HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

ins
INSTITUTE FOR
NETWORKED SOLUTIONS

Digital Twin eines Netzwerkes

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2018

Autoren:	Patrik Peng, Raphael Hämmerli
Betreuer:	Prof. Beat Stettler
Co-Betreuer:	Urs Baumann
Projektpartner:	Institute for Networked Solutions
Experte:	Prof. Beat Stettler

1 Abstract

Netzwerke wurden in den letzten Jahren immer grösser und neue Protokolle auf verschiedenen Layern haben die Komplexität des Gesamtsystems massiv erhöht. So ist denn auch für einen Netzwerk-Ingenieur zunehmend schwierig, auftretende Konfigurationsfehler im Netzwerk zu finden und zu beheben. Eine grosse Hilfe könnte ein digitaler Zwilling des Soll Zustandes sein, welcher das Netz in einwandfreiem Zustand und mit korrekter Konfiguration abbildet. Treten später Fehler auf, kann der Ist-Zustand des Netzes mit dem gespeicherten Soll-Zustand des digitalen Zwillings verglichen werden und so die Ursache von Fehlern schneller gefunden werden. Da sich Netzwerke und die verwendeten Protokolle stets ändern, soll Digital Twin einfach anpass- und erweiterbar sein. Zudem sind in den meisten Netzwerken Geräte von verschiedenen Herstellern vorhanden, die Lösung soll also auch unabhängig vom Gerätehersteller arbeiten.

In dieser Arbeit wurde zuerst ein Konzept entwickelt, wie ein Netzwerk sauber abstrahiert und in einer Datenbank gespeichert werden kann. Als zweiter Schritt wurde dieser Ansatz als Webanwendung umgesetzt und an einem realen Netzwerk geprüft.

Eine besondere Herausforderung war es, die Architektur so zu gestalten, dass zusätzliche Layer einfach hinzugefügt werden können. Insbesondere die Implementation im Frontend benötigte viel Zeit, damit es möglichst unabhängig arbeitet und sich an Änderungen im Datenmodell anpasst.

Der Digital Twin als Webanwendung wurde in Ruby mit dem Ruby on Rails Framework entwickelt. Zur Visualisierung des Netzwerkes im Frontend wird die JavaScript-Library NeXt UI verwendet. Für die Kommunikation mit den Netzwerkgeräten wird Restconf verwendet. Zur Speicherung der Daten wird die Graphen Datenbank Neo4j benutzt.

Als nächste Arbeit am Digital Twin steht die Implementation von den übrigen Layern an. Momentan sind lediglich die ersten zwei vorhanden und für eine realistische Anwendung reicht dies noch nicht komplett aus. Auch kommuniziert die Anwendung momentan nur mit Cisco-Geräten, dies kann allerdings durch die flexible Architektur einfach erweitert werden.

2 Management Summary

Ausgangslage Je grösser ein Unternehmen wird, desto grösser und komplizierter wird auch sein Netzwerk. Für Netzwerkadministratoren wird es immer schwieriger, den Überblick zu behalten. Eine grosse Hilfe wäre ein digitaler Zwilling (Digital Twin), welcher das Netzwerk im Soll-Zustand abbildet. Wenn ein Problem im realen Netzwerk auftritt, kann dieses mit dem Digital Twin verglichen werden. Durch die Unterschiede in der Konfiguration kann dann die Ursache der Störung schnell identifiziert werden. Da sich Netzwerke und die verwendeten Protokolle stets ändern, soll Digital Twin einfach anpass- und erweiterbar sein. Zudem sind in den meisten Netzwerken Geräte von verschiedenen Herstellern vorhanden. Die Lösung soll also auch unabhängig vom Gerätehersteller arbeiten.

Vorgehen / **Technologien** Damit ein solcher Digital Twin realisiert werden kann, muss zunächst das Netzwerk abstrahiert werden, sodass die Informationen lesbar präsentiert werden können. In dieser Arbeit soll eine solche Architektur erstellt und daraus ein funktionierender Digital Twin als Webanwendung entwickelt werden.

Ergebnisse Als Ergebnis der Arbeit wurde eine Architektur erstellt, die das Netzwerk in einzelne Schichten aufteilt. Dadurch wird es für den Netzwerkadministrator einfacher spezifische Informationen zu finden und auch die Entwicklung wird erleichtert. Die Möglichkeit, zusätzliche Schichten einfach hinzuzufügen, gewährleistet eine saubere Erweiterbarkeit.

Diese Architektur wurde als Webanwendung realisiert. Sie unterstützt bereits alle geforderten Use-Cases, muss aber noch um weitere Schichten erweitert werden, um reif für den produktiven Einsatz zu sein. Die Anwendung kommuniziert momentan nur mit Geräten der Firma Cisco, da nur solche Testgeräte zur Verfügung standen.

Ausblick Der nächste Schritt ist das Hinzufügen von zusätzlichen Schichten. Zusätzlich kann die Benutzeroberfläche noch verbessert werden, was dem Benutzer die Bedienung der Software erleichtern soll und die Produktivität steigert. Als weiterer Punkt kann Unterstützung für zusätzliche Netzwerkgerätehersteller hinzugefügt werden, um das Einsatzgebiet vom Digital Twin zu erweitern.

Inhaltsverzeichnis

1 Abstract **1**

2 Management Summary **2**

3 Einführung und Planung **6**
3.1 Zweck . . . 6
3.2 Übersicht . . . 6
3.3 Organisation . . . 6
3.4 Organisationsstruktur . . . 6
3.5 Arbeitsweise . . . 6
3.6 Meilensteine . . . 7
3.6.1 M1: Anforderungen abgeklärt und Planung abgeschlossen (01.10.2018) . . . 7
3.6.2 M2: Prototyp & SAD (05.11.2018) . . . 7
3.6.3 M3: NeXt UI integriert; Meta Devices CRUD implementiert (26.11.2018) . . . 8
3.6.4 M4: Basisimplementation abgeschlossen (10.12.2018) . . . 8
3.6.5 M5: Erster Entwurf der Arbeit fertiggestellt (17.12.2018) . . . 8
3.6.6 M6: Abgabe an Betreuer (21.12.2018) . . . 8
3.7 Risikomanagement . . . 9
3.7.1 Risikoanalyse . . . 9
3.7.2 Umgang mit Risiken . . . 9
3.8 Infrastruktur . . . 10
3.8.1 Entwicklungsinfrastruktur . . . 10
3.8.2 Betriebsinfrastruktur . . . 10
3.9 Qualitätsmassnahmen . . . 11
3.9.1 Git Flow . . . 11
3.9.2 Unit Tests . . . 11
3.9.3 Linting Tools . . . 11
3.9.4 Dokumentation . . . 11

4 Anforderungen **11**
4.1 Anforderungsermittlung . . . 11
4.1.1 Einführung . . . 11
4.1.2 Auftrag . . . 12
4.2 Anforderungsanalyse . . . 12
4.2.1 Beschreibung . . . 12
4.2.2 Abgrenzung . . . 12
4.2.3 Use Cases . . . 13
4.2.4 Nicht funktionale Anforderungen . . . 14
4.2.5 Momentane Lösungen . . . 14
4.2.6 Erwartetes Produkt . . . 16
4.3 Anforderungsspezifikation . . . 16
4.3.1 Zu speichernde Informationen . . . 16
4.3.2 Auflistung Sublayer . . . 19
4.3.3 Benutzerinterface Anforderungen . . . 20
4.4 Generalisierung der Layer . . . 20
4.4.1 Voraussetzungen . . . 20
4.4.2 Layer 1 . . . 20
4.4.3 Layer 2 . . . 21
4.4.4 Zusammenfassung Architekturanforderungen Layer . . . 23
4.4.5 Fazit . . . 24

5 Architektur **25**
5.1 Architekturübersicht . . . 25
5.1.1 Webserver . . . 25
5.1.2 Datenbank . . . 25
5.1.3 Webbrowser . . . 25
5.1.4 Netzwerk . . . 26

5.2 Technologieentscheidungen . . . 27
5.2.1 Webserver . . . 27
5.2.2 Datenbank . . . 27
5.2.3 Webbrowser . . . 28
5.2.4 Netzwerk . . . 28
5.2.5 Fazit Technologieentscheidungen . . . 29
5.3 Layer . . . 31
5.3.1 Layer 1 . . . 31
5.3.2 Layer 2 . . . 31
5.3.3 Layer 2 Sublayer: Link Aggregation . . . 32
5.3.4 Layer 2 Sublayer: MAC . . . 32
5.3.5 Layer 2 Sublayer: Spanning Tree . . . 32
5.3.6 Layer 2 Sublayer: VLAN . . . 33
5.3.7 Layer 2 Sublayer: Per VLAN Spanning Tree . . . 33
5.3.8 Layer 3 . . . 34
5.3.9 Layer 3 Sublayer: Routing . . . 34
5.3.10 Layer 3 Sublayer: OSPF . . . 34
5.3.11 Datenmodell . . . 36
5.4 Datenspeicherung . . . 38
5.4.1 Datenstruktur . . . 38
5.5 Webserver . . . 40
5.5.1 Systemübersicht . . . 40
5.5.2 Sequenzdiagramme . . . 41
5.5.3 Architektur Packages . . . 45
5.6 Deployment Diagramm . . . 50

6 Build-Pipeline 51
6.1 Linting . . . 52
6.1.1 Rubocop . . . 52
6.2 TeamCity . . . 53
6.3 Versionsverwaltung . . . 54
6.3.1 Git-Flow . . . 54
6.4 Testing . . . 55
6.4.1 RSpec . . . 55
6.4.2 Getestete Funktionalitäten . . . 55
6.4.3 Integration . . . 55

7 Infrastruktur 56
7.1 Übersicht . . . 56
7.2 Physikalischer Server . . . 56
7.3 Virtuelle Server . . . 56
7.4 Backups . . . 57

8 Resultat 57
8.1 Implementierte Layer . . . 57
8.2 Use-Cases . . . 57
8.3 Nicht funktionale Anforderungen . . . 59
8.4 Testabdeckung . . . 59
8.5 Ausblick . . . 60
8.5.1 Hinzufügen neuer Layer . . . 60
8.5.2 Hinzufügen von zusätzlichen YANG-Models . . . 60
8.5.3 Erweiterung des Benutzerinterface . . . 60
8.5.4 Mehr Informationen über Tasks . . . 60
8.5.5 Parallelisierung . . . 61
8.6 Auswertung Zeitdaten . . . 61

9 Schlussfolgerungen 62
9.1 Architektur . . . 62
9.2 Projektmanagement . . . 63

9.3 Sonstige Probleme . . . 63
9.4 Projektfazit . . . 63
9.5 Persönliche Berichte . . . 64

10 Glossar 65

11 Anhang 67

A Aufgabenstellung 67

B Eigenständigkeitserklärung 68

C Anleitungen 69
C.1 Hinzufügen eines neuen Layers . . . 69
C.1.1 Beschreibung neue Klassen . . . 69
C.1.2 Ordnerstruktur und Namespaces . . . 70
C.2 Installation & Betrieb . . . 70
C.2.1 Voraussetzungen . . . 70
C.2.2 Installation . . . 70
C.2.3 Betrieb der Applikation . . . 71
C.2.4 Verwendung Digital Twin . . . 72

D Sitzungsprotokolle 78

3 Einführung und Planung

3.1 Zweck

Dieser Abschnitt dient zur Festlegung des Projektplans für die Studienarbeit „Digital Twin“ im Herbstsemester 2018 an der Hochschule Rapperswil.

3.2 Übersicht

Mit dieser Arbeit soll ein Werkzeug entwickelt werden, welches in der Lage ist, ein komplettes digitales Abbild eines physikalischen Netzwerks zu speichern und die gespeicherten Informationen später wieder abzurufen.
Im Beispielszenario wird vorerst der Soll-Zustand des Netzwerks in der Datenbank erfasst. Treten zu einem späteren Zeitpunkt nun Fehler im Netz auf, kann der momentane Ist-Zustand des Netzwerks mit dem abgespeicherten Zustand verglichen werden, um Fehler oder anderweitige Anomalien einfacher zu finden und zu beheben.

3.3 Organisation

Bei der Projektplanung wird im Groben nach den Phasen vom Rational Unified Process (Inception, Ellaboration, Construction, Transition) vorgegangen. Für die Organisation wird ein iterativer Prozess verwendet:

Der momentane Fortschritt und das weitere Vorgehen werden wöchtentlich jeden Montagnachmittag in einem Meeting mit dem Betreuer besprochen. Für die Arbeitsplanung im Team wird ein Meeting jeden Freitag abgehalten. Dieses Meeting ist vergleichbar mit einer Kombination aus Sprintreview und Sprintplanning aus dem SCRUM Prozess.

3.4 Organisationsstruktur

Das Team setzt sich zusammen aus Patrik Peng und Raphael Hämmerli, beides Informatikstudenten der HSR im fünften Semester (HS18). Betreut wird die Arbeit von Prof. Beat Stettler und Urs Baumann vom Institute for Networked Solutions der HSR.

Da die Gruppe nur aus zwei Personen besteht, werden keine spezifische Rollen den entsprechenden Mitgliedern zugeteilt. Dies ermöglicht höhere Flexibilität und soll unnötigen Zeitaufwand für die Koordination vermindern.

3.5 Arbeitsweise

Der grobe Projektplan inklusive Meilensteine orientiert sich am Rational Unified Process (RUP) mit den bereits erwähnten Phasen. In der Inception-Phase wird der Projektplan geschrieben und die Anforderungen genauer analysiert. Die Ellaboration und Construction-Phasen werden mittels mehrerer Iterationen abgearbeitet. Pro Feature wird zuerst die Architekturplanung um die gewünschten Funktionen erweitert und anschliessend werden diese realisiert. In der Ellaboration wird der Fokus stärker auf dem Designen der Architektur liegen, da diese erstmals erstellt wird. Um sicherzustellen, dass die erstellte Architektur der Ellaboration auch funktioniert, wird ein Prototyp erstellt. Die Transition wird sich vor allem mit dem Abschluss der Arbeitsdokumentation und dem abgeben der Arbeit befassen.

3.6 Meilensteine

Das Projekt hat eine Laufzeit vom 17.09.2018 bis 21.12.2018, für die Zeitplanung bis dahin sind folgende Meilensteine definiert:

Semesterwoche/KW	Datum	Meilenstein
Inception		
SW 1 / KW 38	17.09.2018	
SW 2 / KW 39	24.09.2018	
SW 3 / KW 40	01.10.2018	M1: Anforderungen abgeklärt und Planung abgeschlossen
Ellaboration		
SW 4 / KW 41	08.10.2018	
SW 5 / KW 42	15.10.2018	
SW 6 / KW 43	22.10.2018	
SW 7 / KW 44	29.10.2018	
SW 8 / KW 45	05.11.2018	M2: Prototyp & SAD fertig gestellt
Construction		
SW 9 / KW 46	12.11.2018	
SW 10 / KW 47	19.11.2018	
SW 11 / KW 48	26.11.2018	M3: NeXt UI integriert Meta Devices CRUD implementiert
SW 12 / KW 49	03.12.2018	
SW 13 / KW 50	10.12.2018	M4: Basisimplementation abgeschlossen
Transition		
SW 14 / KW 51	17.12.2018	M5: Erster Entwurf der Arbeit fertiggestellt
SW 14 / KW 51	21.12.2018	M6: Abgabe an Betreuer (12:00 Uhr)

3.6.1 M1: Anforderungen abgeklärt und Planung abgeschlossen (01.10.2018)

Folgende Punkte sollen in diesem Meilenstein definiert werden:

- Projektplanung inklusive Meilensteine sind definiert
- Analyse und Spezifikation der Anforderungen

3.6.2 M2: Prototyp & SAD (05.11.2018)

Für den Prototyp werden folgende Funktionen erwartet:

- Der Webserver ist funktionsfähig
- Das Webportal stellt die verfügbaren Daten visuell dar.
- Ein Administrator kann dem System neue Geräte und Verbindungen auf Layer 1 hinzufügen
- Der Server kann Layer 1 Informationen von einem Netzwerkgerät auslesen und in die Datenbank eintragen

Zusätzlich zur Implementierung dieser Funktionen wird auch 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.

3.6.3 M3: NeXt UI integriert; Meta Devices CRUD implementiert (26.11.2018)

Die Frontendbibliothek NeXt UI wurde komplett in die Applikation integriert, sodass potenzielle Geräte und Verbindungen bereits dargestellt werden können. Zudem können Meta Devices manuell erstellt, angezeigt, bearbeitet und gelöscht werden (CRUD).

3.6.4 M4: Basisimplementation abgeschlossen (10.12.2018)

Die fundamentale Implementation ist abgeschlossen und der erste Layer wurde gemäss der Architektur in der Applikation integriert. Devices, Interfaces, und Connections können manuell erfasst, bearbeitet, aufgelistet und gelöscht werden. Mittels Bootstraptasks können zudem Komponenten automatisch erfasst werden. Nach diesem Zeitpunkt werden am Code keine Änderungen mehr vorgenommen (Code Freeze).

3.6.5 M5: Erster Entwurf der Arbeit fertiggestellt (17.12.2018)

Ein erster Entwurf der definitiven Arbeit wurde erstellt.

3.6.6 M6: Abgabe an Betreuer (21.12.2018)

Die Arbeit wird finalisiert und bis am 21.12.2018 12:00 Uhr mittags in schriftlicher Form an den Betreuer abgegeben.

3.7 Risikomanagement

3.7.1 Risikoanalyse

Eine kurze Auflistung und Analyse der Projektrisiken sind in folgendem Bild als Tabelle ersichtlich.

Risikomanagement

Projekt:	DigitalTwin
Erstellt am:	05.10.2018
Autor:	Raphael Hämmerli &Patrik Peng
Gewichteter Schaden:	46.4

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten beim Eintreten
R1	Schwierigkeiten beim Setup der Entwicklungsinfrastruktur	Unerwarteter zeitlicher Mehraufwand durch Schwierigkeiten beim Setup und der Verkopplung von SCM(git), Timetracking(Redmine) und Build/Deploy/CI Tool(TeamCity)	20	25%	5	Gründliche Recherchen der verfügbaren Tools und verwendung von bereits genutzer Software	Setup muss trotzdem komplett abgeschlossen werden. Priorisierung der einzelnen Komponenten.
R2	Schwierigkeiten beim Setup der Produktinfrastruktur	Unerwarteter zeitlicher Mehraufwand durch Schwierigkeiten beim Setup von Backendserver, DBserver, etc.	20	25%	5	Gründliche Recherche und Evaluation der verwendeten Technologien	Priorisierung und evtl. Anpassung der verwendeten Komponenten
R3	Ausfall Serverinfrastruktur (Entwicklungs- und/oder Produktinfrastruktur)	Potenzieller Datenverlust und/oder eingeschränke Produktivität durch fehlerhafte Infrastruktur	20	2%	0.4	Korrektes Setup, sowie Monitoring der Infrastruktur Einführung von regulären und geprüften Backups Evtl. Datenredundanz (RAID)	Schadensbegrenzung durch Einspielung von Backups
R4	Erschwehrte Kommunikation mit den entsprechenden Netzwerkgeräten	Informationen über Netzwerkgeräte können nicht einfach über eine Schnittstelle abgefragt werden und es muss dadurch "improvisiert" werden.	50	40%	20	Vorhergehende Recherchen über verfügbare Schnittstellen und Bibliotheken, welche bereits verwendet werden können.	Entsprechende Situationsanalyse mit Betreuer
R5	Probleme/Bugs in verwendeten Bibliotheken	Zeitverzögerungen durch Probleme/Bugs in verwendeten Bibliotheken oder Frameworks	20	10%	2	Known issues im Vorhinein überprüfen	Schadensbegrenzung oder alternative Bibliothek verwenden
R6	Zeitverlust durch Ungeübtheit im Umgang mit LaTeX	Zeitverlust durch zuszätzlich benötigter Rechercheaufwand für die korrekte verwendung von LaTeX	10	80%	8	-	Bei zu starker Verzögerung Hilfe bei versierter Person beantragen
R7	Probleme / Inkompabilitäten der Integrierung	Probleme / Inkompabilitäten bei der Integrierung der Visualisierung von der Multicast-SA	15	40%	6	-	Priorisierung und evtl. Anpassung der Requirements

Abbildung 1: Risikoanalyse

3.7.2 Umgang mit Risiken

Für die bekannten Risiken werden so weit wie möglich Vorsichtsmassnahmen getroffen, um die Eintrittswahrscheinlichkeit und die Auswirkungen zu minimieren. Gewisse Features des Clients sind mit niedrigerer Priorität eingeplant. Falls unerwartete Zeitverzögerungen eintreffen sollten, werden diese Features unter Umständen gekürzt oder gestrichen und die verfügbare Zeit als Reserve genutzt.

3.8 Infrastruktur

Die Infrastruktur wird überwiegend in Entwicklungs- und Betriebsinfrastruktur aufgeteilt.

3.8.1 Entwicklungsinfrastruktur

Zeiterfassung
Für die Zeiterfassung wird eine bestehende Redmine Installation verwendet, welche auf einem Linux Server bei Patrik gehostet ist.

Dateiverwaltung
Als Dateiverwaltung wird ein git-Repository verwendet, welches bei GitHub gehostet ist.

Versionsverwaltung
Die Versionsverwaltung wird mit git realisiert, welches bei GitHub gehostet ist.

Continuous Integration/Deployment
Für Continuous Integration/Deployment ist der Einsatz von TeamCity geplant. Geplant wird dies auf einem Server bei Raphael zu hosten. Eventuell wird fürs Testing weitere Netzwerkhardware benötigt.

Entwicklungsumgebung
Die Entwicklung der Software findet auf dem Privatgerät des jeweiligen Teammitglieds statt. Um Datenverlust zu vermindern, sollten Änderungen möglichst rasch in das remote git-Repository hochgeladen werden.

Entwicklungshardware
Für die Entwicklung der Software werden verschiedene Netzwerkgeräte benötigt, damit die Software laufend mit den Geräten getestet werden kann.

3.8.2 Betriebsinfrastruktur

Server
Die Serverapplikation mit der Datenbank wird voraussichtlich auf dem gleichen Server gehostet, wie Redmine.

Client
Für den Betrieb des Clients wird lediglich ein Laptop/PC mit Netzwerkfunktionalität benötigt.

3.9 Qualitätsmassnahmen

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

3.9.1 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.

3.9.2 Unit Tests

Für die API Ebene werden Unit-Tests erstellt, die ein korrektes Funktionieren der API überprüfen. Diese Tests werden auf einer Separaten Test-Datenbank ausgeführt. Die Client-Funktionen werden nicht komplett getestet, da das Mocking des API-Servers zu viel Zeit beanspruchen würde. Diese Tests werden bei jedem Build ausgeführt und der Status and GitHub übermittelt. Solange dieser Status nicht ok ist, kann ein Pull-Request nicht gemerged werden. Damit wird verhindert, dass fehlerhafter Code integriert wird.

3.9.3 Linting Tools

Auf den Entwicklungsumgebungen, sowie auf der CI 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.

3.9.4 Dokumentation

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

4 Anforderungen

4.1 Anforderungsermittlung

4.1.1 Einführung

Grosse Netzwerke mit vielen Netzwerkkomponenten sind im Jahr 2018 keine Seltenheit. Netzwerke umspannen oft mehrere Gebäude, Städte oder gar Länder. Mit Zunahme der Grösse eines Netzwerks nimmt jedoch auch der Aufwand für Wartung und Fehlerbehebung zu.

