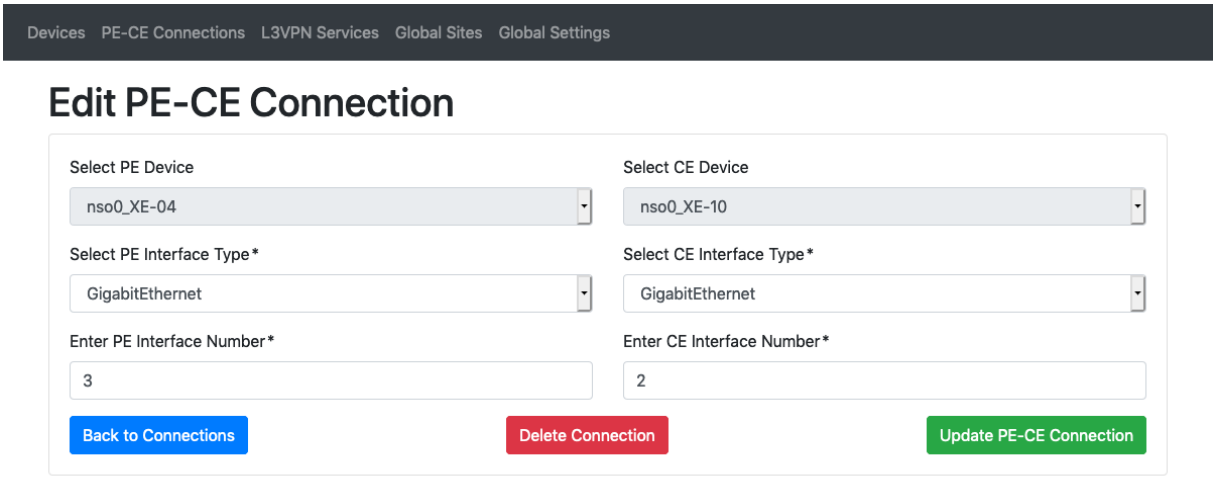

Abbildung 43: Bearbeitung einer PE-CE Connection

B.3.4 Globale Sites

Als weiteren Schritt werden nun sogenannte global Sites erstellt. Diese repräsentieren die effektiven physikalischen Standorte, welche dann einem VPN Service hinzugefügt werden können. Auch hier können bestehende Sites in der Liste ausgewählt werden. Über die Schaltfläche `Create new Site` können neue global Sites erstellt werden. Falls mehrere CE Router in einer Site ausgewählt werden wollen, können diese mit `CTRL-Klick` in der Liste ausgewählt werden.

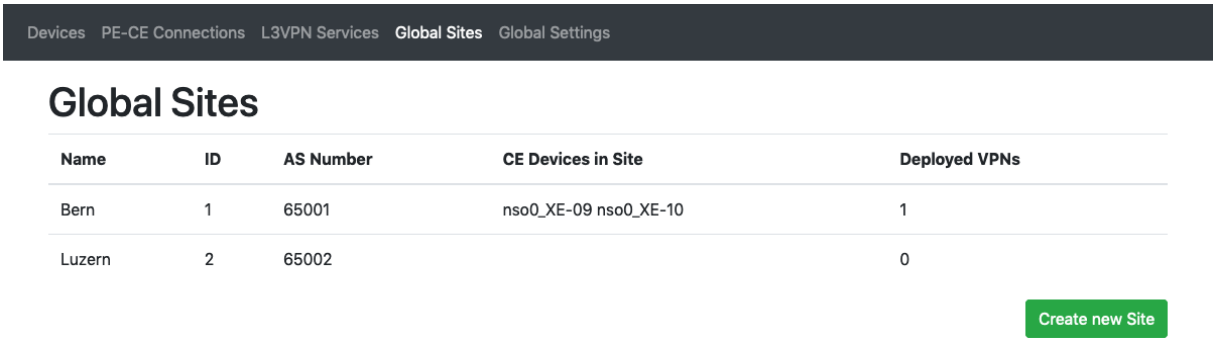

Abbildung 44: Auflistung von globalen Sites

Bei der Ansicht einer bereits erstellten globalen Site werden am Seitenende Services aufgelistet, welche auf bereits dieser Seite deployed sind. Mit Klick auf den Servicenamen können zusätzliche Informationen zu diesem Dargestellt werden.