Meist werden Netzwerke nur reaktiv überwacht, sprich es wird gemessen, wie sich das Netz verhält und reagiert im Fehlerfall. Oft ist jedoch die ursprünglich spezifizierte Topologie nicht bis ins Detail bekannt, oder es haben sich unerwartete Änderungen und kleine Fehler über den Zeitraum eingeschlichen. Viele dieser Fehler bleiben oft unentdeckt, solange sie keinen kritischen Einfluss auf die Netzwerkleistung haben. Sie werden meist erst bei einem grösseren Zwischenfall bemerkt, wenn das Netz genauer unter die Lupe genommen wird. Um jedoch ein Netzwerk dauerhaft sicher und stabil zu halten, muss sicher gestellt werden, dass das Netzwerk sich exakt so verhält wie geplant.

4.1.2 Auftrag

Wenn im Detail bekannt ist, wie die Netzwerktopologie spezifiziert ist, kann dies für die Fehleranalyse als Hilfsmittel verwendet werden. Die Idee ist, dass eine spezifizierte Netzwerktopologie digital erstellt wird, oder nach korrekter Einrichtung und Überprüfung eines physikalischen Netzwerks eine digitale Kopie dessen erstellt wird. Dieser digitale Zwilling (engl. Digital Twin) stellt den Soll-Zustand dar und repräsentiert das Netz in einwandfreiem Zustand. Der Digital Twin soll dabei hardwareunabhängig sein, sodass dieser bei einem Hardwarewechsel nicht aktualisiert werden muss. Zur Verifizierung der Spezifizierung, oder im Fehlerfall kann der Ist-Zustand des Netzes mit dem gespeicherten Soll-Zustand verglichen werden. Ersichtliche Differenzen zwischen den beiden Versionen sind ein Anhaltspunkt für mögliche Fehler und sollen bei dessen Behebung behilflich sein.

Zusätzliches Merkmal des Endprodukts ist eine einfache Erweiterbarkeit. Für zukünftige Entwickler soll es ohne grossen Aufwand möglich sein, neue Komponenten für das Produkt zu entwickeln.

4.2 Anforderungsanalyse

4.2.1 Beschreibung

Wie bereits im Namen „Digital Twin“ angedeutet, soll das Produkt einen digitalen Zwilling eines echten Netzwerks repräsentieren. Dieser Zwilling bildet den vorgesehenen Zustand des Netzwerkes im Normalbetrieb ab. Dazu ist es nötig, alle Zustandsinformationen einer Netzwerkkomponente zu erfassen. Eine Komponente beinhaltet immens viele Informationen, die einen Netzwerkadministrator rasch überfluten können. Deshalb werden die Informationen über das gespeicherte Netzwerk im Digital Twin nach den einzelnen Schichten des OSI-Layer Modells gruppiert. Dies erleichtert die Speicherung der Daten und ermöglicht dem Benutzer immer nur die für ihn relevanten Informationen anzeigen.

4.2.2 Abgrenzung

Der Digital Twin speichert die Topologie eines Netzwerks und kann bei Bedarf dieses mit den entsprechenden Informationen über Verbindungen und verwendete Protokolle repräsentieren. Bei Überprüfungen oder im Fehlerfall des Netzwerks kann der Digital Twin mit der physikalischen Netzwerkinfrastruktur verglichen werden, um Differenzen zwischen Soll-Zustand und Ist-Zustand zu erkennen. So kann einfach festgestellt werden, ob das Netzwerk der Spezifikation entspricht und im Fehlerfall als Assistenz dienen.

Der Digital Twin ist jedoch nicht als Monitoring-Tool gedacht und es sind keine Funktionen zur Echtzeitüberwachung geplant. Des Weiteren verhält sich der Digital Twin gegenüber dem Netzwerk rein passiv, sprich es werden zu keiner Zeit Änderungen am Netzwerk vorgenommen. Bei jeder Interaktion des Digital Twin mit den physikalischen Netzwerkkomponenten werden lediglich Informationen abgerufen und in einer Datenbank gespeichert. Die Speicherung der Zustandsdaten soll unabhängig vom Hardwarehersteller sein, jedoch begrenzt sich das automatische Auslesen der Daten vorerst auf Cisco-Geräte.

4.2.3 Use Cases

Durch die Analyse der Aufgabenstellung ergeben sich folgende Use Cases, welche vom Produkt abgedeckt werden sollten.

UC01 Netzwerk manuell erstellen [Priorität: 1] Der Benutzer erstellt eine neue Netzwerktopologie im System.

Jede einzelne Netzwerkkomponente wird mit den entsprechenden Informationen zu Verbindungen und Protokollen vom Benutzer manuell in das System eingetragen.

UC02 Bestehendes Netzwerk als Digital Twin erfassen [Priorität: 1] Der Benutzer erfasst eine bestehende Netzwerktopologie als Digital Twin.

Vorab wird vom Benutzer eine Liste von Geräten mit Informationen über Zugangsdaten von Netzwerkkomponenten erstellt, welche erfasst werden sollen. Nachfolgend werden die Informationen von den aufgelisteten physikalischen Netzwerkkomponenten abgefragt und in das System eingetragen.

UC03 Bearbeitung von Netzwerken [Priorität: 2] Der Benutzer bearbeitet ein bereits abgespeichertes Netzwerk.

Im Falle von Änderungen an der Netzwerktopologie oder der gespeicherten Informationen einer spezifischen Komponente können diese individuell angepasst werden.

UC04 Vergleich Soll-Ist Zustand [Priorität: 2] Der Benutzer vergleicht das momentane physikalische Netzwerk mit dessen abgespeicherten Digital Twin.

Die Netzwerkinformationen werden vom physikalischen Netzwerk abgefragt und mit denen des gespeicherten Digital Twin verglichen. Bei Differenzen werden diese dem Benutzer angezeigt.

Use-Case Diagramm Nachfolgend ein Use-Case Diagramm, welches die Use Cases darstellt.

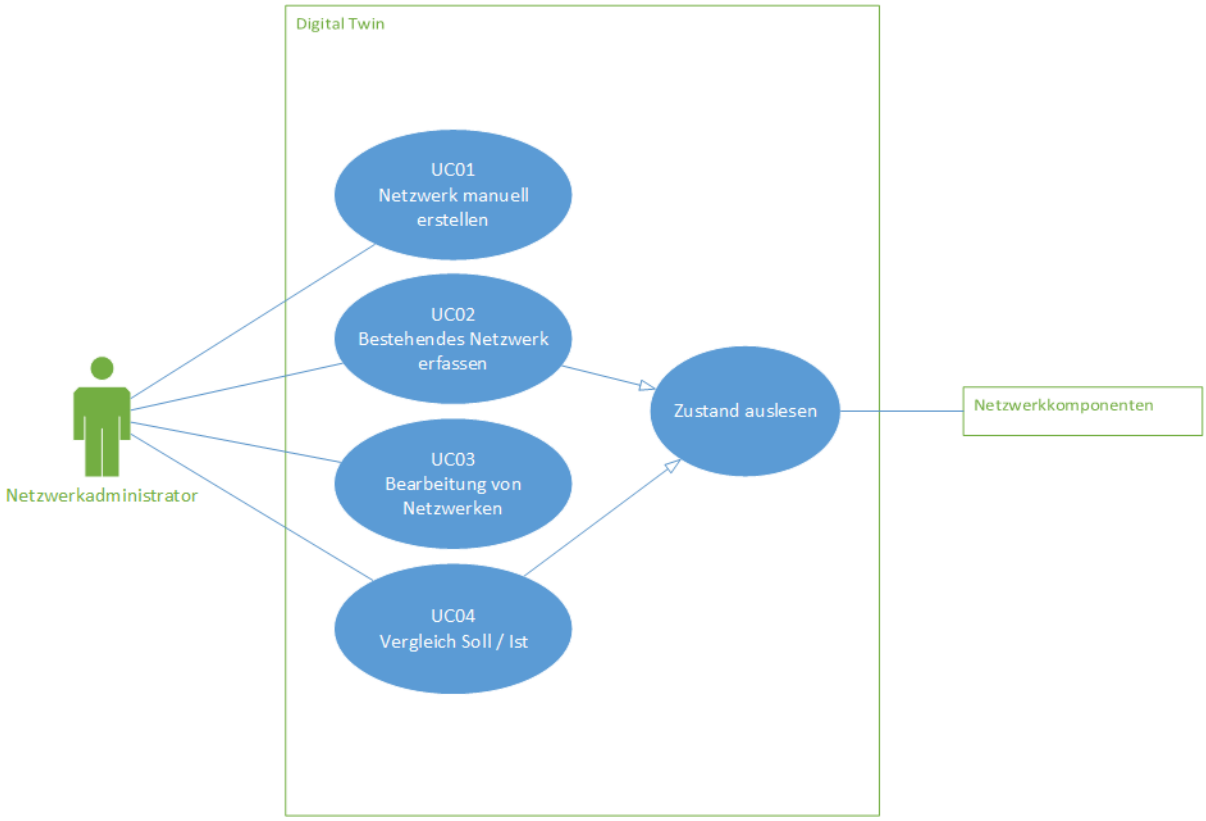

Abbildung 2: Use-Case Diagramm

4.2.4 Nicht funktionale Anforderungen

Sicherheit Keine vertraulichen Informationen werden im Klartext übermittelt. Jegliche Kommunikation zwischen Benutzer und Applikation, sowie dem physikalischen Netzwerk findet verschlüsselt statt. Sensitive Informationen, wie Passwörter, sollen verschlüsselt in der Datenbank gespeichert werden.

Skalierbarkeit Die Applikation ist in der Lage, auch grössere Netzwerktopologien mit 20 Komponenten oder mehr abzubilden.

Reaktionszeit Die Applikation wird in einem Zeitfenster von drei Sekunden auf Benutzerinteraktionen reagieren. Bei Vorgängen mit längerer Verarbeitungszeit wird dies dem Benutzer korrekt vermittelt. Die Applikation soll zu keinem Zeitpunkt den Benutzer uninformiert lange warten lassen.

Erweiterbarkeit Die Applikationsarchitektur wird so entworfen, dass zukünftige Komponenten einfach hinzugefügt werden können. Dies wird mit klarer Unterteilung in Komponenten und definierten Schnittstellen erreicht.

4.2.5 Momentane Lösungen

Seit es Netzwerke gibt, existiert auch ein Bedarf an Dokumentation dieses Netzwerks. Dadurch entstanden viele verschiedene Lösungen die darauf abzielen das Netzwerk digital festzuhalten und zu überwachen. Zwei kommerzielle Produkte die genauer betrachtet wurden, sind netTerrain logical und SolarWinds Network Topology Mapper.

netTerrain logical netTerrain logical ist eine Webbasierte Netzwerk-Dokumentierungs- und Visualisierungssoftware. Netzwerkdiagramme können automatisch erstellt werden und auch nach verschiedenen Layern dargestellt werden. Ein spezielles Feature ist das Change-Tracking, mit dem nachvollzogen werden kann welcher Systemadministrator welche Änderung am Netzwerk vorgenommen hat.

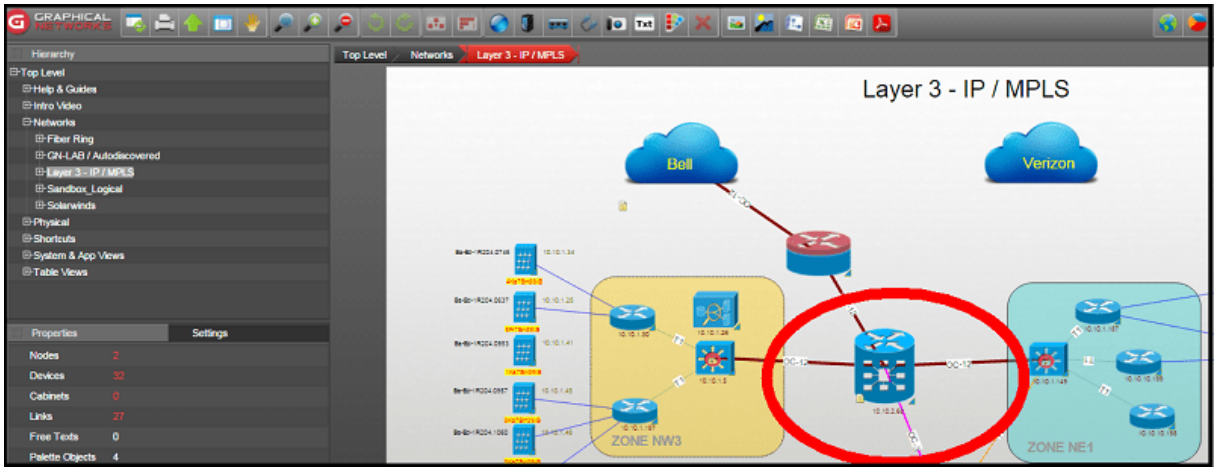

Abbildung 3: netTerrain logical [Sof18]

SolarWinds Network Topology Mapper Eine ähnliche Lösung bietet der Network Topology Manager. Er ermöglicht ebenfalls die automatische Erstellung von Netzwerkübersichten. Es werden mehrere Möglichkeiten zum Erkennen von Netzwerkgeräten bereitgestellt, unter anderem ICMP, SNMP oder das Cisco Discovery Protocol. Er fokussiert sich stark auf die Darstellung des Netzwerks und auf Berichten von Metriken wie Netzwerktraffic auf bestimmten Interfaces.

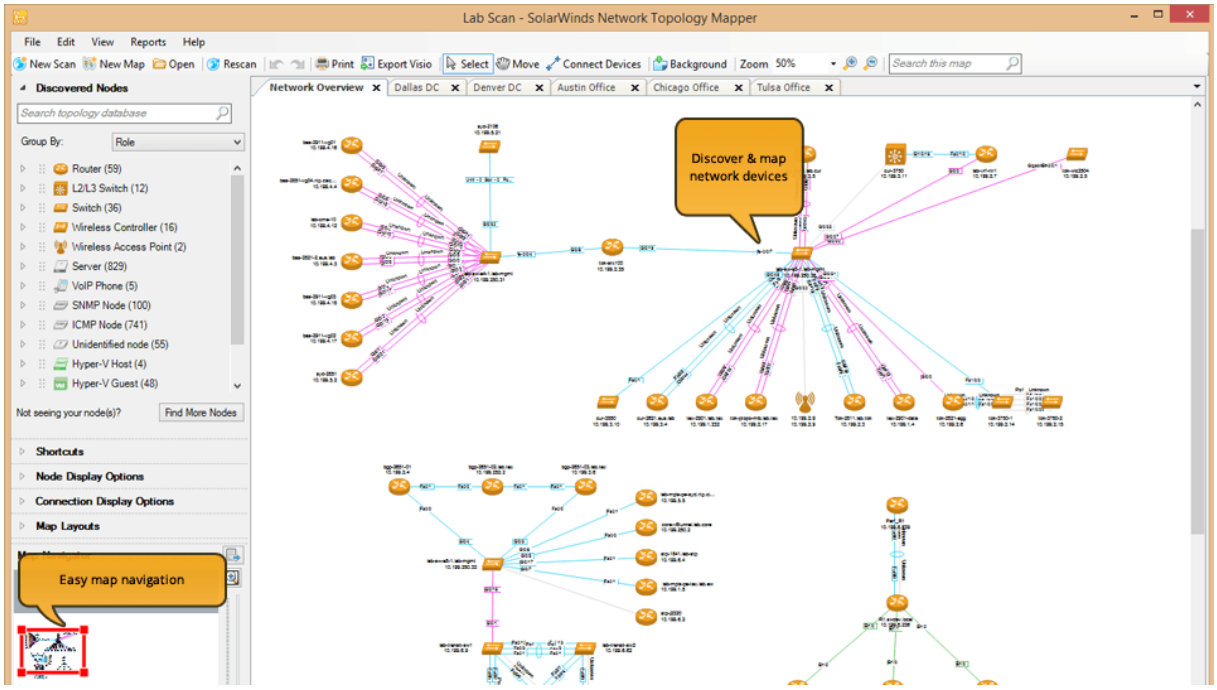

Abbildung 4: SolarWinds Network Topology Mapper [BV18]

Fazit Wie auch die beiden beschriebenen Lösungen, fokussieren sich bereits verfügbare Tools mehr auf Monitoring und Reporting. Es lassen sich einfach Netzwerkgraphen und Übersichten erstellen. Wenn ein Netzwerkgerät ausgewechselt wird, ändern sich die Informationen im System. Im Gegensatz zum Digital Twin werden auch Informationen zu den spezifischen Geräten gespeichert, wie Installationsdatum oder Patchlevel. Die Idee vom Digital Twin ist es jedoch eine Abbildung der Netzwerktopologie zu erstellen, die unabhängig von der Hardware ist und nur die Funktion des Geräts widerspiegelt.

4.2.6 Erwartetes Produkt

Im Vergleich mit den erwähnten kommerziellen Lösungen, fokussiert sich der Digital Twin auf die layerspezifisch korrekte Speicherung und Bereitstellung der Informationen. Funktionen wie die automatische Geräteerkennung, oder graphische Visualisierung des Netzwerks stehen nicht im Vordergrund. Das zu erwartende Produkt soll die aufgelisteten Use Cases abdecken und den funktionierenden Zustand eines Netzwerks repräsentieren.

4.3 Anforderungsspezifikation

Um, wie in der Analyse der Anforderungen beschrieben, den Zustand eines Netzwerks zu speichern, sind sehr viele Informationen von unzähligen Netzwerkprotokollen vonnöten. Das Endprodukt soll nicht sämtliche Netzwerktechnologien unterstützen, sondern sich auf die Wesentlichen fokussieren. In dieser Spezifikation wird definiert, welche einzelnen Protokolle pro Layer vom Endprodukt gespeichert werden können und wie diese Daten gespeichert werden.

4.3.1 Zu speichernde Informationen

Wie bereits in der Analyse erwähnt, wird das Netzwerk in einzelne Layer aufgeteilt. Auf jedem Layer arbeiten verschiedene Netzwerkprotokolle und Technologien.

Sublayer Viele dieser Technologien arbeiten auf einem spezifischen Layer im OSI Modell. Sie verändern die Netzwerktopologie und haben so Einfluss auf den Verlauf des Datenverkehrs in den überstehenden Layern. Damit dies sauber voneinander abstrahiert werden kann, wird jede Änderung der Netzwerktopologie oder eine Gruppe von Funktionalität in einen eigenen Sublayer unterteilt. Die Sublayer sind untereinander strikt geordnet. Sie sind abhängig vom darunterliegenden Netzwerklayer, auf welchem sie arbeiten. Als Beispiel kann der Sublayer „VLAN“ nur mit dem Layer 2 Protokoll „Ethernet“ verwendet werden. Zwischen „Ethernet“ und „VLAN“ können aber auch zusätzliche Sublayer wie „Link Aggregation“ existieren. Pro Sublayer können mehrere Protokolle abstrahiert werden, welche voneinander unabhängig sind. Im Sublayer „Link Aggregation“ können die Protokolle „LACP“ oder „EtherChannel“ beschrieben werden.

Sublayer Veranschaulichung Um das Konzept der Sublayer besser zu vermitteln, folgt ein Beispiel mit einem kleinen Netzwerk, mit dem die einzelnen Sublayer dargestellt werden. Dies ist nur als Veranschaulichung gedacht, es werden nicht alle geplanten Sublayer oder Informationen dargestellt.

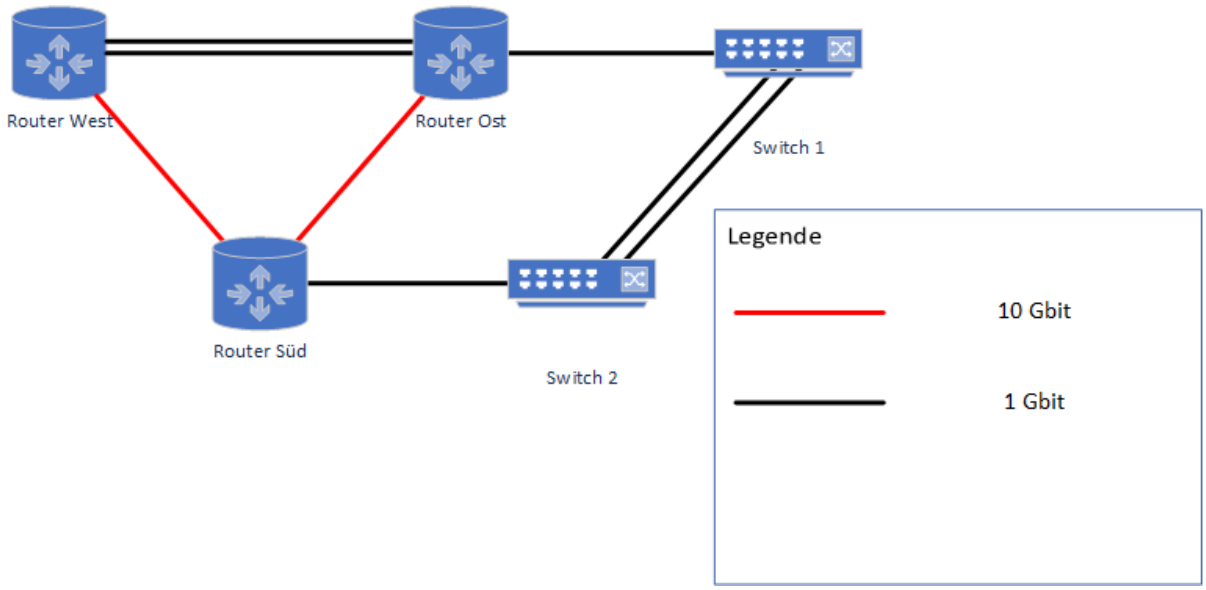

Abbildung 5: Layer 1 Beispieltopologie

Bei der Layer 1 Betrachtung wird die physikalische Verkabelung des Netzwerks gespeichert. Für jedes Gerät werden verbundene Interfaces mit Verbindungsinformationen gespeichert.

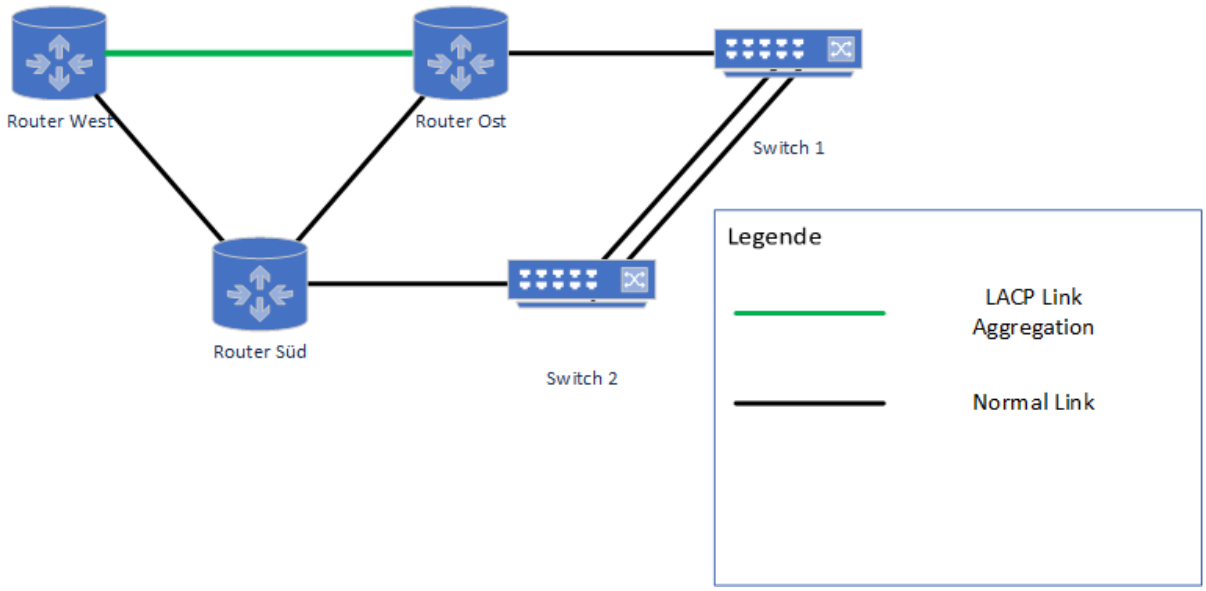

Abbildung 6: Layer 2 Topologie mit Link Aggregation

Aus Sicht von Layer 2 werden die Netzwerkinterfaces mit den entsprechenden MAC Adressen gespeichert. Wird in einer Topologie eine Layer 2 Technologie wie Link Aggregation verwendet, wird dies als Sublayer von Layer 2 im Digital Twin gespeichert. Kommt Spanning Tree zum Einsatz, werden auch diese Informationen als Sublayer von Layer 2 gespeichert.

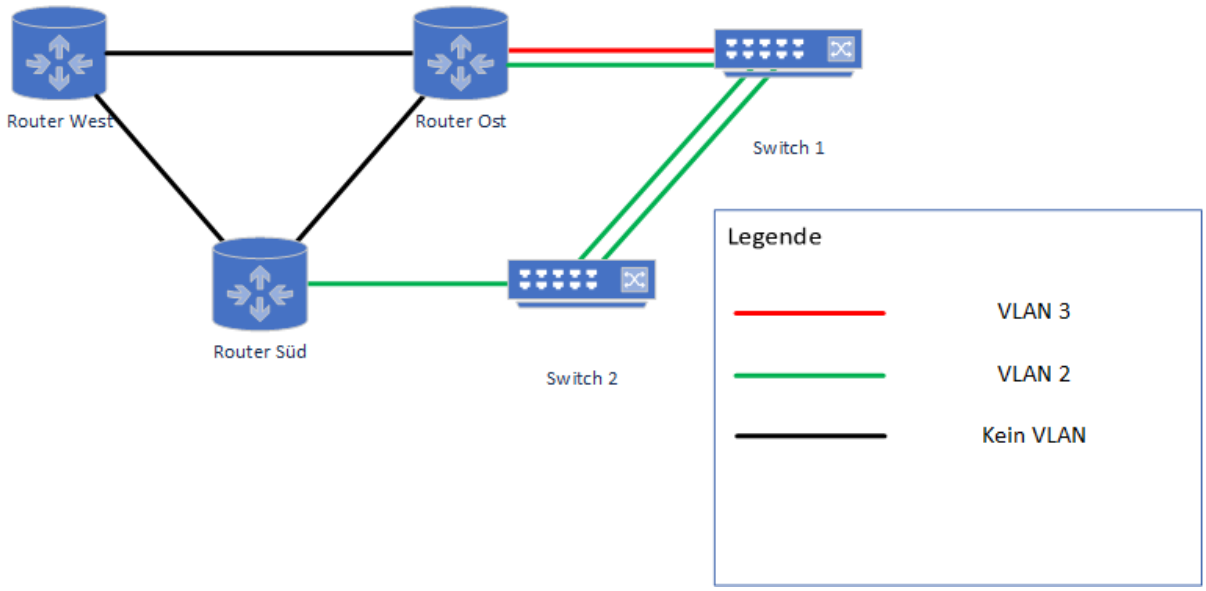

Abbildung 7: Layer 2 Topologie mit 2 VLAN's

Auch bei der Verwendung von VLAN's werden dessen Informationen eines Geräts als Sublayer von Layer 2 gespeichert.

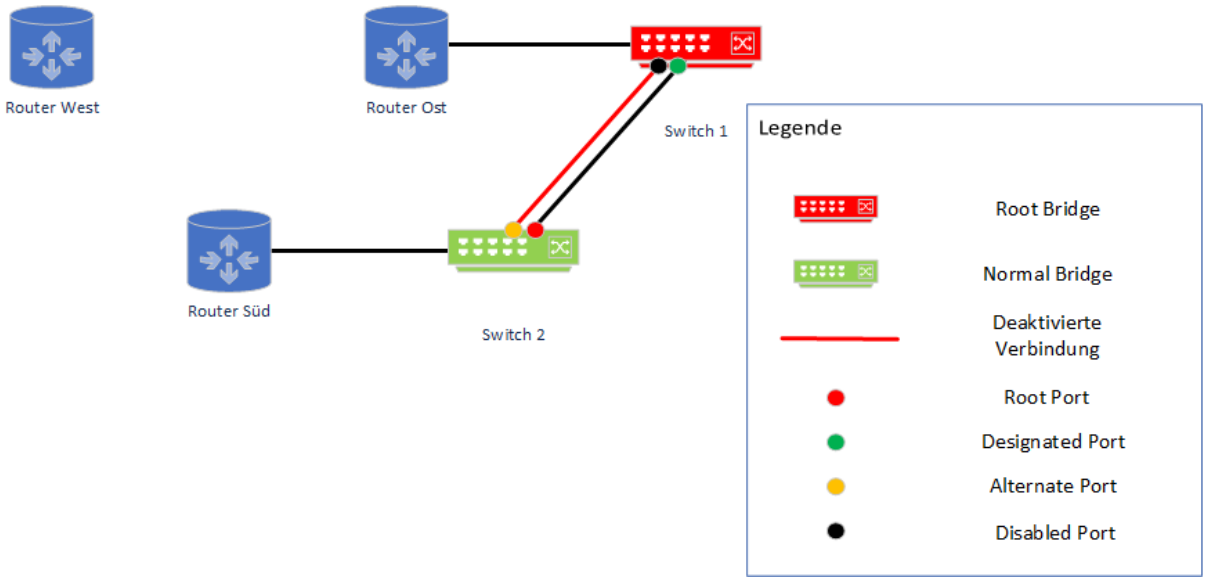

Abbildung 8: Layer 2 Topologie mit per VLAN Spanning Tree

Wird im Netzwerk per VLAN Spanning Tree verwendet (hier im VLAN 2), bilden auch diese Informationen einen Layer 2 Sublayer.

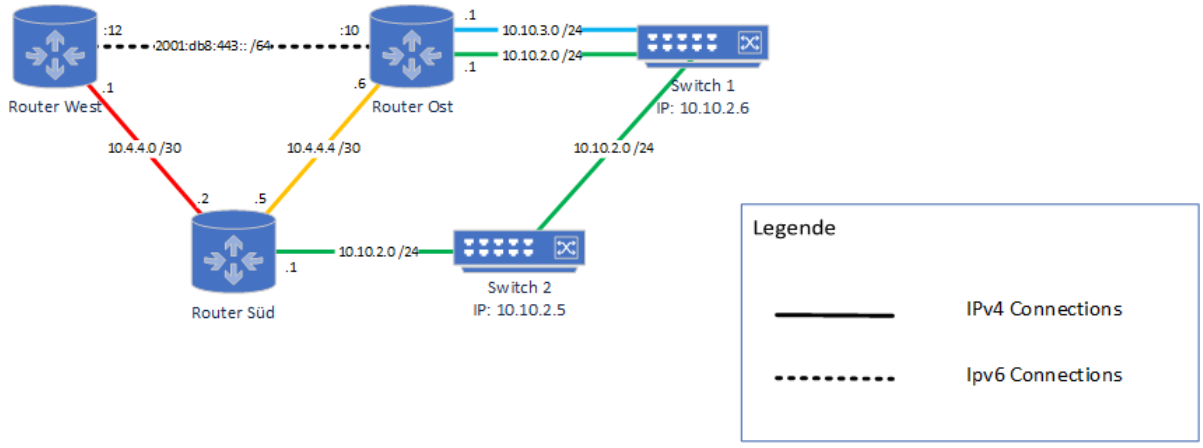

Abbildung 9: Layer 3 Darstellung mit visualisierten Subnetzen

IPv4 und IPv6 Informationen von Geräteinterfaces werden im Layer 3 gespeichert. Werden Funktionen wie NAT, VPN, ACL's, DHCP Server/Relay oder ICMP Konfigurationen verwendet, werden diese Informationen in entsprechenden Sublayern von Layer 3 gespeichert.

4.3.2 Auflistung Sublayer

Nebst den layerspezifischen Informationen, welche gespeichert werden sollen nachfolgend eine nicht finale Auflistung der bis jetzt eingeführten Sublayer abhängig der OSI Layer mit den unterstützten Protokollen:

- Layer 1 (Physical)
  Keine Sublayer vorhanden
- Layer 2 (Data Link)
  - Informationen zu Link Aggregation:
    LACP, EtherChannel
  - Spanning Tree
    RSTP, STP
  - VLAN
    IEEE 802.1Q
  - Per VLAN Spanning Tree
    Rapid PVST+
  - VPN
    PPPoE, L2TP
- Layer 3 (Network)
  - Routingtabelle
  - Routingprotokolle
    OSPF, EIGRP, BGP, RIP
  - VPN Konfiguration
    IPSec, GRE
  - NAT Konfigurationen
  - DHCP Funktionalität
  - Failover Funktionalität
    HSRP, VRRP

4.3.3 Benutzerinterface Anforderungen

Damit ein abgespeicherter Digital Twin mit den entsprechenden Informationen verständlich dargestellt werden kann, ist ein graphisches Benutzerinterface von Nöten. Die Visualisierung der Netzwerke wird mittels Webseite realisiert, welche via Webbrowser vom Benutzer erreicht werden kann.

Genauere Informationen über die Architektur sind in der Architekturübersicht einsehbar.

4.4 Generalisierung der Layer

Eine der wichtigsten Anforderungen an das Projekt ist die einfache Erweiterbarkeit. Die Technologien in Netzwerken verändern sich ständig und neue Technologien sollen dem Digital Twin einfach hinzugefügt werden können. Um das zu ermöglichen, wird wie bereits erwähnt, das Netzwerk in Layer und Sublayer aufgeteilt. Diese Layer und Sublayer sollen es ermöglichen, neue Technologien einfach dem Digital Twin „beizubringen“, indem für eine Neuerung ein neuer Sublayer definiert wird. Optimal wäre es, die Definitionen in ein externes File auszulagern. So könnte der Digital Twin einfach erweitert werden, ohne den Programmcode anfassen zu müssen. In diesem Abschnitt des Dokuments wird nun geklärt, ob es innerhalb der SA möglich ist dieses Feature umzusetzen.

Eine solche Auslagerung ist jedoch alles andere als trivial. Da der Entscheid hierüber grosse Auswirkungen auf das Projekt haben wird, widmet sich der gesamte folgende Abschnitt dieser Frage.

4.4.1 Voraussetzungen

Damit die Layer und Sublayer ausserhalb des Programmcodes definiert werden können, muss zuerst klar sein, was so ein Layer überhaupt können muss. Folgende Fragen müssen beantwortet werden:

- Was unterscheidet Layer und Sublayer voneinander?
- Wie beeinflusst ein (Sub)Layer den nächsten, falls überhaupt?
- Welche Abhängigkeiten existieren zwischen den (Sub)Layern?
- Welche Informationen kann ein Layer speichern / darstellen?

Um diesen Fragen beantworten zu können, werden Layer 1 und 2 mit den dazugehörigen Sublayer untersucht und dessen Anforderungen analysiert. Dazu wird zuerst identifiziert welche konkreten Informationen gespeichert werden müssen, damit der aktuelle Zustand des Netzwerkes korrekt gespeichert werden kann. Anhand von diesen Daten werden dann generelle Anforderungen an die Layer und Sublayer gestellt. Es werden nur Layer 1 und 2 analysiert,

4.4.2 Layer 1

Der Layer 1 ist die Basis des gesamten Netzwerkes, und definiert die grundsätzlichen Verbindungen im Netzwerk. Dies bedeutet, für Layer 1 existieren keine Layer darunter, welche potenziellen Einfluss auf diesen haben können. Für diesen Layer müssen nicht viele Informationen gespeichert werden. Lediglich Informationen über physikalische Geräte im Netzwerk und dessen Verbindungen sind vonnöten.

Für Layer 1 sollen folgende Informationen gespeichert werden:

- Name einer Netzwerkkomponente
- Verbindungen zwischen den Netzwerkkomponenten
- Portname der Verbindung bei beiden angeschlossenen Komponenten
- Geschwindigkeit und Duplexmodus jeder Verbindung

Daraus entstehen folgende generischen Anforderungen an die Architektur eines Layers:

- Ein Layer besitzt einen Namen und mehrere Eigenschaften
- Die Layer sind untereinander strikt geordnet
- Ein Layer speichert Netzwerkkomponenten (nachfolgend Netzwerkknoten genannt)
- Diese Netzwerkkomponenten sind untereinander mit Kanten verbunden
- Die Endpunkte der Kanten, an denen sie mit Netzwerkknoten verbinden, sind als Interfaces gespeichert
- Diese Netzwerkknoten, Kanten und Interfaces können beliebig viele Eigenschaften haben

4.4.3 Layer 2

Der zweite Layer ist der erste, welcher Abhängigkeiten auf einen unterliegenden Layer besitzt. Layer 1 gibt ihm seine Grundstruktur vor. Soll heissen, es kann keine Verbindung entstehen, welche auf Layer 1 nicht vorhanden ist. Folgende Informationen sollen für Layer 2 gespeichert werden:

- Kommunikationsprotokoll (Ethernet)
- MAC-Adressen der einzelnen Interfaces von Layer 1
- Abgrenzung von Broadcast-Domänen

Daraus entstehen folgende Anforderungen an die Architektur eines Layers:

- Ein neuer Layer kann sämtliche Einheiten des unteren Layers kopieren und neue Eigenschaften festhalten. Die alten Eigenschaften werden nicht mitkopiert. Es soll für jede Einheit bekannt sein, zu welcher Einheit sie im unteren Layer gehört.
- Einheiten können gruppiert werden (mehrere Kanten in einer Broadcast-Domain)

Sublayer Port Link Aggregation Port Link Aggregation bildet der erste Sublayer des zweiten Layers. Mittels dieser Technologie können zwei oder mehr physikalische Verbindungen zu einer logischen zusammengefasst werden. Dadurch werden auch die Interfaces zusammengefasst, was bedeutet das mehrere Interfaces aus Layer 1 zu einem neuen werden. Folgende Informationen werden für Port Link Aggregation benötigt:

- Welche Technologie wird verwendet? (LACP, EtherChannel)
- Welche Verbindungen werden gebündelt?
- Was für eine Geschwindigkeit besitzt die neue Kante?
- Was für Interfaces besitzt die neue Kante?

Daraus entstehen folgende neuen Anforderungen an die Architektur eines Sublayers:

- Ein Layer kann mehrere Sublayer besitzen. Diese Sublayer sind wie die Layer, strikt geordnet.
- Neue Einheiten können aus mehreren Einheiten des unteren Layers zusammengefasst werden. Auf der neuen Einheit ist dann ersichtlich aus welchen Einheiten sie entstanden ist. Diese neue Einheit hat ebenfalls eigene Eigenschaften.
- Werden zwei Kanten zusammengefasst, werden auch die zugehörigen Interfaces zusammengefasst und es entsteht ein neues Interface für die neue Kante.
- Es kann auch Einheiten geben, welche von einem Sublayer unangetastet bleiben. Diese werden dennoch vom unteren Layer übernommen, haben allerdings keine eigenen Eigenschaften (abgesehen von der Referenz auf die untere Einheit).

Spanning-Tree Mit Spanning-Tree werden redundante Links deaktiviert, um einen Netzwerkloop zu verhindern. Dazu werden bestimmte Links deaktiviert und im Fehlerfall wieder aktiviert. Für Spanning-Tree werden folgende Informationen benötigt:

- Welches ist die Root-Bridge?
- Was ist die konfigurierte Bridge-ID?
- Was sind die momentanen Zustände der Ports?
- Welche Links werden deaktiviert?

Daraus entstehen folgende zusätzlichen Anforderungen an die Architektur eines Sublayers:

- Einheiten können aus einem Sublayer entfernt werden.
- Der Zustand eines Interfaces beeinflusst den Zustand der Kante. (Blocking Port führt zu entfernen einer Kante)

VLAN Durch VLAN kann man ein physikalischen Netzwerk in mehrere logische aufteilen. Über eine physikalische Verbindung können mehrere logische VLAN-Verbindungen verlaufen. VLAN bringt folgende Informationen mit:

- Welche VLANs sind auf einem Netzwerkknoten gespeichert?
- Welche VLANs sind auf einem Interface getaggt?
- Welches ist das Default-VLAN einer Verbindung?

Daraus entstehen folgende zusätzliche Anforderungen für die Architektur der Sublayer:

- Speichern von mehreren Einheiten, die alle dieselbe Ursprungseinheit im unteren Layer haben. (in diesem Fall Kanten)
- Mehrere Kanten können mit dem gleichen Interface verbunden sein.

Per VLAN Spanning-Tree Per VLAN Spanning-Tree ist einfach das Spanning-Tree Protokoll innerhalb eines VLANs angewendet. Es funktioniert identisch wie Spanning-Tree ausserhalb von VLANs, nur gibt es für jedes VLAN einen eigenen Spanning-Tree. Folgende Informationen werden zusätzlich zu jenen des normalen Spanning-Trees benötigt:

- Auf welchen VLANs arbeitet dieser Spanning-Tree?
- Welche Verbindungen sind in diesem VLAN überhaupt relevant?

Folglich entstehen zusätzliche Anforderungen für die Architektur der Sublayer:

- Einheiten können komplett aus einem Sublayer ausgeblendet werden, sofern sie nicht relevant sind.(Geräte die nicht im VLAN sind)
- Es können mehrere Instanzen eines Sublayers gleichzeitig vorhanden sein. (Für jedes VLAN wird ein eigener Sublayer vonnöten sein).

Layer 2 VPNs Mit Layer 2 VPN Protokollen wie zum Beispiel L2TPoverIPSec können auf Layer 2 zusätzliche Verbindungen zwischen zwei Geräten hergestellt werden, die physikalisch nicht verbunden sind. Dass der Transport von L2TP über Layer 3 erfolgt, spielt hier keine Rolle, denn die Auswirkungen von L2TP sind schlussendlich im Layer 2 zu finden. Für ein Layer 2 VPN müssen folgende Informationen gespeichert werden:

- Welche Netzwerkkomponenten werden durch den Tunnel verbunden?
- Was für eine Verschlüsselung wird verwendet?

Dies bringt folgende zusätzliche Architekturanforderungen für einen Sublayer:

- Es können Einheiten hinzugefügt werden, die keine Verknüpfung zu Einheiten auf unteren Layern haben.
- Einheiten können sich aus anderen Einheiten von verschiedenen Geräten zusammenfügen, die aber immer den gleichen Typ besitzen müssen (z.B. Kanten können nur auf Kanten basieren). Diese anderen Einheiten können auch auf höheren Sublayern existieren, allerdings nur alle auf demselben.

Die zweite Anforderung ist für den Fall, dass man den Pfad des VPNs über Layer 3 aufzeigen will.

4.4.4 Zusammenfassung Architekturanforderungen Layer

Aus der Analyse der Layer sind einige Anforderungen entsprungen. Um die Layer ausserhalb des Programms definieren zu können, müssen sie alle diese Punkte erfüllen. In der Definition der Layer muss aber nicht nur ersichtlich sein, welche Informationen er beinhaltet und wie diese Netzwerkknoten, Kanten und Interfaces zugewiesen werden, sondern auch wie der Digital Twin an diese Information herankommt. Um eine bessere Übersicht zu gewinnen, werden die Anforderungen gruppiert nach Meta-Anforderungen die nicht von Netzwerkkomponenten abgerufen werden können, simplen Anforderungen die nur den Abruf von Daten erfordern und erweiterte Anforderungen, welche Entscheidungen benötigen.

Meta-Anforderungen:

- Ein Layer/Sublayer besitzt einen Namen und mehrere Eigenschaften.
- Die Layer/Sublayer sind untereinander strikt geordnet.
- Ein Layer kann mehrere Sublayer besitzen. Diese Sublayer sind wie die Layer, strikt geordnet.

Simple Anforderungen:

- Ein Layer speichert Netzwerkknoten.
- Diese Netzwerkknoten sind untereinander mit Kanten verbunden.
- Die Endpunkte der Kanten, an denen sie mit Netzwerkknoten verbinden, sind als Interfaces gespeichert.
- Diese Einheiten können beliebig viele Eigenschaften haben.
- Ein neuer Layer kann sämtliche Einheiten des unteren Layers einsehen und ihnen mit neuen Eigenschaften ergänzen. Die alten Eigenschaften werden nicht mitkopiert. Es soll für jede Einheit bekannt sein, zu welcher Einheit sie im unteren Layer gehört.
- Es können Einheiten hinzugefügt werden, die keine Verknüpfung zu Einheiten auf unteren Layern haben.
- Es kann auch Einheiten geben, die von einem Sublayer unangetastet bleiben. Diese werden dennoch vom unteren Layer kopiert, haben allerdings keine eigenen Eigenschaften (abgesehen von der Referenz auf die untere Einheit).

Erweiterte Anforderungen:

- Neue Einheiten können aus mehreren Einheiten des unteren Layers zusammengefasst werden. Auf der neuen Einheit ist dann ersichtlich aus welchen Einheiten sie entstanden ist. Diese neue Einheit hat ebenfalls eigene Eigenschaften.
- Einheiten können Gruppiert werden (mehrere Kanten in einer Broadcast-Domain).
- Werden zwei Kanten zusammengefasst, werden auch die zugehörigen Interfaces zusammengefasst und es entsteht ein neues Interface für die neue Kante.

- Einheiten können aus einem Sublayer entfernt werden.
- Der Zustand eines Interfaces beeinflusst den Zustand der Kante. (Blocking Port führt zu entfernen einer Kante)
- Einheiten können sich aus anderen Einheiten von verschiedenen Geräten zusammenfügen, die aber immer den gleichen Typ besitzen müssen (z.B. Kanten können nur auf Kanten basieren). Diese anderen Einheiten können auch auf höheren Sublayern existieren, allerdings nur alle auf demselben.

4.4.5 Fazit

Auswertung der Fragen Die zu Beginn gestellten Fragen können nun wie folgt beantwortet werden:

- Was unterscheidet Layer und Sublayer voneinander?

  Grundsätzlich gibt es keine grossen Unterschiede zwischen Layer und Sublayer. Die Layer sind die spezifischen OSI-Layern, Sublayer sind Gruppierungen von Technologien, die zu diesen Layern gehören. Layer und Sublayer untereinander unterscheiden sich vor allem darin, dass sie andere Technologien repräsentieren.
  Layer untereinander unterscheiden sich in drei Eigenschaften: verwendete Technologie, gespeicherte Informationen und Netzwerktopologie. Manche Layer ändern nichts an der Topologie, sondern fügen nur neue Informationen hinzu, andere ändern die Topologie ziemlich stark. Ein Beispiel für Informationen ist beispielsweise die Geschwindigkeit einer Layer1 Verbindung. Die Technologie des Layers gibt an auf welchem Layer oder Sublayer er basiert, wie beispielsweise per VLAN STP auf VLAN aufbaut.

- Wie beeinflusst ein (Sub)Layer den nächsten, falls überhaupt?

  Ein Layer gibt seinem Nachfolger die Topologie vor, heisst seine Änderungen an der Topologie werden an alle weiteren Layer weitergegeben. Die verwendete Technologie des Layers oder die Informationen, die ein Layer speichert spielen für den nächsten Layer keine Rolle.