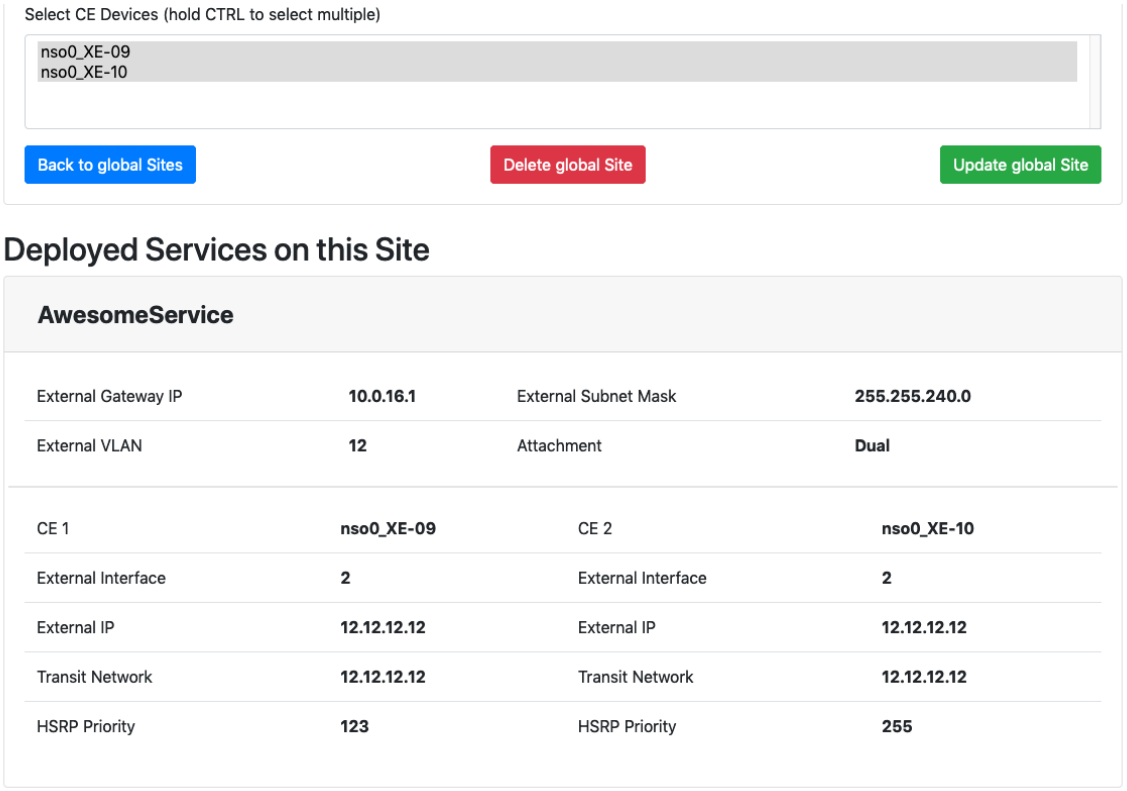

Abbildung 45: Konfigurierte Services auf dieser Site

B.3.5 VPN Service

Nun ist alles bereit für die Erstellung eines VPN Services. Mittels L3VPN Services Eintrag in der Navigationsleiste kann eine Liste mit bereits erstellten Services aufgerufen werden. Wieder kann ein bestehender Service aus der Liste ausgewählt werden, oder ein neuer mit der Schaltfläche `Create new Service` erstellt werden.