- Welche Abhängigkeiten existieren zwischen den (Sub)Layern?

  Zwischen den einzelnen Layern bestehen nur Abhängigkeiten in der Topologie. Ein Sublayer wie BGP der zum Layer3 gehört, kann nicht direkt auf Layer1 aufsetzen, da die Topologie durch die ganzen Layer2 Technologien stark verändert wird.

- Welche Informationen kann ein Layer speichern / darstellen?

  Diese Frage ist sehr Layerspezifisch, aber grundsätzlich können alle Informationen entweder zu Netzwerkknoten, Interfaces oder Verbindungen zugeordnet werden.

Wenn wir die Fragen vom Anfang anschauen, sehen wir, dass drei Informationen für einen Layer vonnöten sind: Technologie, Informationen und Netzwerktopologie. Die Daten zur verwendeten Technologie (z.B. VLAN) können ohne Probleme ausserhalb des Programms definiert und dann einfach eingelesen werden. Die Informationen des Layers sind schon ein bisschen trickreicher, es muss definiert werden wie die Informationen vom Netzwerkgerät abgerufen werden können. In der konkreten Implementation werden das Restconf-URLs sein. Die Topologie ist nochmals ein Schritt komplizierter, da hier nicht einfach Informationen abgerufen werden können, sondern es muss Logik eingebaut werden um die Änderungen

an der Topologie zu erkennen. Als Beispiel, wenn bei Spanning-Tree ein Port im Status „Blocking“ ist, kann die angrenzende Verbindung für weitere Layer nicht mehr verwendet werden.

Entscheid Die Möglichkeit, Layers ausserhalb des Programmes zu definieren klingt verlockend, so könnten ganz einfach neue Netzwerktechnologien zum Digital Twin hinzugefügt werden. Das Problem ist, dass eine Technologie die grosse Änderungen an der Topologie bewirkt, wie Spanning-Tree sehr viel Logik mit sich bringt. Wenn man diese Layer in einem File extern vom Digital Twin definieren würde, müsste man auch diese Logik in das File schreiben. Dies erfordert entweder das Entwickeln einer eigenen Sprache, eine *Domain Specific Language* oder das Verwenden einer bereits etablierten Sprache. Zusätzlich muss der Digital Twin diesen Code auch interpretieren können, was seine Komplexität immens erhöht. Aus diesen Grund haben wir uns gegen die Definition von Layern ausserhalb des Digital Twins entschieden. Wenn sowieso mit einer Art von Code gearbeitet werden muss, kann auch genauso gut im Code der Applikation selbst gearbeitet werden.

Das Konzept der Layer und Sublayer wird beibehalten, allerdings nicht generalisiert, heisst sie müssen direkt im Code des Digital Twin definiert werden. Der Fokus liegt stattessen darauf das Erstellen neuer Sublayer möglichst einfach zu gestalten. Ruby on Rails ist dabei eine grosse Hilfe, da mit Ruby als interpretierte Sprache der Digital Twin bei Änderungen am Code nicht neu kompiliert werden muss.

5 Architektur

5.1 Architekturübersicht

Die Architektur des Digital Twin wird unterteilt in Webserver, Datenbank, Webbrowser und das Netzwerk, welches erfasst werden soll. Die einzelnen Komponenten sind unabhängig voneinander und werden auch mit verschiedenen Technologien realisiert. Das folgende Bild zeigt den grob strukturierten Aufbau der Softwarelösung. Nachfolgend werden die einzelnen Komponenten der Architektur kurz aufgelistet und beschrieben.

5.1.1 Webserver

Der Webserver bildet das Herzstück der Applikation und beinhaltet den Grossteil der Logik. Er kümmert sich um das Abspeichern der Daten in der Datenbank, stellt diese per Website dem Benutzer dar und bietet die Möglichkeit, diese zu bearbeiten. Ebenfalls kommuniziert er mit Netzwerkkomponenten im Netzwerk und wandelt dessen Informationen um, sodass sie in der Datenbank gespeichert werden können.

5.1.2 Datenbank

Die Datenbank ist die simpelste Komponente des Digital Twin. Sie wird vom Webserver mittels Schnittstelle angesprochen. Ihre Aufgabe ist das persistente Speichern von Daten für die Applikation.

5.1.3 Webbrowser

Der Webbrowser stellt das Bindeglied zwischen Benutzer und Applikation dar. Er ist auf dem Benutzergerät installiert und kommuniziert mit dem Webserver. Der Browser stellt die entsprechenden Informationen dar und bietet die Möglichkeit, diese zu verändern.

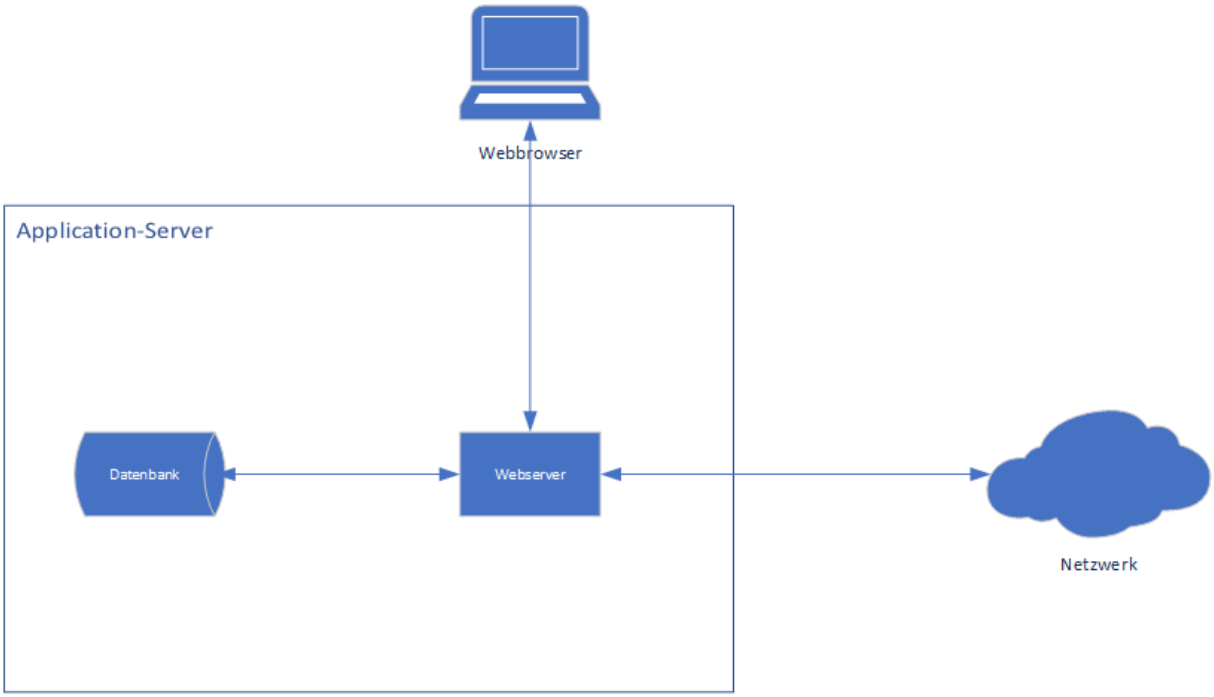

Abbildung 10: Architekturdiagramm ohne Technologieentscheidungen

5.1.4 Netzwerk

Das Netzwerk ist nicht direkter Bestandteil der Software. Einzelne Komponenten des Netzwerks werden jedoch vom Webserver mittels Schnittstelle angesprochen, um entsprechende Informationen zu abrufen.

5.2 Technologieentscheidungen

Für die zahlreichen Aufgaben, die im Rahmen dieser SA 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.

5.2.1 Webserver

Damit die Umsetzung des Webservers vereinfacht werden kann, soll dafür ein Webframework verwendet werden. Da der Webserver ein Grossteil der Logik beinhalten und mit den Netzwerkkomponenten kommunizieren wird, spricht dies für die Wahl eines serverseitigen Frameworks, anstelle eines clientseitigen Frameworks. Als mögliche Kandidaten wurden Django und Ruby on Rails genauer unter die Lupe genommen. Beides sind serverseitige Webframeworks, welche in einer Interpretersprache entwickelt wurden (Django in Python, Ruby on Rails in Ruby). Das hat den Vorteil, dass bei Änderungen am Code nicht das komplette Projekt neu kompiliert werden muss.

Nach entsprechender Recherche haben wir uns schlussendlich für Ruby on Rails entschieden. Ruby on Rails (kurz Rails) ist ein serverseitiges Webframework, welches in der Programmiersprache Ruby geschrieben wurde. Grundsätzlich kann Ruby auf verschiedenen Plattformen (Windows, Linux, MacOS) verwendet werden, es bestehen jedoch Einschränkungen unter Windows. Daher wird sich der Fokus dieser SA auf die Unterstützung von Unix-basierten Systemen (Linux, MacOS) beschränken.

Für die Kommunikation mit dem Datenbankserver wird eine Bibliothek namens Neo4jrb (mehr dazu im Nächsten Abschnitt) verwendet, welche die HTTP(S) Schnittstelle der Datenbank für die Kommunikation verwendet.

Gründe für die Wahl von Ruby on Rails sind:

- Rails erfüllt die Funktionsanforderungen der Gesamtarchitektur und ist kostenlos
- Kleinerer Zeitaufwand für Einarbeitung durch Vorkenntnisse im Team
- Gute Dokumentation und Support Community durch grosse Popularität
- Kein grosses Zeitbudget für Recherche von Alternativen
- Gute Integration mit verwendeter Datenbanktechnologie (OGM für Ruby)

5.2.2 Datenbank

Für die persistente Speicherung der Informationen wird eine Datenbank verwendet. Zu Beginn stellte sich die Frage nach der Art der Datenbank. Zur Auswahl standen traditionelle SQL Datenbanken, wie zum Beispiel PostgreSQL, oder sogenannte NoSQL Datenbanken.

Hauptunterschied zwischen diesen beiden Arten ist die Strukturierung der Daten im System. Traditionelle SQL Datenbanken speichern die Informationen in Tabellen, welche untereinander verknüpft sind. NoSQL Datenbanken speichern die Informationen in Formaten wie Graphen, Key-Value Stores, Document Stores und mehr.

Je nach Struktur der Daten, welche gespeichert werden sollen, spielt die Datenbankstruktur eine wichtige Rolle. Wird eine Datenbank mit ungünstiger Struktur verwendet, kann sich dies negativ auf die Softwarearchitektur und Performance der Applikation auswirken.

Durch einen Vorschlag von Urs Baumann, einer der Projektbetreuer, sind wir auf Neo4j aufmerksam gemacht worden. Neo4j ist ein Graph Datenbank Management System entwickelt von Neo4j Inc., welches

als Community Edition kostenlos zur Verfügung steht.

Da eine Netzwerktopologie grundsätzlich ein Graph mit Knoten und Kanten ist, macht es Sinn, diese in einer Graph Datenbank zu speichern. Nach entsprechender Recherche und Abgleich mit den Anforderungen an die Datenbank haben wir uns für Neo4j als Datenbanklösung entschieden.

Neo4j lässt sich zudem gut mit einer Ruby Applikation kombinierten. Neo4jrb ist eine Bibliothek für Ruby, welche ein Object Graph Mapping (OGM) zur Verfügung stellt. Dadurch können Ruby Objekte einfacher in der Datenbank gespeichert und abgerufen werden, ohne Datenbankqueries von Hand zu schreiben.

Dadurch, dass Neo4j Neuland für beide Teammitglieder ist, bringt dies auch Risiken mit sich. Die Einarbeitung benötigt mehr Zeit und es können unerwartete Schwierigkeiten auftreten. Trotzdem sind wir der Meinung, dass es die richtige Lösung ist und bietet uns die Möglichkeit uns eine neue Technologie kennen zu lernen.

Gründe für die Wahl von Neo4j sind:

- Optimale Struktur für Datenspeicherung
- Kostenlos verfügbar und populärste Art von Graph Datenbank
- Neo4jrb für Ruby
- Neue Technologie lernen

5.2.3 Webbrowser

Für den Webbrowser gibt es grundsätzlich keinen Technologieentscheid, da dieser vom Benutzer auf seinem Gerät installiert wird und er bei dessen Wahl frei ist. Im Rahmen dieser SA beschränkt sich jedoch die Auswahl von Unterstützten Browsern auf die neusten Versionen von Mozilla Firefox und Google Chrome.

Für die Kommunikation mit dem Webserver wird dessen HTTP(S) Schnittstelle verwendet.

5.2.4 Netzwerk

Wie bereits erwähnt, ist das Netzwerk kein direkter Bestandteil der Architektur, sondern eine externe Schnittstelle. Damit der Webserver korrekt mit Geräten im Netzwerk kommunizieren kann, muss jedoch die Art der Kommunikation mit dem Netzwerk klar definiert sein. Dazu ist zudem eine möglichst grosse Kompatibilität mit verschiedenen Geräten gewünscht.

Die Art der Kommunikation zwischen dem Webserver und den Netzwerkkomponenten ist entscheidend, da nicht alle Protokolle gleich viele Informationen anbieten und die Konvertierung der Daten in brauchbare Objekte unterschiedlich verläuft.

Nachfolgend werden drei Optionen kurz erläutert und mit einem Fazit die Technologiewahl kommuniziert.

SSH Der direkteste Ansatz ist eine Kommunikation über SSH auf die Konsole des Netzwerkgerätes. Ein grosser Vorteil ist sicher, dass über diese Konsole sämtliche Informationen eines Gerätes abrufbar sind. Mit C.R.A.SSH. oder Netmiko sind auch Bibliotheken verfügbar, welche die Kommunikation erleichtern können. Das Hauptproblem bei der Verwendung von SSH ist die Hardwareunabhängigkeit. Von Gerät zu Gerät können Befehle und Antworten unterschiedlich sein, eventuell sogar abhängig von der installierten Gerätesoftware. Unterschiede zwischen verschiedenen Herstellern sind noch grösser. Zudem muss der Output der Konsole zuerst in ein brauchbares Objektformat geparst werden, was wieder sehr abhängig von Gerät und Softwarestand ist.

NETCONF NETCONF wurde Ende 2006 als RFC4741 veröffentlicht. Es wurde entwickelt um die Konfiguration von Netzwerkkomponenten zu erleichtern, welche damals alle per Kommandozeile automatisch konfiguriert wurden. Dies brachte einige Probleme mit sich, wie zum Beispiel fehlendes Error-Handling und starke Abhängigkeit vom Hersteller. NETCONF baut auf dem Remote Procedure Call (RPC) Protokoll auf und verwendet für den Austausch von Daten XML. Als Transportprotokoll stehen verschiedene Varianten zur Verfügung, unter anderem auch SSH. Mit YANG (Yet Another New Generation) werden die Datenmodelle und Befehle von NETCONF definiert. Es gibt spezifische YANG-Models von Herstellern der Netzwerkgeräte, oder auch frei verfügbare. Durch diese freien YANG-Modelle kann NETCONF Geräteunabhängig arbeiten.

RESTCONF RESTCONF ist eine REST-like API, welche über HTTP ansteuerbar ist. Sie wurde als alternative Variante von NETCONF entwickelt, um die Kommunikation zu erleichtern. Genauso wie NETCONF verwendet RESTCONF auch YANG und XML, bietet allerdings auch noch JSON als Alternative zu XML. Durch die Verwendung von HTTP ist RESTCONF von jeder Software ansprechbar die HTTP-Requests unterstützt.

Fazit Netzwerkkommunikation SSH steht als Kommunikationsmöglichkeit ausser Frage, da der Digital Twin möglichst Geräteunabhängig werden soll und SSH stark abhängig vom Gerät ist. Im Aspekt Geräteunabhängigkeit ist RESTCONF ideal, da es von mehreren Herstellern verwendet wird. Das Problem an NETCONF ist der Aufwand für die Kommunikation. Für jeden Request muss ein XML erstellt und per SSH gesendet werden. RESTCONF hebelt dies aus, da die Requests bequem per HTTP gesendet werden können, es wird also keine zusätzliche SSH Verbindung benötigt. Anstatt von XML kann JSON verwendet werden, was die Kommunikation nochmals erleichtert. Aus diesen Gründen haben wird uns für RESTCONF als Kommunikationsart entschieden.

5.2.5 Fazit Technologieentscheidungen

Schlussendlich sind wir zu folgendem Entschluss gekommen:

Der Webserver wird als Ruby on Rails Applikation entwickelt, welche eine Neo4j Datenbank mittels HTTP(S) Schnittstelle anspricht. Die Kommunikation zwischen Webserver und Netzwerkkomponenten wird mittels RESTCONF stattfinden. Der Benutzer interagiert mit der Software mittels Webbrowser, welcher ebenfalls mittels HTTP(S) Schnittstelle mit dem Webserver kommuniziert.

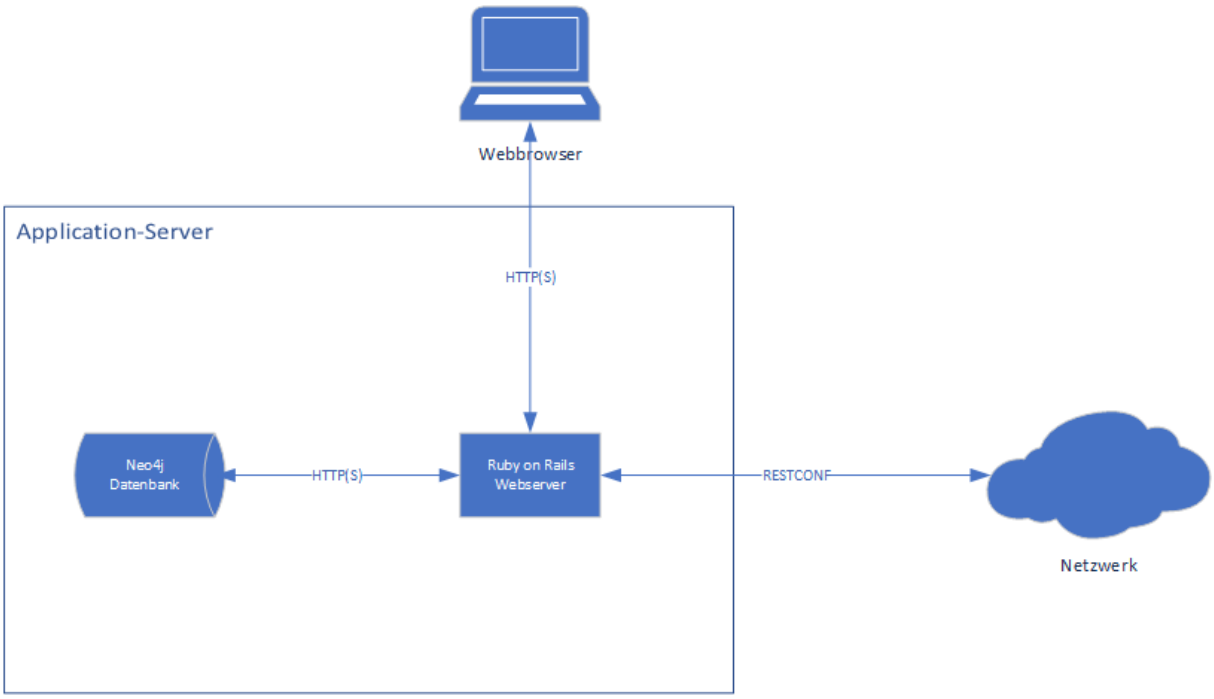

Abbildung 11: Architekturdiagramm mit Technologieentscheidungen

5.3 Layer

Im folgenden Abschnitt werden die Layer und Sublayer, sowie die darin enthaltenen Informationen definiert. Im Vergleich zu der vorherig dokumentierten Analyse, werden im folgenden Abschnitt einige unterschiede auftauchen. Dies führt daher, dass bei der definitiven Erstellung der Architektur Anpassungen vorgenommen wurden.

Die Informationen eines (Sub)Layers werden in folgende Gruppen eingeteilt, welche schlussendlich auch für die Strukturierung der Daten in der Datenbank verwendet werden:

- Informationen über einen Knoten
- Informationen über ein Interface
- Informationen über eine Kante

Die zu speichernden Informationen werden nach folgendem Schema aufgelistet:

Information	**Beispieldaten**

Die erste Spalte der Tabelle stellt die zu speichernden Informationen dar. In der zweiten Spalte werden Beispieldaten aufgelistet. Diese dienen hier lediglich zur Veranschaulichung und sollen keine abschliessende Liste von möglichen Werten darstellen.

5.3.1 Layer 1

Entspricht dem ersten Layer entsprechend dem OSI Modell und speichert grundlegende Informationen über physische Einheiten.

Informationen über einen Knoten

Art des Geräts	Layer2 Switch, Router
Geräteidentifier	Hostname

Informationen über ein Interface

Interface-State	Up, Down
Line Protocol State	Up, Down
Interface Identifier	1/0/1
Art des Interface	FastEthernet, GigabitEthernet, Serial

Informationen über eine Kante

Media Typ	1000BaseT
Duplex Mode	Full, Half
Übertragungsprotokoll	Ethernet
Medium Casttyp	Broadcast, Unicast

5.3.2 Layer 2

Enthält Informationen über den zweiten Layer entsprechend dem OSI Modell. Wird bei einem Interface Ethernet verwendet, wird zwischen Switchport und Routerport differenziert. Ein Switchport wird auf einem höheren Layer keine IP Adresse besitzen. Einem Routerport kann auf einem höheren Layer eine IP Adresse zugewiesen werden. Das Managementinterface eines Switchs entsprechend als Routerport repräsentiert.

Informationen über einen Knoten

Keine	

Informationen über ein Interface

Konfigurationsart des Ports	Switchport, Routerport
Ist Managementinterface?	Ja, Nein
Übertragungsprotokoll	Ethernet

Informationen über eine Kante

MTU	1500 Bytes
Bandwidth	1000000 Kbit/s

5.3.3 Layer 2 Sublayer: Link Aggregation

Speichert Informationen bei der Verwendung von Link Aggregation. Interfaces und Kanten aus unterliegenden Layern werden zusammengefasst und bilden neue Interfaces und Kanten. Informationen über verwendete Aggregations-Technologien werden ebenfalls erfasst.

Informationen über einen Knoten

Keine	

Informationen über ein Interface

Beteiligte Interfaces	Referenz auf unterliegende Interfaces
Verwendete Technologie	LACP, EtherChannel

Informationen über eine Kante

Beteiligte Kanten	Referenz auf unterliegende Kanten

5.3.4 Layer 2 Sublayer: MAC

Dieser Layer speichert hauptsächlich die MAC Adressen der Interfaces. Eigentlich könnte diese Information auch in den Layer 2 integriert werden. Um eine spätere Erweiterung zu vereinfachen, wird diese Information jedoch in einen eigenen Sublayer ausgelagert.

Informationen über einen Knoten

Keine	

Informationen über ein Interface

MAC Adresse	5C:A4:8A:0D:70:58

Informationen über eine Kante

Keine	

5.3.5 Layer 2 Sublayer: Spanning Tree

In diesem Sublayer werden Informationen über verwendetes Spanning Tree gespeichert. Wird durch das Spanning Tree Protokoll ein Interface auf einem Switch deaktiviert und somit eine Verbindung „getrennt“, wird diese Verbindung trotzdem in diesem Sublayer erfasst. Der Grund dafür ist, dass an dieser Verbindung weitere, dem System nicht bekannte, Hosts angeschlossen sein könnten.