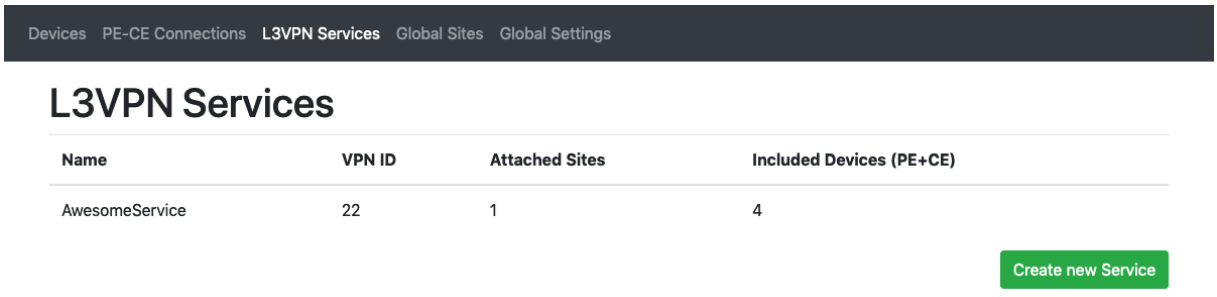

Abbildung 46: Auflistung von Services

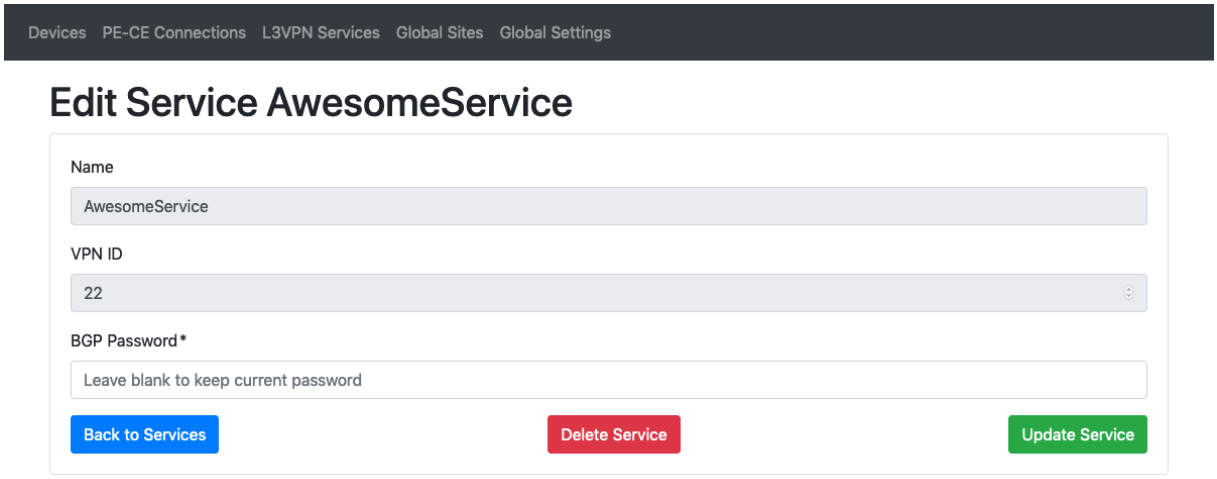

Abbildung 47: Frisch erstellter Service

Ein frisch erstellter Service bringt noch keine Funktionalität mit sich. Dafür muss diesem nun eine, oder mehrere Sites hinzugefügt werden. Dies kann mittels der Schaltfläche `Add a Site to this Service` bewerkstelligt werden.

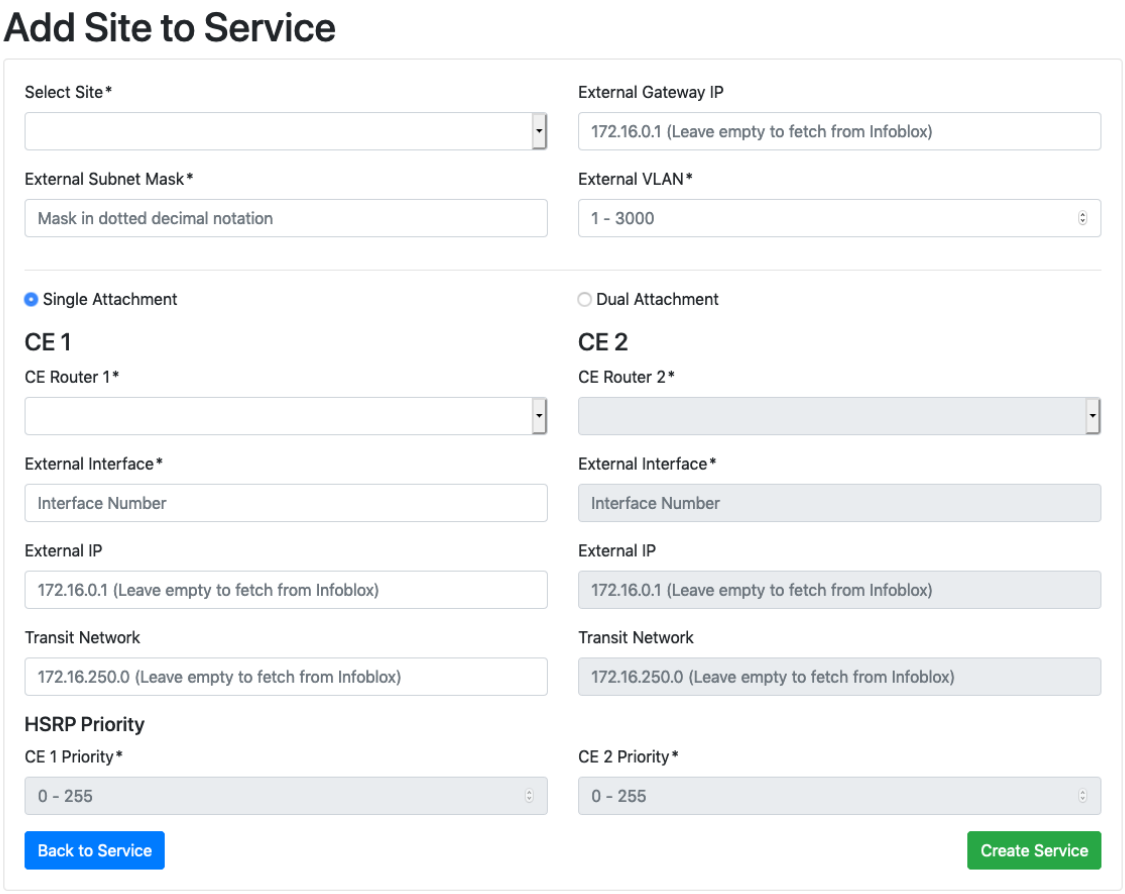

Abbildung 48: Site einem Service hinzufügen

Mittels dem Formular in Abbildung 48 werden verschiedene Konfigurationen erfasst. Die Namen der Felder sollten selbsterklärend sein. Gewisse Felder können leer gelassen werden, sodass diese Informationen automatisch aus dem Infoblox IPAM bezogen werden. Des Weiteren kann hier zwischen Single- und Dual-Attachment unterschieden werden.

Sind dem Service Sites hinzugefügt worden, werden diese in der Serviceansicht am Seitenende dargestellt. Mit Klick auf den Namen der Site werden erweiterte Informationen dargestellt. Diese Sites können auch mittels Select ausgewählt und wieder entfernt werden.