Informationen über einen Knoten

Spanning Tree Version	STP, RSTP
Bridge ID	32769.5ca4.8a0d.7058
Root Bridge?	Ja, Nein

Informationen über ein Interface

Portstatus	Root Port, Designated Port, Non Designated, Alternate Port
Port ID (Prio.Port)	128.25
Root Path Cost	20

Informationen über eine Kante

Keine	

5.3.6 Layer 2 Sublayer: VLAN

Speichert Informationen über verwendete VLANs. Im Vergleich mit den vorherigen Sublayern besitzt dieser eine spezielle Eigenschaft. Anders an diesem Sublayer ist, dass dieser mehrfach Vorhanden sein kann. Denn pro verwendetes VLAN wird jeweils eine neue Sublayerinstanz erstellt, welche nur die Topologie des jeweiligen VLANs beinhaltet.

Informationen über einen Knoten

MTU	1500 Bytes

Informationen über ein Interface

VLAN ID	1002
Getagged?	Ja, Nein

Informationen über eine Kante

Keine	

5.3.7 Layer 2 Sublayer: Per VLAN Spanning Tree

In diesem Sublayer werden Informationen bezüglich Spanning Tree innerhalb eines VLANS gespeichert. Wie bereits beim VLAN Sublayer, kann auch dieser Sublayer mehrfach existieren. Für jeden per VLAN Spanning Tree Sublayer besteht ein VLAN Sublayer, welcher referenziert wird.

Informationen über einen Knoten

Bridge ID	32769.5ca4.8a0d.7058
Root Bridge?	Ja, Nein

Informationen über ein Interface

Portstatus	Root Port, Designated Port, Non Designated, Alternate Port
Port ID (Prio.Port)	128.25
Root Path Cost	20

Informationen über eine Kante

Keine	

5.3.8 Layer 3

Enthält Informationen über den dritten Layer entsprechend dem OSI Modell. Sämtliche Layer die durch VLAN und Per VLAN Spanning Tree erstellt wurden, werden hier wieder zusammengefasst. Die Router bekommen ihre Loopback-Interfaces und IP Adressen.

Informationen über einen Knoten

Keine	

Informationen über ein Interface

IP Adresse	192.168.1.0
Subnetzmaske	255.255.255.0

Informationen über eine Kante

Keine	

5.3.9 Layer 3 Sublayer: Routing

Enthält generelle Informationen zu Routing, hauptsächlich die Routingtabellen der einzelnen Geräte.

Informationen über einen Knoten

Die nachfolgenden Informationen werden zu einem „Route-Objekt“ zusammengefasst. Abgespeichert wird eine Liste dieser Objekte.

Routentyp	connected, local, static
Subnetze	10.20.1.0/24, 10.20.1.21/32, 10.2.0.0/24
next-hops	Nothing, Nothing, 10.20.1.1
next-Hop-Interfaces	directly connected, directly connected, Nothing

Informationen über ein Interface

Keine	

Informationen über eine Kante

Keine	

5.3.10 Layer 3 Sublayer: OSPF

Enthält sämtliche Informationen zum Routingprotokoll OSPF. Pro Area und pro OSPF-Prozess die auf einem Gerät konfiguriert sind, wird ein neues Area- oder Prozess-Objekt erstellt.

OSPF Prozesse

Ein OSPF-Prozess der auf einem Gerät ausgeführt wird. Ein OSPF-Prozess ist immer genau mit einem Gerät verbunden.

Prozess Name	OSPF 1
Router ID	192.168.0.1
Reference-Bandwidth	1000
Propagate Default static route	Yes

OSPF Areas

Eine OSPF-Area die auf einem Prozess läuft. Eine Area ist auch immer genau mit einem Prozess verbunden.

Area	BACKBONE 0
Area authentication	None
Area route summarization	10.1.0.0 255.255.252.0
Area redistribute other routes	BGP
Area Networks	10.20.1.0 0.0.0.255

Informationen über einen Knoten

Keine (Nur Referenzen auf OSPF-Prozesse)	

Informationen über ein Interface

Ein Interface ist immer mit einer oder mehreren OSPF-Areas verbunden und enthält noch zusätzliche Informationen.

OSPF Passive Interface	No
OSPF Hello Intervall	1
OSPF Dead Interval	3
OSPF Wait Interval	3
OSPF Retransmit Interval	5
OSPF Priority	1

Informationen über eine Kante

Keine	

5.3.11 Datenmodell

Nachfolgend ein zweiteiliges Datenmodell der einzelnen Layern und Sublayern. Die unteren Layer sind auf der nächsten Seite einsehbar.

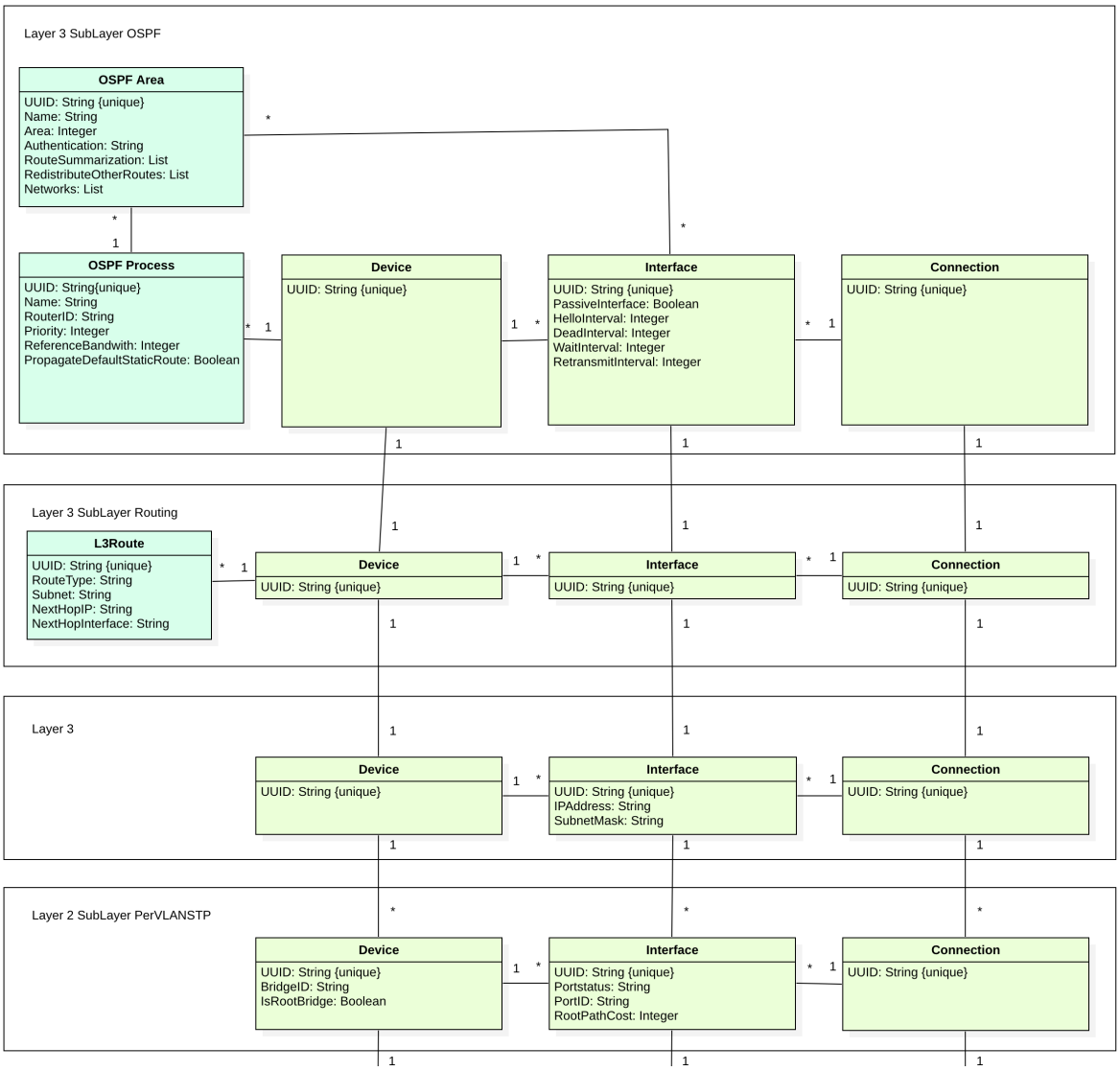

Abbildung 12: Data Model der Layer I

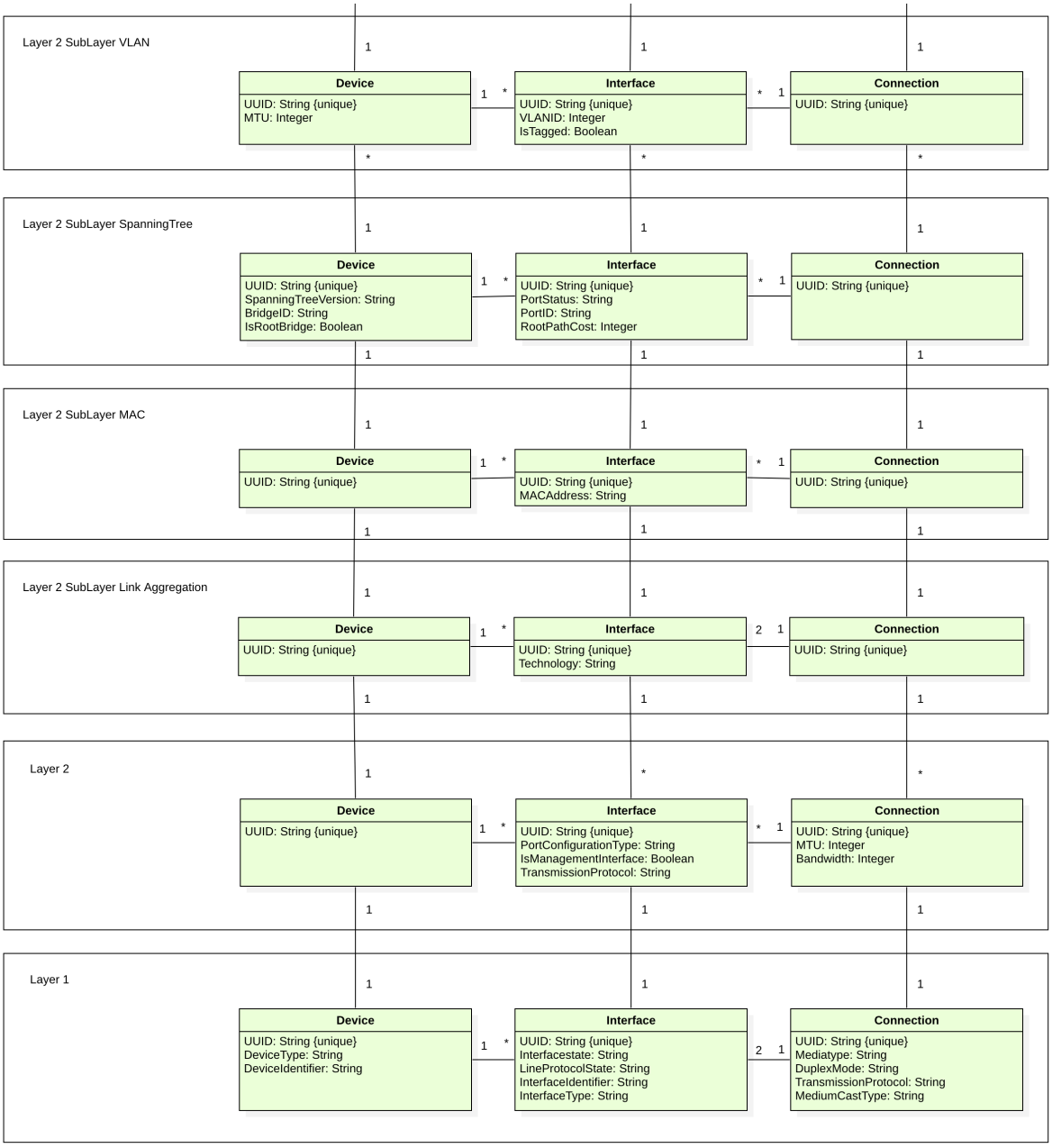

Abbildung 13: Data Model der Layer II

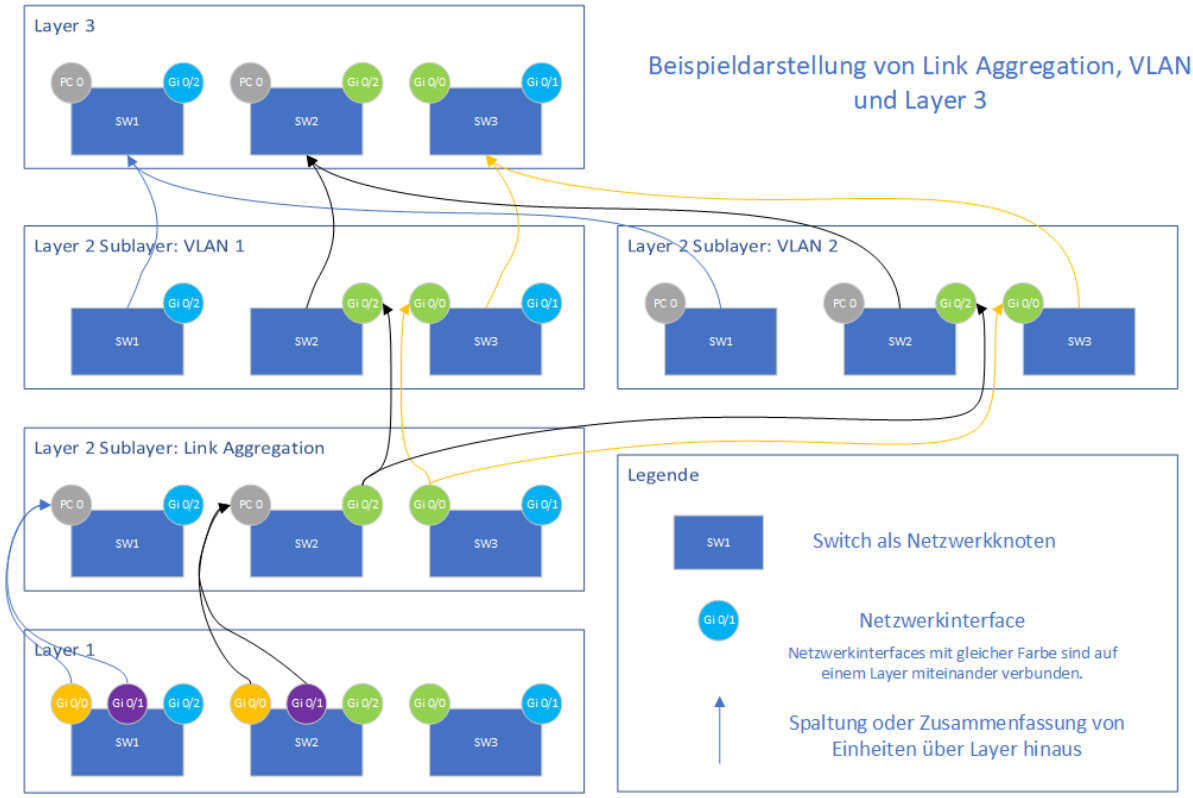

Abbildung 14: Beispiel Layerstruktur

Beispiel Layer Dies ist ein Beispiel einer Layerstruktur mit den Layern VLAN und Link Aggregation. Die Grafik soll illustrieren, wie Knoten, Interfaces und Kanten zwischen den Layern zusammengefasst, bzw. wieder aufgesplittet werden können. Diese Darstellung dient als Beispiel und zeigt lediglich einen Ausschnitt der Layerarchitektur.

5.4 Datenspeicherung

5.4.1 Datenstruktur

Für normale Datenbanken würde hier jetzt ein klassisches Domain-Model präsentiert werden. Eine Graphen Datenbank ist jedoch schemalos, sprich es gibt kein starres Model. In einer Graphen Datenbank existieren Knoten und Kanten, die Knoten verbinden. Es werden verschiedene Arten von Knoten und Kanten mit Eigenschaften definiert und diese können dann mit Daten befüllt und untereinander verbunden werden. In unserem Fall werden für jeden Sublayer eigene Knoten und Kanten definiert. Um die Layer einheitlich zu halten, werden wir eine Struktur vorgeben, an die sich alle zukünftigen Layer halten müssen. Die Eigenschaften der vorgegebenen Knoten und Kanten kann jeder Layer selbst definieren.

Verbindungen zwischen DB-Nodes Für die Speicherung einer Topologie in der Graphen Datenbank werden grundsätzlich drei Arten von DB-Nodes verwendet. Jeweils eine für Netzwerkgeräte, eine für Interfaces und eine Node für Verbindungen. Interfaces werden in der Datenbank mittels einer Is_Attached_To Beziehung an ein Netzwerkgerät gebunden. Verbindungen zwischen Zwei Interfaces werden so realisiert, dass zwei Interfaces eine Is_Connected_To Beziehung zu einer Connection DB-Node besitzen. Eine simple Verbindung zwischen zwei Switches widerspiegelt sich folgendermassen in der Datenbank:

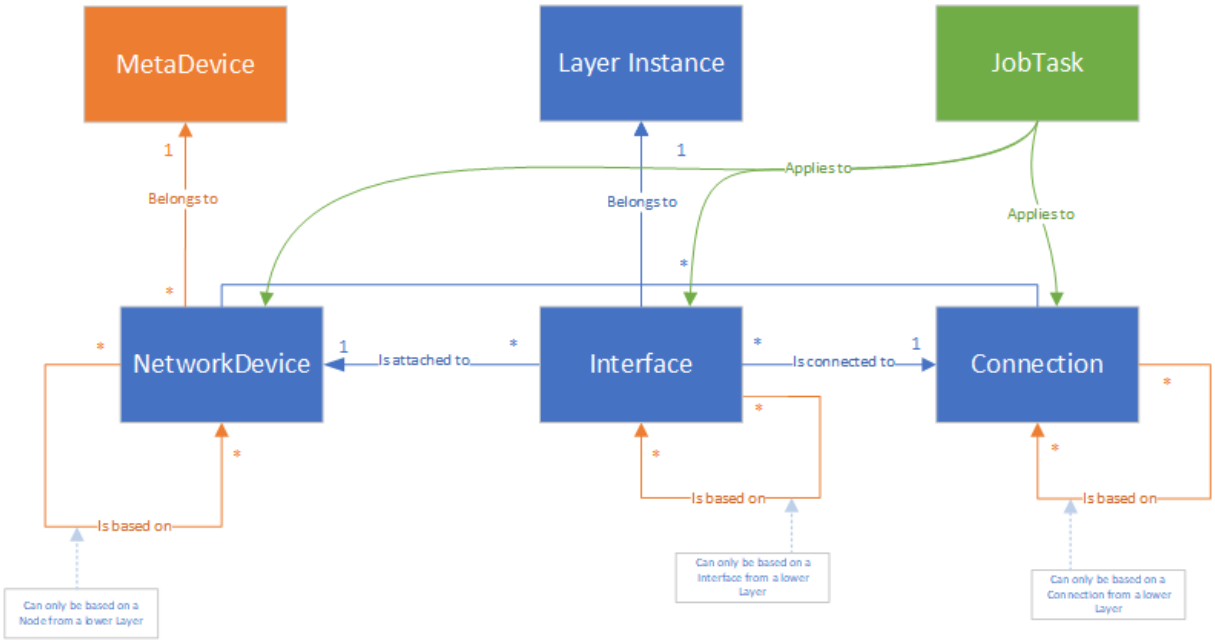

Abbildung 15: Domainmodel

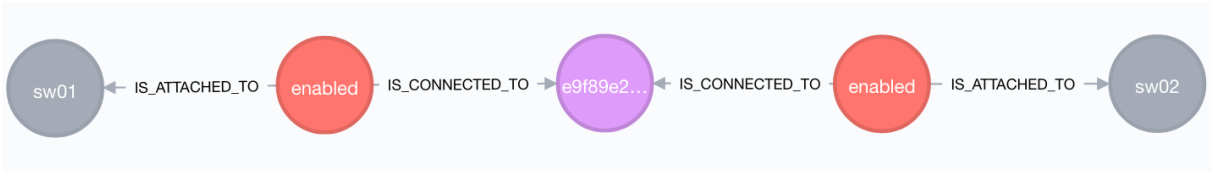

Abbildung 16: Datenbankausschnitt einer einfachen Verbindung

Verbindungen von NetworkDevices finden über Interfaces und Connections statt. Jedes Interface gehört zu genau einem NetworkDevice und einer Connection. Zwei NetworkDevices sind also immer über zwei Interfaces mit einer Connection dazwischen verbunden.

Verbindungen zwischen Layer Damit der Digital Twin auch weiss, welche Komponenten welche sind über die Layer hinaus, braucht es die orange dargestellte „Is based on“ Relation. Diese orangen Relationen sind die einzigen, die mit Knoten in anderen Layern verbunden sein dürfen und auch müssen. Allerdings immer nur auf ein Objekt im direkt darunterliegenden Layer.

Layer Instanz Die Layerinstanz wurde für ein Sublayer wie VLAN, der aus mehreren Instanzen besteht eingeführt (für jedes VLAN eine separate Instanz). Die meisten (Sub)Layer werden nur eine Instanz besitzen.

Meta Device Ein Meta Device repräsentiert ein physikalisches Gerät im Digital Twin und beinhaltet die Informationen über ein Gerät die nötig sind, um es über das Netzwerk ansprechen zu können. Für den Digital Twin sind das IP Adresse, Benutzername und Kennwort. Jeder Netzwerkknoten auf einem Layer ist mit seinem MetaDevice verknüpft, damit die Zugangsdaten einfach abgerufen können, wenn nötig.

Damit sensitive Daten, wie Benutzername und Kennwort nicht im Klartext in der Datenbank gespeichert werden, verschlüsselt die Railsapplikation diese beim Erstellen eines MetaDevices. In der Datenbank

sind diese Informationen verschlüsselt gespeichert und können nur mit entsprechendem Schlüssel entschlüsselt werden. Dieser Schlüssel ist nur der Railsapplikation zur Laufzeit bekannt. Dies bringt den Vorteil, dass sensitive Daten von unbefugten weniger einfach auszulesen sind und wird zudem von unseren Anforderungen an das Produkt verlangt.

JobTask Ein Task wird in Rails im Hintergrund ausgeführt. Dies bedeutet, dass Exceptions und Informationen über den Task nicht normal abgefragt werden können, wenn dieser gestartet wurde. Das JobTask Objekt dient dazu, alle relevanten Informationen über einen laufenden Task zu speichern und dem Benutzer diese anzubieten.

5.5 Webserver

5.5.1 Systemübersicht

Der Digital Twin besteht nur aus einem Tier, ist also ein Monolith. Auf dem Server wird eine Neo4j-Datenbank gehostet, die auch auf einen separaten Server ausgelagert werden könnte. Ausser der Datenbank werden sämtliche Komponenten im Ruby on Rails-Framework ausgeführt.