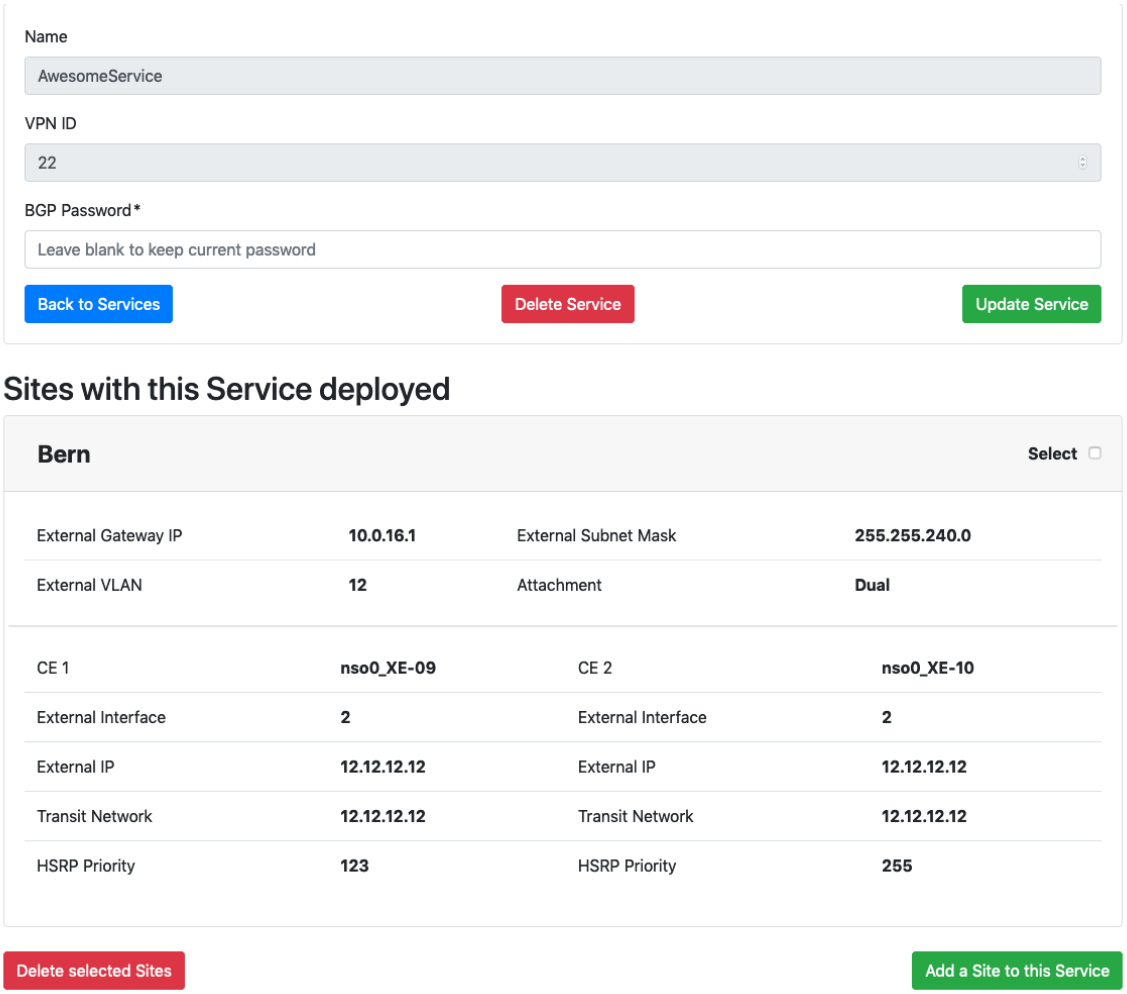

Abbildung 49: Details eines erstellten Service

C Aufgabenstellung

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Bachelor Thesis (FS2019): Intent based networking with NSO

Introduction

For the service automation on its new IP backbone, the swiss army would like to evaluate Cisco NSO, an industry-leading orchestration platform for hybrid networks.

NSO is a model driven (YANG) platform for automating the network orchestration. It supports multi-vendor networks through a rich variety of Network Element Drivers (NEDs). It supports the process of validating, implementing and abstracting the network configuration and network services, providing support for the entire transformation into intent based networking.

Goal

The goal of that thesis is to implement multi-vendor Service automation using NSO and to build a GUI that support the operation.

The configuration tool will be developed in the laboratory of the INS on Cisco and Juniper virtual routers.

Location, Date: _____

Signature Supervisor: _____

22 March 2019 Page 1 of 3

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Use Cases

ID	Use Case	Optional	Priority
UC-1	It should be possible to deploy a Layer 3 VPN on a network composed of virtual routers IOS-XE, IOS-XR and JUN-OS. The L3 VPN configuration should be deployed on the PEs and on the CEs. On a customized web GUI, the user should be able to fill in in a form the following parameters: - site (for example Dübendorf) - Single/ redundant access A drop-down list of the available CEs should pop up and the one to configure should be selected. - VPN name (for example Komsys) - VPN ID (for example 1010) - LAN mask (for example /24) - QOS policy (drop-down list) The IP addresses have to be taken from Infoblox and the reservation has to be documented in Infoblox (Description, router name, port). The Routing Protocol between PE-CE must be E-BGP and each L3 service will be placed in a VRF and transported by a sub-interface using a dot1q encapsulation (VPN ID) between PE and CE.	No	1
UC-1.1	L3VPN single attachement with IP addresses that are manually assigned (= no IPAM) with a single service.		
UC-1.2	L3VPN single attachement with IP addresses that are extracted from the Infoblox IPAM.		
UC-1.3	L3VPN single attachement with IP addresses that are extracted from the Infoblox IPAM and with several services (= with subinterfaces)		
UC-1-4	L3VPN dual attachement with IP addresses that are extracted from the Infoblox IPAM and with several services (= with subinterfaces)		
UC-2	On the web GUI, there should be an option to display a list of all the sites where a specific VPN has been deployed. The VPN name can be chosen from a drop-down list. By clicking check box(es), I should be able to modify the parameters or decommission the VPN service on one or more sites concurrently.	No	1

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

UC-3	It should be possible to get a graphical view of the out-of-sync configuration in the network with the option to sync from the NSO DB or to sync from the network.	No	2
UC-4	The Deployment of the Layer 3 VPN Service should support a triple control-plane backbone. (Flexalgo) On a customized web GUI, the user should be able to specify which backbone, Prefix-SID and ALGO Number is going to be used for a specific service. For example, backbone RED, Prefix-SID 16809 (ALGO 128) and Prefix-SID 16909 (ALGO 129) for backup path of backbone GREEN.	No	2
UC-5	It should be possible to deploy a Layer 2 Pseudowire from CE to CE. The web GUI should ask in a form for the following parameters: - Site A and Site B A drop-down list of the available CEs should pop up. The IP addresses has to be taken from Infoblox and the reservation has to be documented (Description, router name, interface type and number)	Yes	1
UC-6	Health check of the VPN status on a dashboard of the GUI. IP SLA will be used to guarantee the IP connectivity within a VRF. Thanks to IP SLA ICMP echo probes, end-to-End response time between sites can be measured and be displayed in that dashboard.	Yes	2
UC-7	The GUI should support the addition of new PEs and new CEs into NSO.	No	2