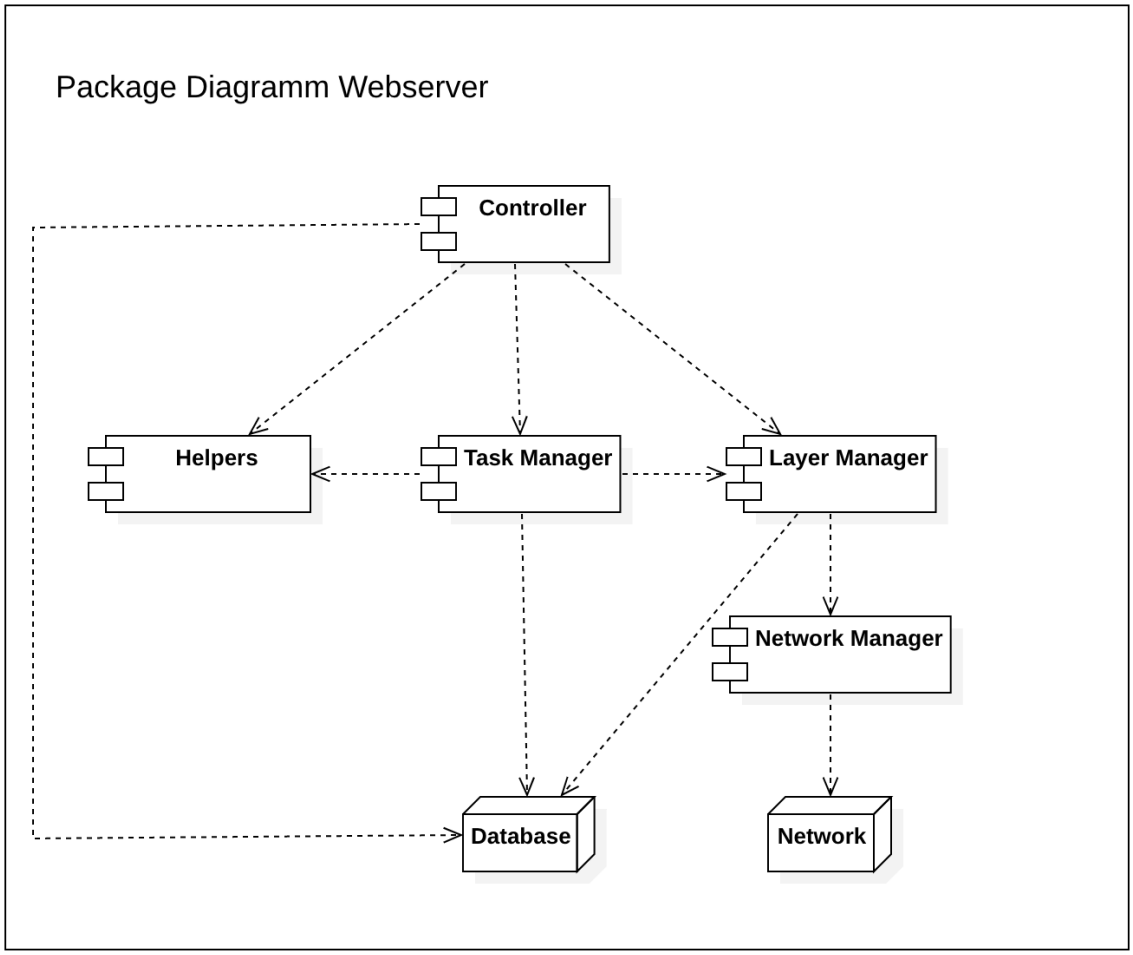

Abbildung 17: Package-Diagramm mit den zugehörigen Abhängigkeiten

Controller Rails verwendet das Model-View-Controller Pattern. Sämtliche Controller, die von Rails benutzt werden, sind im Package „Controller“ vorhanden. Bei jedem Webrequest wird eine Funktion in einem Controller aufgerufen. Dadurch dienen die Controller als Einstiegspunkt für unser Programm.

Task Manager Wie bereits erwähnt, kümmern sich die Controller um das Handling der Webrequests. Das bedeutet im Umkehrschluss, dass der Controller keine Funktionen besitzen darf die lange blockieren, da ansonsten der Webrequest zu lange dauert. Der Task-Manager ist genau dafür da, er erstellt Tasks für Aufgaben die länger dauern können und entlastet so den Controller. Er bietet zudem Informationen zu den vergangenen und noch laufenden Tasks.

Network Manager Der Network Manager kümmert sich um sämtliche Kommunikation mit Netzwerkgeräten. Er kann die verfügbaren Yang-Models abfragen und direkt RESTCONF-Anfragen abschicken.

Layer Manager Wie schon im Namen erwähnt, kümmert sich der Layer Manager um die verschiedenen Layer und stellt diese für den Controller bereit. Wenn jemand dem Digital Twin einen Layer hinzufügen will, ist sämtlicher Code den er editieren muss, im Layer Manager zu finden. Hier sind die Daten-Models und auch die Definition der jeweiligen YANG-Models zu finden.

5.5.2 Sequenzdiagramme

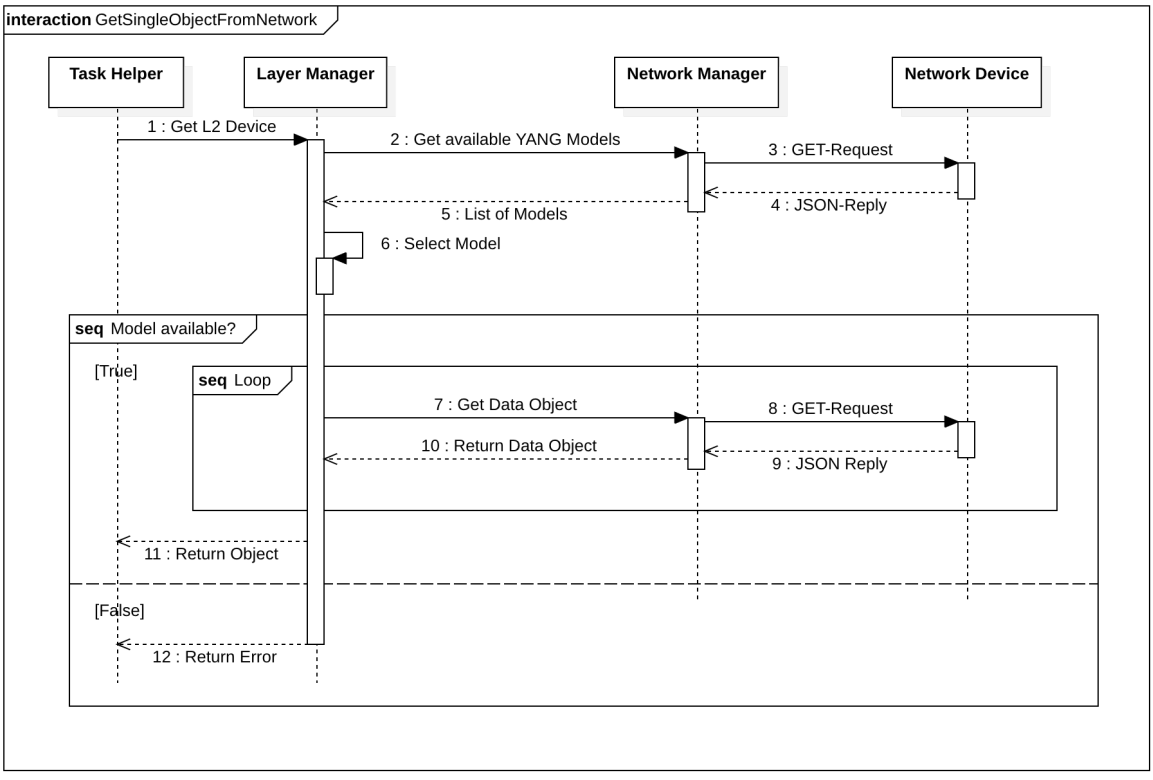

Abbildung 18: Sequenzdiagramm für GetSingleObjectFromNetwork

GetSingleObjectFromNetwork GetSingleObjectFromNetwork ist eine grundlegende Aktion des Digital Twin. Es wird ein Object (Device, Interface, Connection) aus dem Netzwerk abgefragt. Dazu wird zuerst ermittelt welche YANG-Models das Ziel unterstützt, anschliessend werden alle benötigten Daten

per RESTCONF abgerufen. Für den Fall, dass kein kompatibles YANG-Model gefunden wird, wird ein Error zurückgegeben.

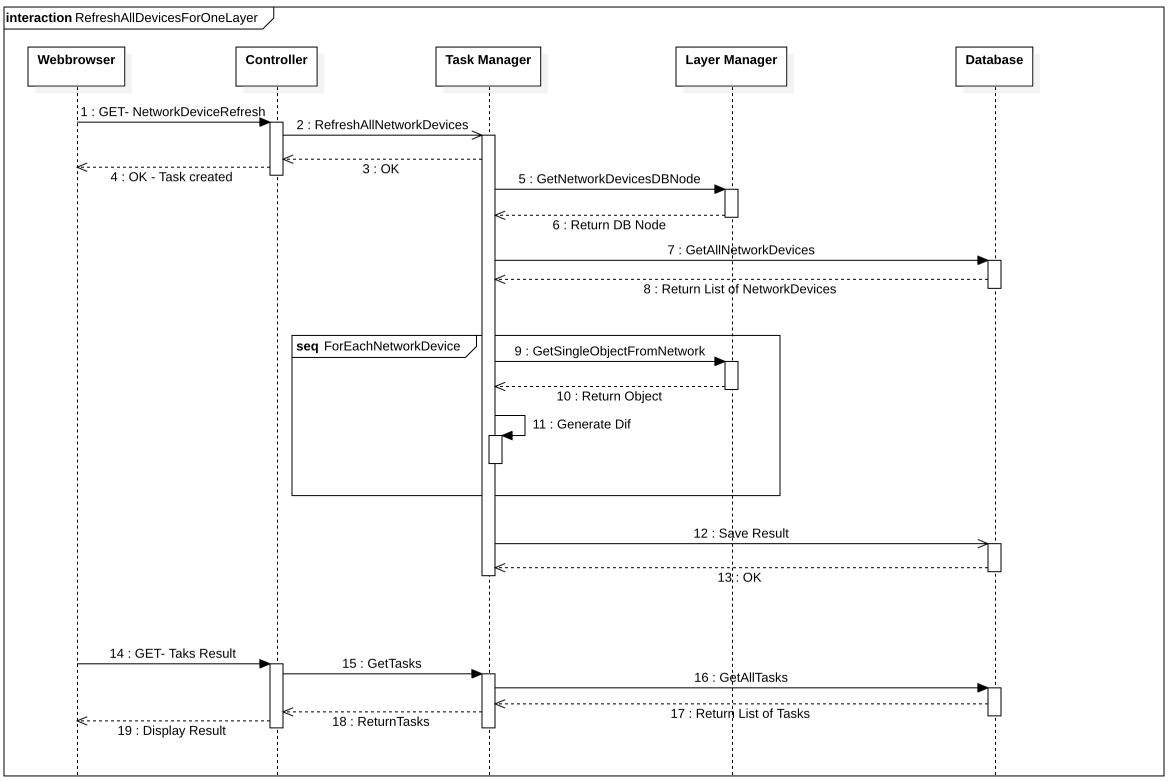

Abbildung 19: Sequenzdiagramm für RefreshAllDevicesForOneLayer

RefreshAllDevicesForOneLayer RefreshAllDevicesForOneLayer ist ein perfektes Beispiel für einen Einsatz des Task Helpers. Auch wenn GetSingleObjectFromNetwork schon lange dauern kann, dieselbe Prozedur für alle Geräte im Layer zu wiederholen wäre definitiv zu viel für den Controller. Deshalb macht er nur eine kleine Anfrage an den Task Helper und dieser übernimmt den Rest. Zuerst besorgt er sich die DB-Node vom Layer-Manager mit der er die benötigten Objekte aus der Datenbank laden kann. Dies ist nötig, da die Geräte layerabhängig sind, und ausserhalb des Layer Managers niemand die einzelnen Layer kennt. Mittels dieser DB-Node kommt er dennoch an die Objekte. Anschliessend muss nur noch GetSingleObjectFromNetwork für jedes Gerät abgerufen werden. Die Anfrage ganz unten stellt eine Abfrage des aktuellen Status des Tasks dar.

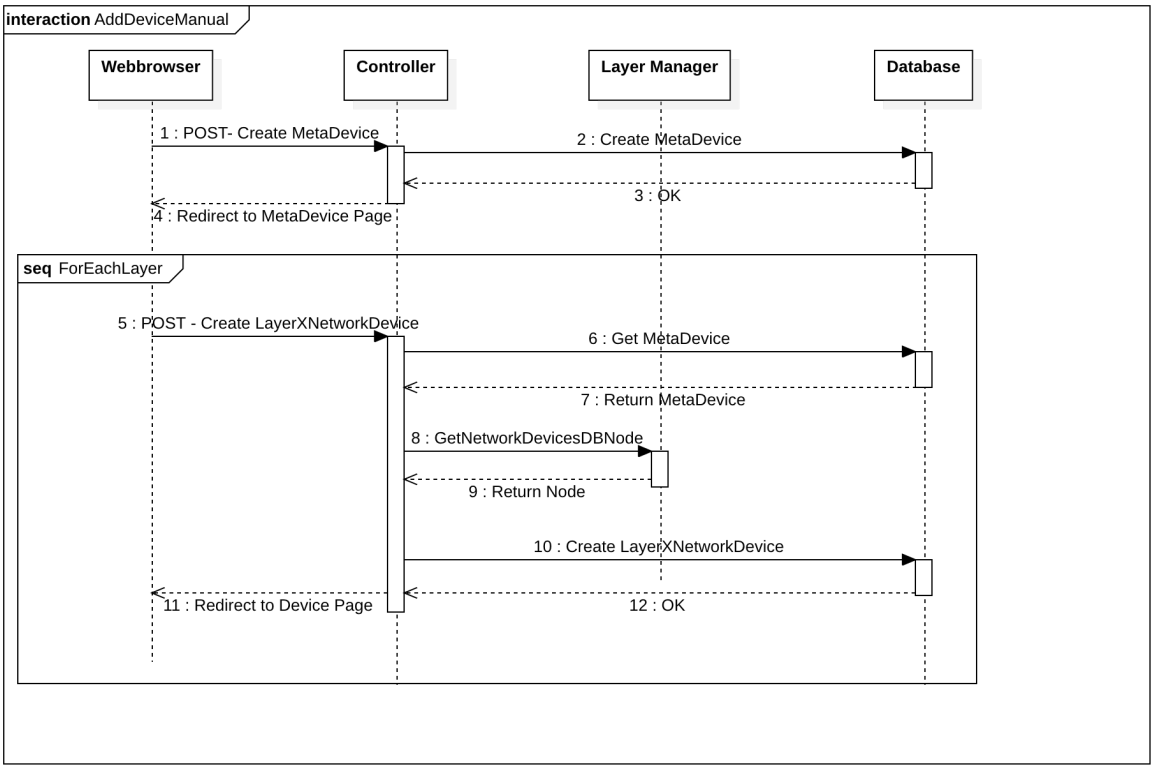

Abbildung 20: Sequenzdiagramm für AddDeviceManual

AddDeviceManual AddDeviceManual ist die Prozedur, die durchgeführt wird, wenn ein Benutzer ein Netzwerkgerät manuell erfassen will. Zuerst erstellt er das MetaDevice, dieses beinhaltet IP, Benutzername und Password. Anschliessend wird für jeden Layer ein Device erstellt. Da für jeden Layer der Netzwerkknoten auch mit dem MetaDevice verknüpft werden soll, muss das MetaDevice auch jedesmal geladen werden.

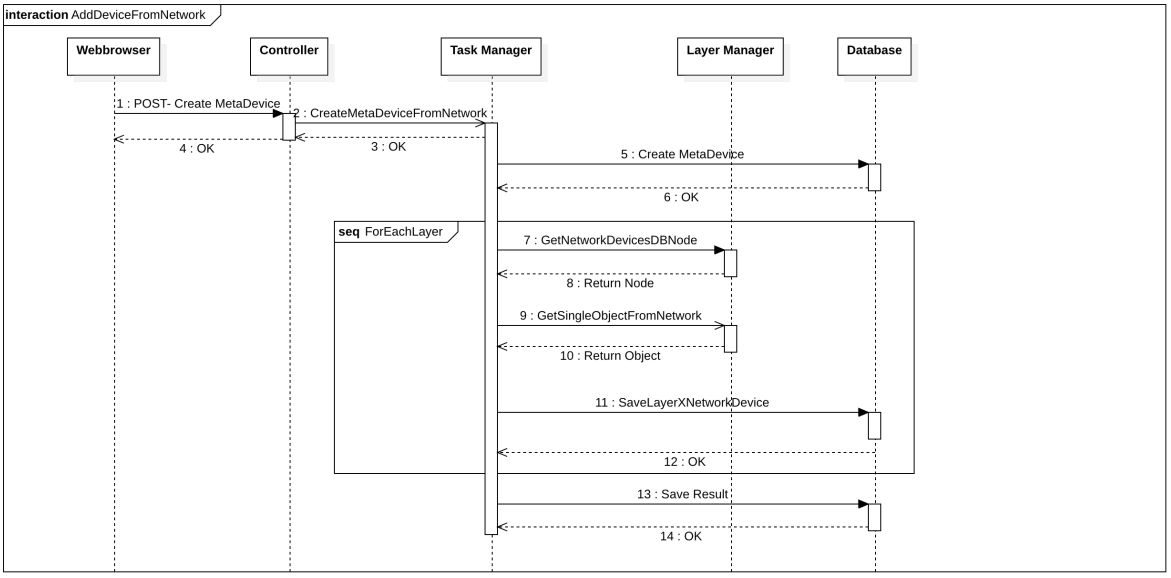

Abbildung 21: Sequenzdiagramm für AddDeviceFromNetwork

AddDeviceFromNetwork AddDeviceFromNetwork ist die automatische Version der vorherigen Aktion. Zuerst wird das MetaDevice erstellt, danach für jeden Layer GetSingleObjectFromNetwork ausgeführt. Am Schluss wird noch das Ergebnis des Tasks in der Datenbank gespeichert.

5.5.3 Architektur Packages

Für jedes Package wird hier ein Klassendiagramm erstellt, um eine Übersicht zu gewährleisten. Jede Klasse die in der Datenbank gespeichert werden kann, ist grün gekennzeichnet. Normale Klassen sind blau.

Abbildung 22: Klassen für den NetworkHelper

Network Manager Der Network Manager ist ein einfaches Package, es besteht nur aus dem MetaDevice, das die relevanten Zugangsdaten eines Gerätes besitzt und dem NetworkHelper, der RESTCONF-Anfragen über das Netzwerk ermöglicht.

Das MetaDevice beinhaltet alle Informationen, die vonnöten sind um mit einem Netzwerkgerät zu kommunizieren. Der Benutzername und das Passwort werden verschlüsselt in der Datenbank gespeichert. Durch eine Inkompatibilität von Neo4j und Rails mussten noch zwei zusätzliche Variablen „decrypted_username“ und „decrypted_password“ erstellt werden, diese beiden Felder werden allerdings nicht in der Datenbank gespeichert.

Abbildung 23: Klassen für den TaskManager

Task Manager Der Task Manager ist für die Erledigung von Hintergrundtasks verantwortlich. Für jede Aufgabe, die parallelisiert werden soll, gibt es ein eigenes Job-Objekt, das die Logik beinhaltet, dies ist von Rails vorgegeben. Der aktuelle Status und das Ergebnis eines Tasks werden zusammen mit zusätzlichen Metainformationen in einem JobTask Objekt gespeichert, welches in der DB gesichert wird. Der TaskHelper bündelt Funktionen, die von vielen Jobs verwendet werden und vermindert so duplicated Code.

Die möglichen Werte für den Status eines JobsTasks sind:

- **In Queue**
  Der Task ist mit einem bestimmten Job erstellt worden, aber da die Anzahl paralleler Threads begrenzt ist, kann er noch nicht ausgeführt werden und befindet sich in einer Warteschlange. In diesem Status kann der Name des Tasks noch nicht genau definiert werden und ist deshalb noch generisch.
- **Searching Objects**
  Da ein Job in Rails nur einfache Typen und keine komplexen Objekte als Parameter annehmen kann, werden dem Job IDs übergeben. In diesem Status sucht der Task nach den Objekten mit den entsprechenden IDs. Mithilfe dieser Objekte kann er anschliessend den richtigen Namen des JobTasks bestimmen.
- **Running**
  Nachdem die Objekte gefunden und der Name gesetzt wurde, beginnt der Task mit der Abarbeitung des Jobs. Dieser Status kann je nach Job länger dauern.
- **Finished**
  Der Task hat seinen Job erfolgreich abgeschlossen. Das Ergebnis wurde in das Result geschrieben.
- **Failed**
  Es ist ein Fehler während dem Ausführen des Tasks aufgetreten und er konnte nicht erfolgreich abgeschlossen werden. Weitere Informationen zum Fehler stehen im Result.

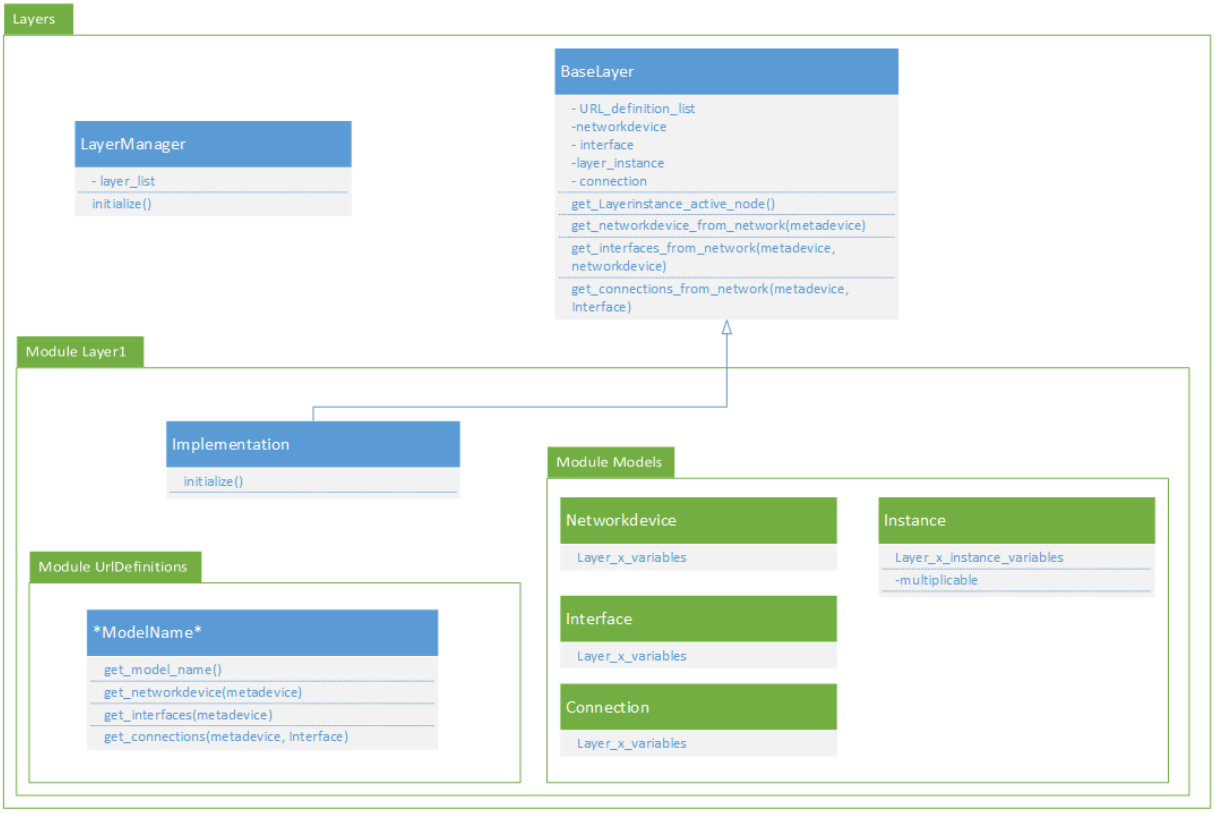

Abbildung 24: Klassen für den LayerManager

Layer Manager Der Layer Manager kapselt einzelnen Layer und stellt sie für andere Komponenten so bereit, dass diese die genaue Implementation der Layer nicht kennen müssen. So kann der Digital Twin um weitere Layer erweitert werden, ohne das Code ausserhalb des Layer-Manager geändert werden muss.

Der Layer Manager benutzt als erstes Package das Konzept von Ruby-Modulen. Ein Modul ist eine Sammlung von Klassen die voneinander isoliert sind. Jeder Layer ist in einem Modul implementiert, so können die Klassen der einzelnen Layer den gleichen Namen besitzen ohne dass es Namenskonflikte gibt. Für eine bessere Übersichtlichkeit werden Models und URLDefinitions nochmal in eigene Module verpackt.

Das Models-Modul beinhaltet sämtliche Klassen eines Layers die in der Datenbank gespeichert werden. Dies sind Netzwerkknoten, Interface und Connection und zusätzlich noch die Klasse Instance. Ein Layer kann mehrere Instanzen besitzen, als Beispiel der VLAN Layer, hier wird für jedes VLAN eine neue Instanz generiert. Jede Entity des Layers (Knoten, Interface, Connection) wird in der Datenbank mit einer Instanz zugeordnet.

Das URLDefinitions-Modul kapselt die einzelnen URL-Definitionen für einen Layer. Die Klasse *ModuleName* ist ein Beispiel wie eine solche Klasse aussehen kann. Für jedes YANG-Model das unterstützt wird gibt es eine solche Klasse. In dieser ist festgelegt wie eine Entity über das Netzwerk von einem Gerät abgerufen werden kann. Eine URL-Definition bedient sich immer genau einem YANG-Model um an die benötigten Daten zu kommen. Der Name dieses YANG-Models kann per *model_name* abgerufen werden. Die anderen drei Funktionen rufen je eine Entity per RESTCONF ab und geben das entsprechende Objekt aus dem Models-Modul mit den Daten zurück. Eine URL-Definition muss nicht alle drei Funktionen implementieren.

Der LayerManager besitzt eine Liste mit allen Layern. Durch diese Liste können andere Komponenten auf die einzelnen Layer zugreifen. Die List muss in der *initialize* Methode des LayerManagers manuell mit den einzelnen Implementationen der Layer befüllt werden.

Die Implementation-Klasse ist diejenige, die im LayerManager gespeichert wird. In der Methode *initialize* werden die Felder der Base-Klasse BaseLayer manuell befüllt. Dadurch weiss der BaseLayer welche Objekte er bedienen muss. Zusätzlich werden auch sämtliche URL-Definitionen in die *url_definition_list* geladen.

Die BaseLayer Klasse besitzt die Funktionen, die von den anderen Komponenten des Digital Twin benutzt werden, um über die URL-Definitionen Daten aus dem Netzwerk zu beziehen. Dazu durchsucht der BaseLayer seine *url_definition_list* um eine geeignete URL-Definition zu finden, die ein YANG-Model besitzt das das gewünschte Gerät unterstützt.

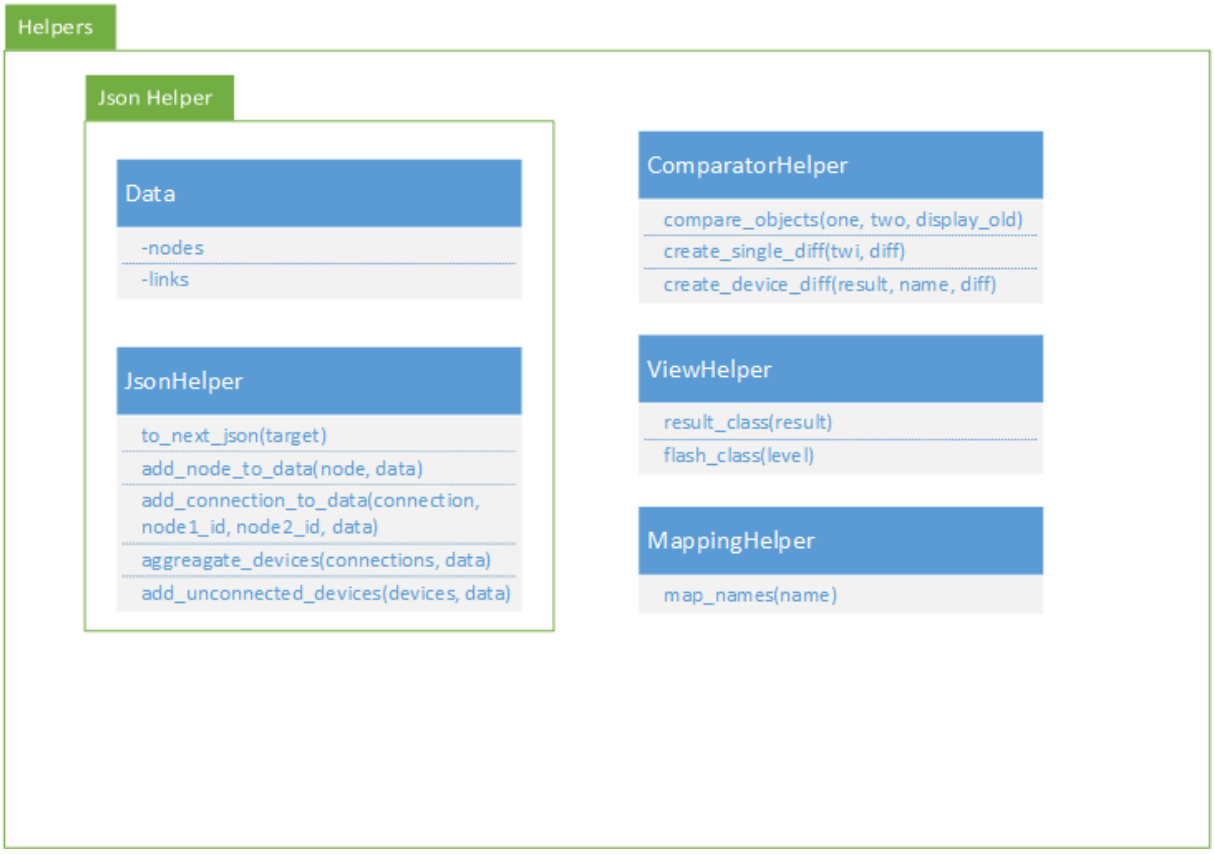

Abbildung 25: Helperklassen

Helpers Helper sind Klassen, die anderen Klassen isolierte Funktionalitäten anbieten. Diese Funktionen können von beliebig vielen Klassen verwendet werden. Die Helpers helfen so Duplikationen zu vermeiden und spezifische Funktionen zu isolieren, was die Code-Qualität verbessert.

Der JSON-Helper ist zuständig für die Konvertierung von Objekten in JSON-Daten die vom NeXt-UI verstanden werden. Dazu benötigt der Helper auch noch ein Data-Objekt, welches das gesammelte JSON speichert.

Der Comparator-Helper bietet die Möglichkeit zwei Objekte nach ihren Attributen zu vergleichen. Dies wird verwendet um Unterschiede anzuzeigen, z.B. zwischen dem Zustand eines Netzwerkdevices in der Datenbank und dem Zustand im echten Gerät. Er besitzt auch die Funktion ein Diff von einem Kompletten Gerät, inklusive zugehörige Interfaces und Connections zu erstellen.

Der View-Helper bietet Funktionalitäten für die Views an, damit zwischen den HTML-Code nicht viel Logik gepackt werden muss. Dies macht den HTML-Code sehr viel lesbarer und bietet die Möglichkeit die Helper-Funktionen separat zu testen.

Der Mapping-Helper wird benötigt um die abgekürzten Namen von Interfaces in vollständige Namen zu übersetzen. Zum Beispiel wird „Gi“ zu „GigabitEthernet“ oder „Fa“ zu „FastEthernet“. Dies wird benötigt, da die Antworten der Geräte bei den Namen der Interfaces nicht konsistent sind, manchmal wird die Abkürzung verwendet, manchmal der volle Name.

5.6 Deployment Diagramm

Der DigitalTwin wird auf einem Puma-Webserver des Ruby on Rails Framework betrieben. Zusammen mit der Neo4j-Datenbank basiert das Ganze auf einem Fedora-Linux. Die Datenbank muss nicht auf dem selber Server wie Rails installiert sein, für unsere Zwecke ist dies aber ausreichend. Die Netzwerkgeräte werden über RESTCONF angesprochen und die Webbrowser-Clients über HTTP(s). Theoretisch könnte Rails auch auf einem Windows-System ausgeführt werden, allerdings ist die Implementation von Ruby auf Windows nicht stabil und zuverlässig genug für den Digital Twin.

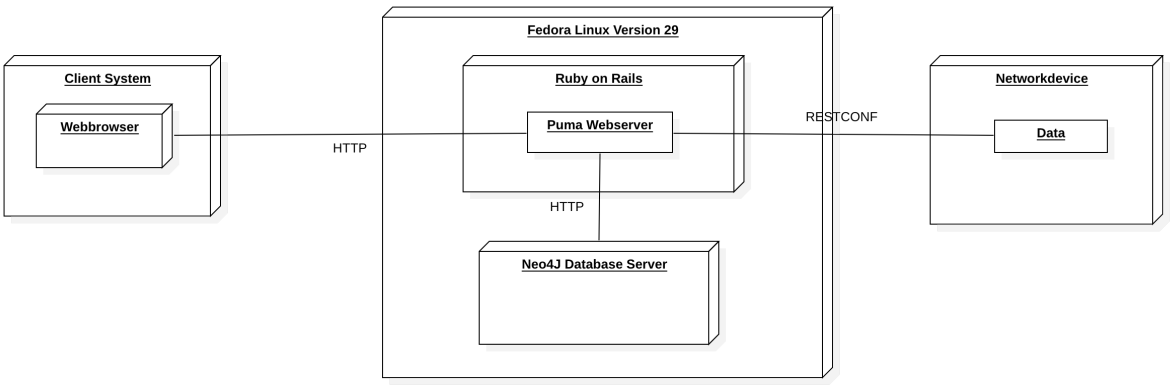

Abbildung 26: Deployment Diagramm

6 Build-Pipeline

Um gute Code-Qualität sicherzustellen, werden Änderungen am Code mittels Build Pipeline vorgenommen. Diese stellt sicher, dass nur Code der sauber getestet und von mindestens zwei Personen kontrolliert wurde, in das Endprodukt kommt. Unsere Build-Pipeline besteht hauptsächlich aus dem Version-Control-System GitHub und dem Build-Server TeamCity mit zusätzlichen Tools.

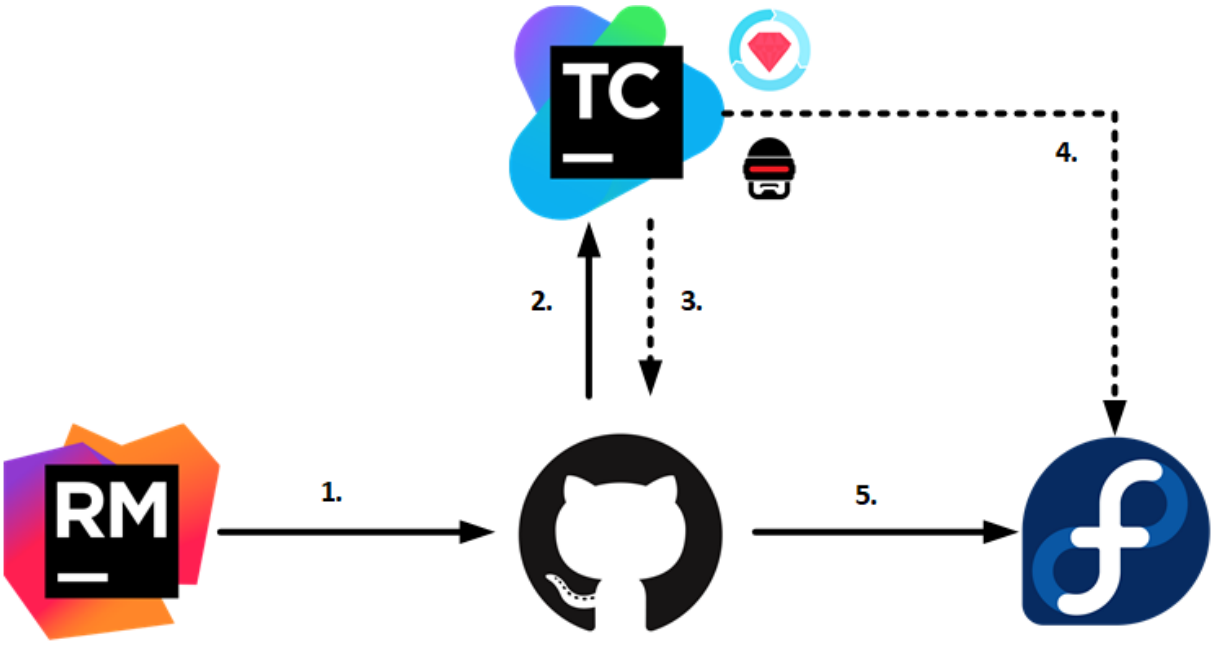

Abbildung 27: Darstellung der Build-Pipeline [Inc18a; Com18; Bc18; Jet18b; Jet18a; Inc18c]

Der Build-Prozess läuft folgendermassen ab:

1. Code wird mittels RubyMine auf den Entwicklermaschinen geschrieben und auf GitHub mittels git eingecheckt.
2. TeamCity pullt bei Änderungen den Code von GitHub, überprüft ob der Code Syntaxfehler enthält, ob alle Tests erfolgreich sind, und ob das Linting Tool keine Fehler meldet.
3. Sind alle Checks erfolgreich, sendet TeamCity eine Benachrichtigung an GitHub. GitHub ist so konfiguriert, dass Code nur per Pull-Request in den Develop-Branch gemerged werden kann, wenn die entsprechenden Checks von TeamCity erfolgreich sind.
4. Bei einem Master-Release wird von TeamCity ein Deployment auf dem Production-Server ausgelöst.
5. Der Production-Server zieht sich automatisch die neue Version von GitHub und wendet diese an. So wird die produktive Umgebung automatisch auf den neusten Stand gebracht.

6.1 Linting

Ein Linter ist ein Tool, das hilft die Qualität des geschriebenen Codes sicherzustellen. Es analysiert den Sourcecode nach Programmfehlern, stilistischen Fehlern und verdächtigen Konstrukten. Ein Linter sorgt auch für einen einheitlichen Codestyle, was externen Programmieren das lesen des Codes vereinfacht.

6.1.1 Rubocop

Rubocop ist ein Linting Tool für Ruby. Es ist der DeFacto Standard für die Überprüfung von Code Guidelines und wird in vielen Ruby on Rails Projekten eingesetzt.

Abbildung 28: Rubocop Logo [Bc18]

Ein grosser Vorteil von Rubocop ist seine einfache Integration in Rails. Es muss nur als zusätzliches Gem (Programm im Package Manager von Ruby) installiert werden und ist schon einsatzbereit. Zusätzlich bietet es auch die Möglichkeit, kleinere Fehler automatisch zu korrigieren, was Zeit spart.

```
Inspecting 69 files
..C.....................................................................

Offenses:

Gemfile:67:1: C: Bundler/OrderedGems: Gems should be sorted in an alphabetical order within their section of the Gemfile. Gem render_async should appear before slim.
gem 'render_async'
^^^^^^^^^^^^^^^^^^

69 files inspected, 1 offense detected
```

Abbildung 29: Beispielausgabe von Rubocop

6.2 TeamCity

TeamCity ist ein CI(continuous Integration) Server. Wie oben beschrieben, wartet er auf neue Änderungen von GitHub, holt diese ab und startet einen neuen Build. Für unsere Buildpipeline wird TeamCity auf einem Server gehostet.

Abbildung 30: TeamCity Logo [Jet18b]

Für den Digital Twin haben wir drei verschiedene Build-Konfigurationen erstellt: „Deployment“, „Rubocop“ und „Tests“. Die beiden Konfigurationen Rubocop und Tests laufen bei jedem Commit, für jeden Branch auf GitHub. Sie führen jeweils den Rubucop-Linter oder die RSpec-Tests aus und melden das Ergebnis an GitHub. Die Deployment Konfiguration wird nur ausgeführt, wenn ein neuer Commit auf dem Master-Branch erkannt wird. Nach diesem Build wird keine Rückmeldung an GitHub gesendet, sondern der Production-Server wird benachrichtigt. Dieser startet daraufhin das Deployment per Skript.

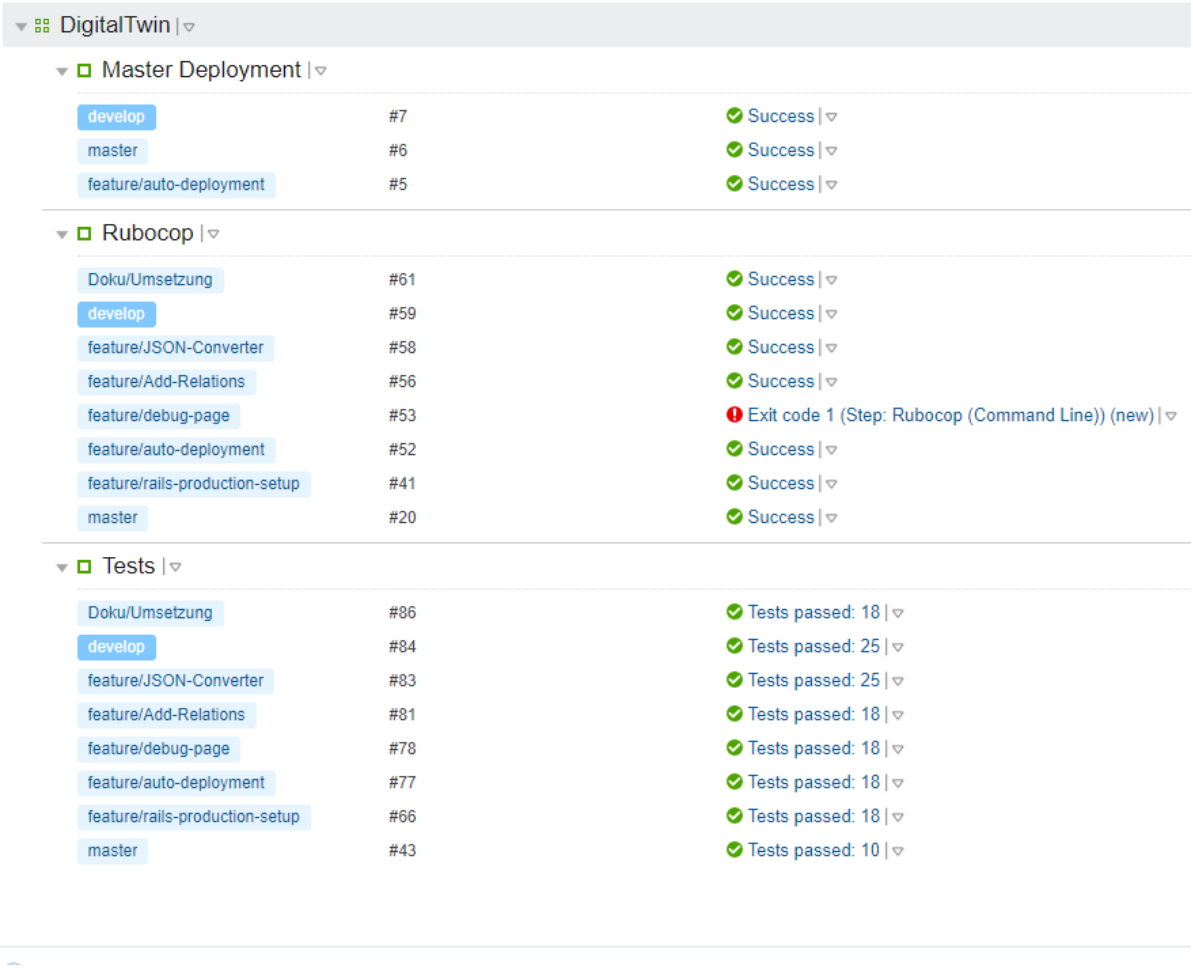

Abbildung 31: Die verschiedenen Build-Konfigurationen auf TeamCity

6.3 Versionsverwaltung

Das Herzstück unserer Build-Pipeline ist die Versionsverwaltung, ein Online Service von GitHub. Alle Änderungen am Code laufen über GitHub und es stellt mittels Git-Flow und den entsprechenden Checks sicher, dass jeglicher neuer Code mindestens von zwei Personen kontrolliert wurde und lauffähig sein sollte.

Abbildung 32: GitHub Logo [Inc18a]

6.3.1 Git-Flow

Das Prinzip von Git-Flow besagt, dass sämtliche neuen Features auf ihren eigenen Feature-Branchs entwickelt werden. Auf diesen Branchs kann sorglos programmiert werden, und es besteht keine Notwendigkeit für erfolgreiche Checks (Tests und Linter). Sobald der Code reif ist, wird ein Pull-Request erstellt, ein Request, den Code des Features zum Rest der Applikation hinzuzufügen. Dafür müssen jedoch alle Checks erfolgreich sein, plus der Code muss von einer anderen Person überprüft werden.

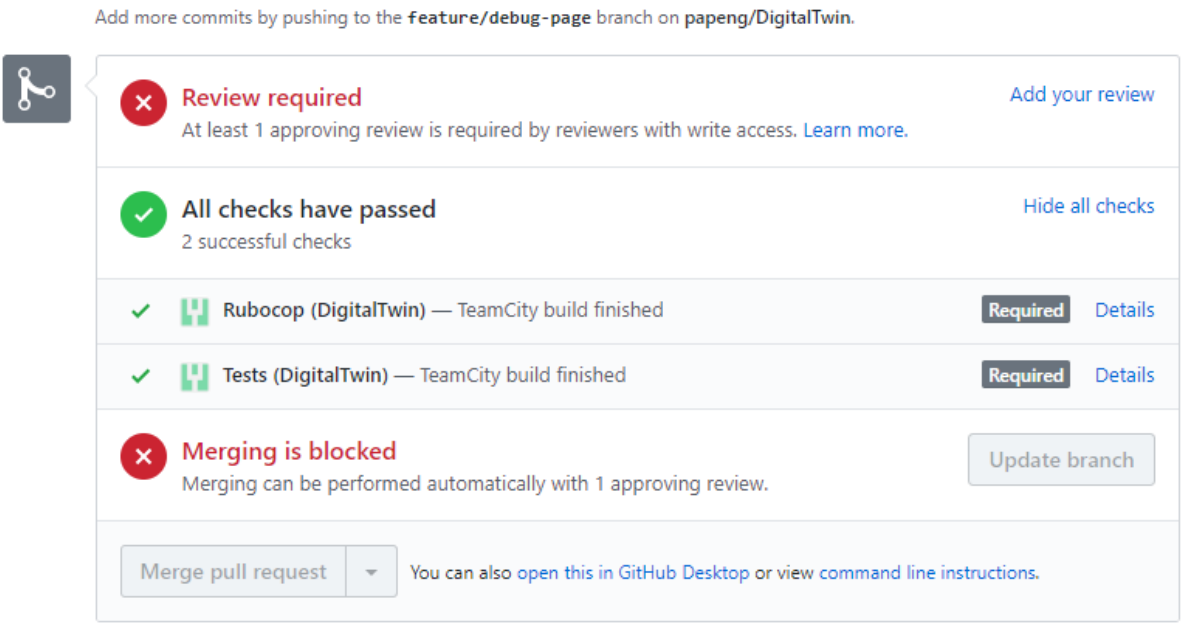

Abbildung 33: Beispieldarstellung eines Pull-Requests

6.4 Testing

Da der Digital Twin erweiterbar sein soll und durch diese Erweiterungen immer wieder Änderungen am Code vorgenommen werden, ist es wichtig dem Entwickler eine Möglichkeit zu geben zu überprüfen ob das Programm noch richtig funktioniert. Dies wird durch Unit-Testing erreicht.

6.4.1 RSpec

Damit die Tests möglichst gut in den Digital Twin integriert und einfach auszuführen sind, wird RSpec als Testing-Framework benutzt. RSpec wird als zusätzliche Library (ein Ruby Gem) zu Rails hinzugefügt und ist so automatisch einsatzbereit, sofern Rails und Ruby richtig eingerichtet worden sind.

Abbildung 34: RSpec Logo [Com18]

RSpec bietet einfache Möglichkeiten zum Mocking von Funktionen und Klassen. So können Funktionen wie das Aufrufen eines Restconf-HTTPS Aufrufs einfach simuliert werden, ohne ein echtes Gerät mit Restconf zur Verfügung zu benötigen. Damit wird sichergestellt, dass die Tests auf jedem Computer erfolgreich ausgeführt werden können und nicht abhängig von der Umgebung sind.

6.4.2 Getestete Funktionalitäten

Grundsätzlich werden zwischen Unit-Tests und Integration-Tests unterschieden. Unit-Tests überprüfen die korrekte Implementation von Klassen und Methoden. Ein solcher Test soll möglichst genau eine Funktion testen, dafür aber schnell durchlaufen. Ein Integration Test deckt mehr Funktionalität ab, kann dadurch aber deutlich längere Laufzeit haben, da z.B. auf Antworten von Webrequests gewartet werden müssen. Je genauer die Funktionalitäten getestet werden sollen, desto mehr Zeit wird benötigt. Tests an sich bringen keinen Fortschritt im der Funktionalität des Projekts, sie erleichtern jedoch Änderungen am Code in der Zukunft. Da wir nur begrenzt Zeit während unserer Studienarbeit haben, konzentrieren wir uns auf Unit-Tests. Unser Hauptziel dabei ist, einen Grossteil des Backends zu testen. Der Fokus dieser Arbeit liegt nicht auf dem Frontend, deshalb werden dafür auch keine Tests erstellt.

6.4.3 Integration

Damit die Tests ihre Wirkung nicht verfehlen, wurden sie Mithilfe von TeamCity und GitHub in unseren Build-Prozess miteingebunden. Ein Feature kann so dem DigitalTwin nur hinzugefügt werden, wenn sämtliche Tests mit positivem Resultat enden. Dies wird im nächsten Kapitel noch genauer beschrieben.

7 Infrastruktur

TeamCity und der Production Server werden von uns selbst vor Ort an der HSR gehostet, was dazu führte, dass wir eine eigene Serverinfrastruktur erstellen mussten. Die Gründe, warum wir uns für Self-Hosting entschieden haben sind unter anderem:

- Von der Schule wurde uns ein Desktopgerät für die Studienarbeit zur Verfügung gestellt, welches wir nach unseren Wünschen benutzen können.
- Im Vergleich zu externem Hosting durch einen kommerziellen Anbieter, ist die gewählte Lösung kostenlos.

Nachfolgend wird der Aufbau der Infrastruktur mit den Verwendeten Technologien genauer erörtert.

7.1 Übersicht

Die Infrastruktur stellt sich zusammen aus einem physikalischen Host an der HSR, auf welchem zwei virtuelle Maschinen (VMs) gehostet sind und einem bereits bestehenden externen Server, auf welchen Backups hochgeladen werden.

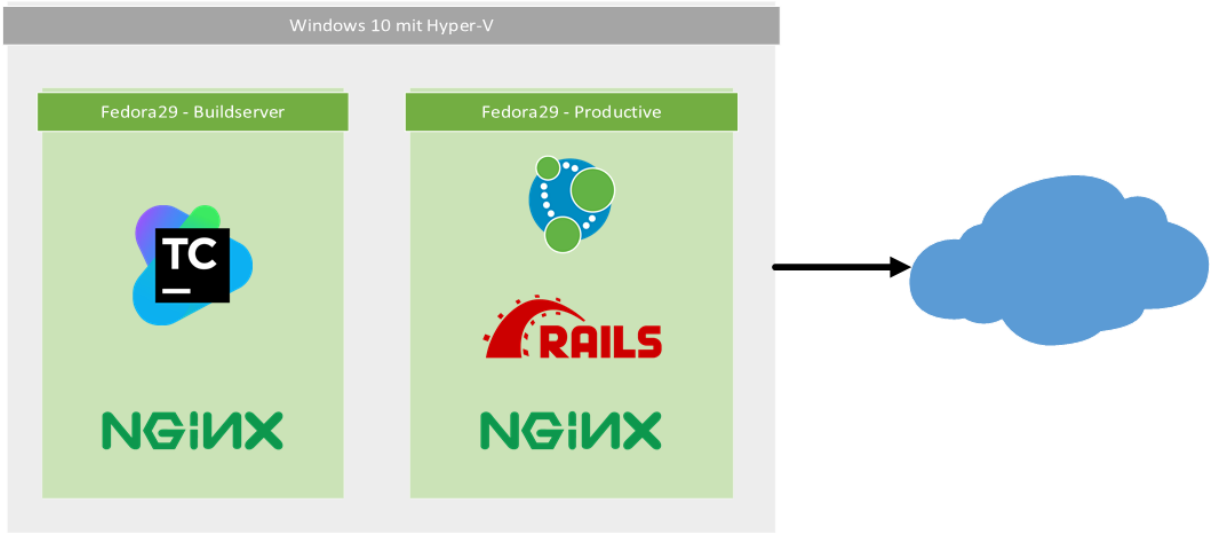

Abbildung 35: Übersicht Infrastruktur [Jet18b; Inc18b; Neo18; Han18]

7.2 Physikalischer Server

Der physikalische Server, welcher für das Hosting der VMs verwendet wird, ist ein Mini-PC mit installiertem Windows 10 Pro, auf welchem Microsoft Hyper-V installiert wurde.

7.3 Virtuelle Server

Mittels Hyper-V werden zwei Virtuelle Maschinen gehostet, eine für den Production-Server und eine für TeamCity. Der Grund, dass nicht beides innerhalb der gleichen Maschine gehostet wird ist, dass so die unterschiedlichen Services sauber voneinander getrennt sind. Für den späteren Betrieb des Digital Twins ist die Buildpipeline nicht nötig.

Als Betriebssystem der Virtuellen Maschinen wird die Linux Distribution Fedora verwendet. Sowohl für die die Rails Applikation, wie auch dem TeamCity Server wird jeweils eine Nginx Instanz als Reverse-

Proxy davor geschaltet. Dies hat den Vorteil, dass so SSL/TLS Verbindungen einfacher konfiguriert werden können. Auf der Virtuellen Maschine mit der Railsapplikation ist zudem eine Neo4j Instanz installiert.

7.4 Backups

Um die Auswirkungen eines Serverausfalls zu minimieren, werden die zwei Virtuellen Maschinen wöchentlich gespeichert und in die private Cloud eines Teammitglieds auf einen externen Server hochgeladen.

8 Resultat

Das Ziel der Arbeit war es eine Architektur zu gestalten, die sämtliche Use-Cases abdeckt und zudem noch einfach erweiterbar und geräteunabhängig ist. Aus dieser Architektur sollte dann ein Digital Twin programmiert werden. Dieser implementierte Digital Twin soll zeigen, dass die vorhandene Architektur auch in der Realität funktioniert.

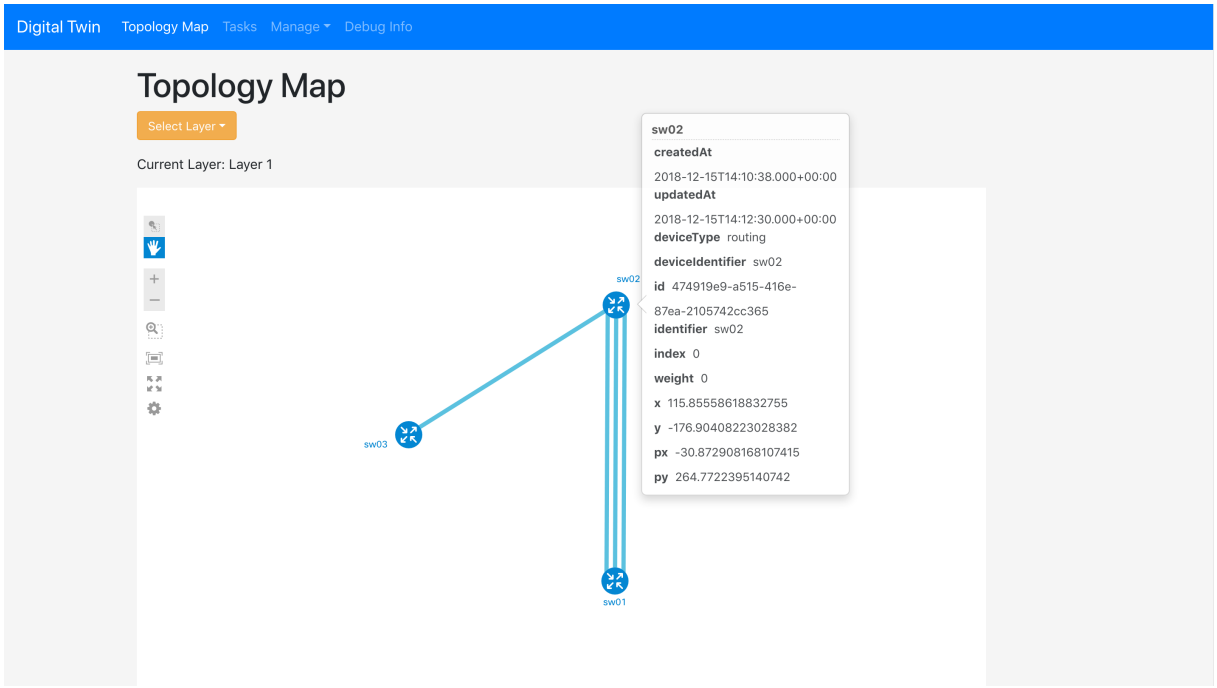

Abbildung 36: Darstellung einer einfachen Netzwerkinfrastruktur auf Layer 1

8.1 Implementierte Layer

Momentan sind nur Layer 1 und Layer 2 (ohne zusätzliche Sublayer) implementiert, für die Implementation weiterer Layer hat die Zeit nicht gereicht. Die beiden vorhandenen Layer reichen jedoch bereits aus, um die Features des Digital Twins zu demonstrieren.

8.2 Use-Cases

Als Resultat der Arbeit entstand ein Digital Twin, der sämtliche Use-Cases abdeckt. Es ist möglich ein Netzwerk automatisch und manuell zu erfassen (UC01, UC02). Die so erstellten Netzwerke lassen

sich bearbeiten (UC03) und es ist möglich die Differenzen zwischen dem Digital Twin und dem realen Netzwerk anzuzeigen (UC04).

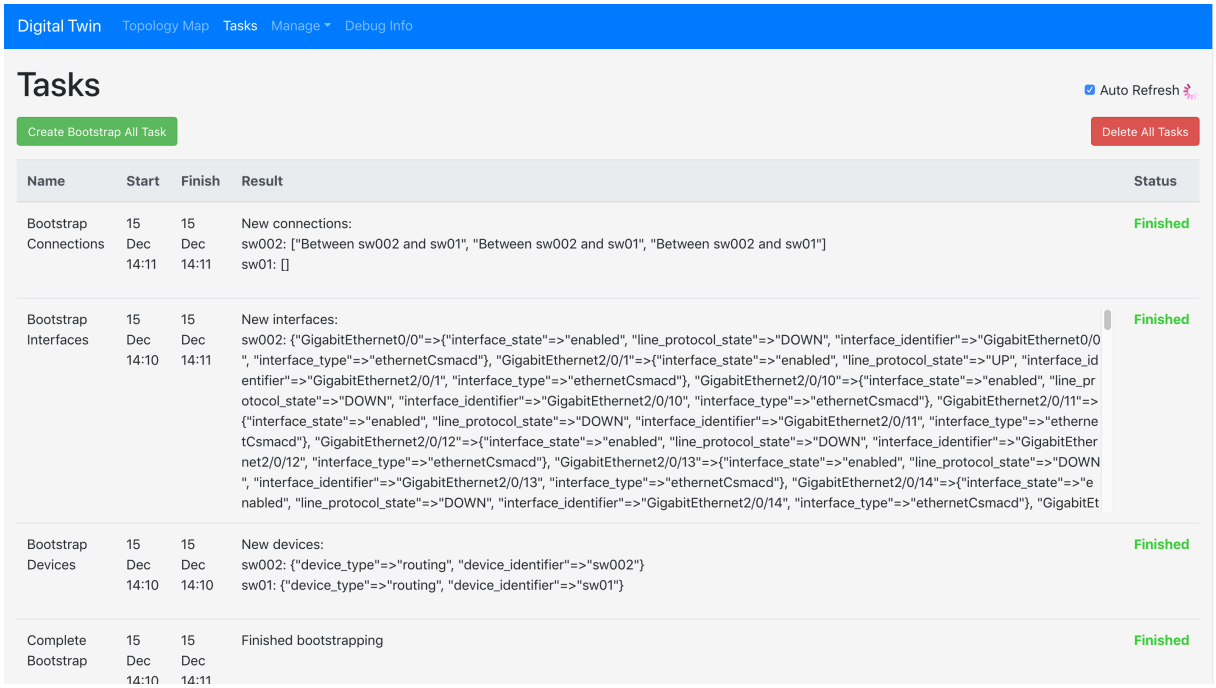

Abbildung 37: Automatische Erfassung eines neuen Netzwerks

Da das Userinterface in dieser Arbeit nicht im Vordergrund stand und nur dazu dient die Daten zu präsentieren, werden die Unterschiede von Digital Twin und echtem Netzwerk momentan nur als Text angezeigt.

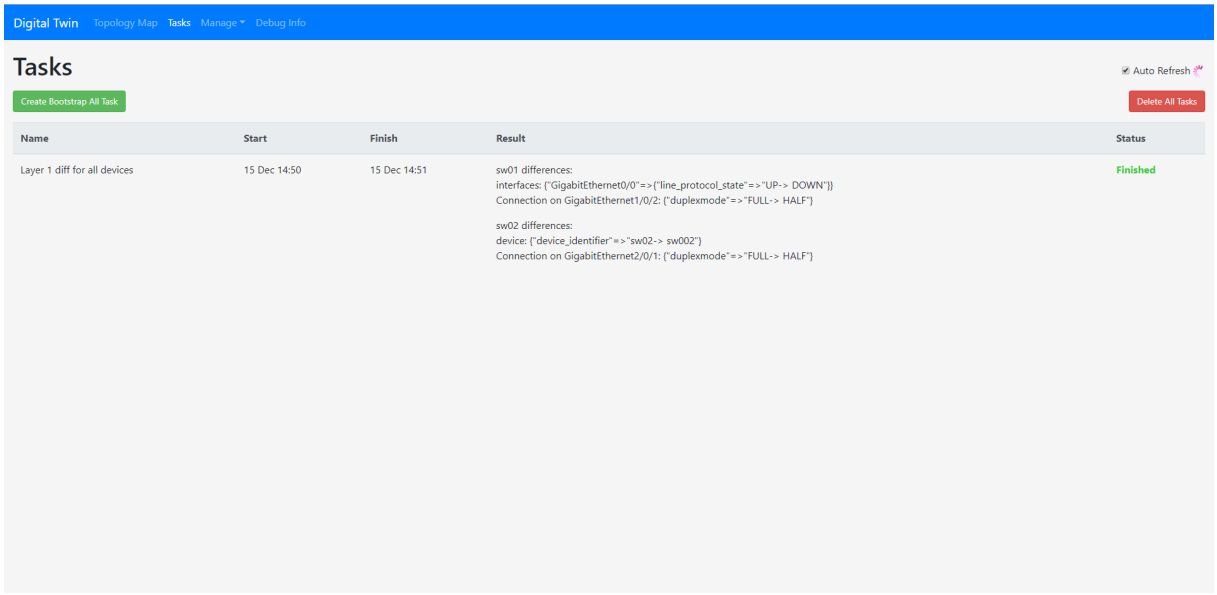

Abbildung 38: Anzeige der Unterschiede

8.3 Nicht funktionale Anforderungen

Sicherheit Im Bezug auf Sicherheit wurden sämtliche NFAs erfüllt. Durch einen Reverse-Proxy wird sämtlicher HTTP-Traffic verschlüsselt und die sensitiven Informationen, in diesem Fall der Benutzername und das Passwort im Meta-Device, werden verschlüsselt bevor sie in der Datenbank gespeichert werden.

Skalierbarkeit Die Verwendung von Neo4j gegenüber einer relationalen Datenbank führt zu einer einfacheren DB-Struktur, was eine gute Skalierbarkeit in der Datenspeicherung sicherstellt. Der grösste Flaschenhals ist momentan noch das Abrufen der Information über Restconf. Diese Aufgaben lassen sich jedoch sehr einfach parallelisieren, sollte die Performance zu sehr darunter leiden. Der Puma Webserver stellt mit der Standardkonfiguration fünf Threads für Parallelisierung bereit, dies lässt sich bei Bedarf ebenfalls erweitern.

Reaktionszeit Die Reaktionszeit des Digital Twins bleibt stehts unter 500 Millisekunden der Durchschnitt beträgt 300 Millisekunden. Die Seite mit der tiefsten Performance war die Topology-Übersicht, dies war allerdings zu erwarten, da die NeXt-Library auf dieser Seite zusätzlich geladen wird. Das Ziel von drei Sekunden wurde somit ohne Probleme erreicht. Sämtliche Arbeiten die länger dauern könnten wurden durch Backgroundtasks automatisiert.

Erweiterbarkeit Die Architektur wurde so entworfen, dass nur ein kleiner, spezifischer Teil der Applikation angepasst werden muss, um neue Layer hinzuzufügen, oder die Kommunikation mit neuen Geräten zu ermöglichen. Mehr Informationen sind in der Anleitung „Hinzufügen neue Layer“ zu finden.

8.4 Testabdeckung

Für jede Software, die einfach erweiterbar sein soll, sind automatisierte Unit-Tests essentiell. Falls jemand Anpassungen an der Applikation vornimmt kann er durch die Tests sofort erkennen, ob seine Änderung funktioniert. Der Digital Twin beinhaltet 155 Tests für den Ruby-Code. Diese Tests decken zusammen rund 65.21% des Codes ab. Die meisten getesteten Klassen haben eine Abdeckung von über 90%, allerdings konnten aufgrund fehlender Zeit für einige Klassen keine Tests erstellt werden.

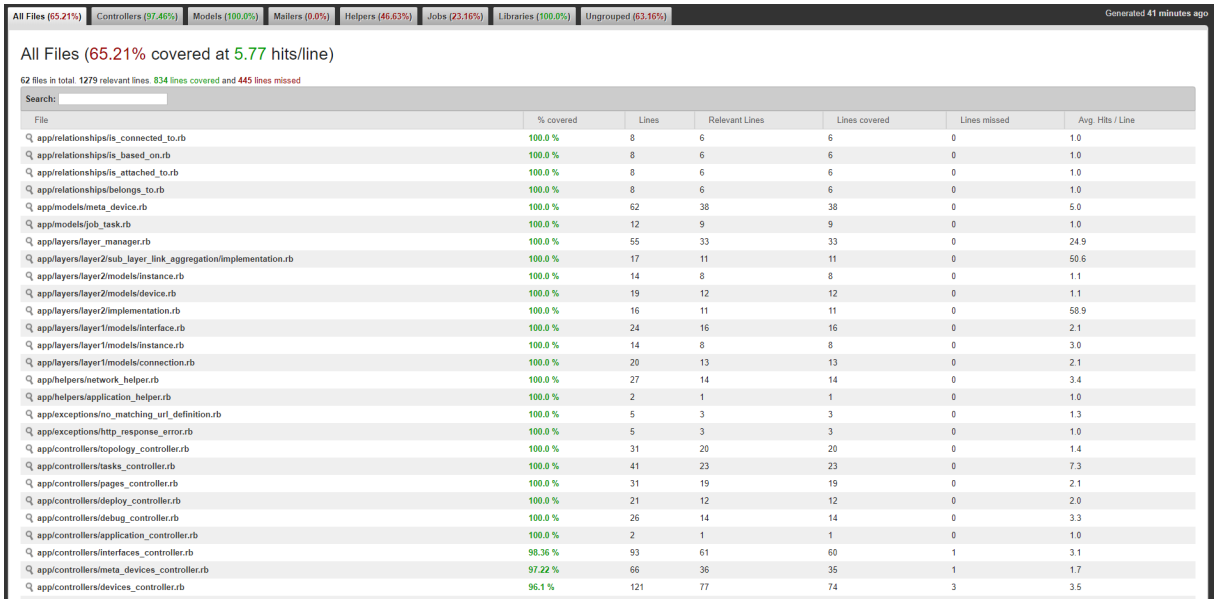

Abbildung 39: Testabdeckung

8.5 Ausblick

Momentan ist der Digital Twin in der Lage eine Netzwerktopologie auf dem ersten Layer darzustellen. Allerdings gibt es noch Potential für Verbesserungen.

8.5.1 Hinzufügen neuer Layer

Damit die Applikation auch wirklich eingesetzt werden kann, muss sie mehr Layer unterstützen. Auch wenn durch die Architektur das Hinzufügen neuer Layer vereinfacht wird, sollte der Aufwand hierfür nicht unterschätzt werden. Das Suchen und Testen von geeigneten YANG-Models und die Interpretation dieser Daten ist dennoch nicht trivial.

8.5.2 Hinzufügen von zusätzlichen YANG-Models

Die vorhandenen YANG-Models wurden nur mit Cisco-Hardware getestet und funktionieren sehr wahrscheinlich nicht mit anderen Geräten. Durch das Hinzufügen von neuen Models kann die Geräteunabhängigkeit des Digital Twins verbessert werden.

8.5.3 Erweiterung des Benutzerinterface

Da der Fokus dieser Arbeit auf der Architektur und nicht auf dem Benutzerinterface lag, gibt es noch viel Potential bei jenem. Allem voran die Darstellung der Informationen der Tasks. Momentan wird diese Information in reinem Text angezeigt.

8.5.4 Mehr Informationen über Tasks

Die Tasks sind momentan nicht in der Datenbank verknüpft mit den Geräten, Interfaces oder Connections, die sie bearbeiten. Es ist für den Benutzer zwar ersichtlich welcher Task welche Geräte bearbeitet, allerdings kann ein Gerät nicht einsehen von welchen Tasks es bearbeitet wurde.

8.5.5 Parallelisierung

Alle Jobs laufen bereits parallel ab und stören so das Laden der Webseite nicht. Das Problem ist, dass grosse Jobs, wie das Scannen des Netzwerks nach Unterschieden eine grosse Laufzeit haben. Diese Jobs könnten ihrerseits parallelisiert werden, um die Laufzeit zu verringern.

8.6 Auswertung Zeitdaten

Der vorgesehene Zeitaufwand für eine Studienarbeit beträgt 8 ECTS Punkte. Ein ECTS-Punkt steht für einen Arbeitsaufwand von 25-30 Stunden. Die Studienarbeit sollte also ungefähr 240 (8 * 30) Stunden pro Student benötigen. Das Semester läuft über 14 Wochen, demnach sind das 17 Stunden pro Woche, auf dem untenstehenden Diagramm blau eingezeichnet.

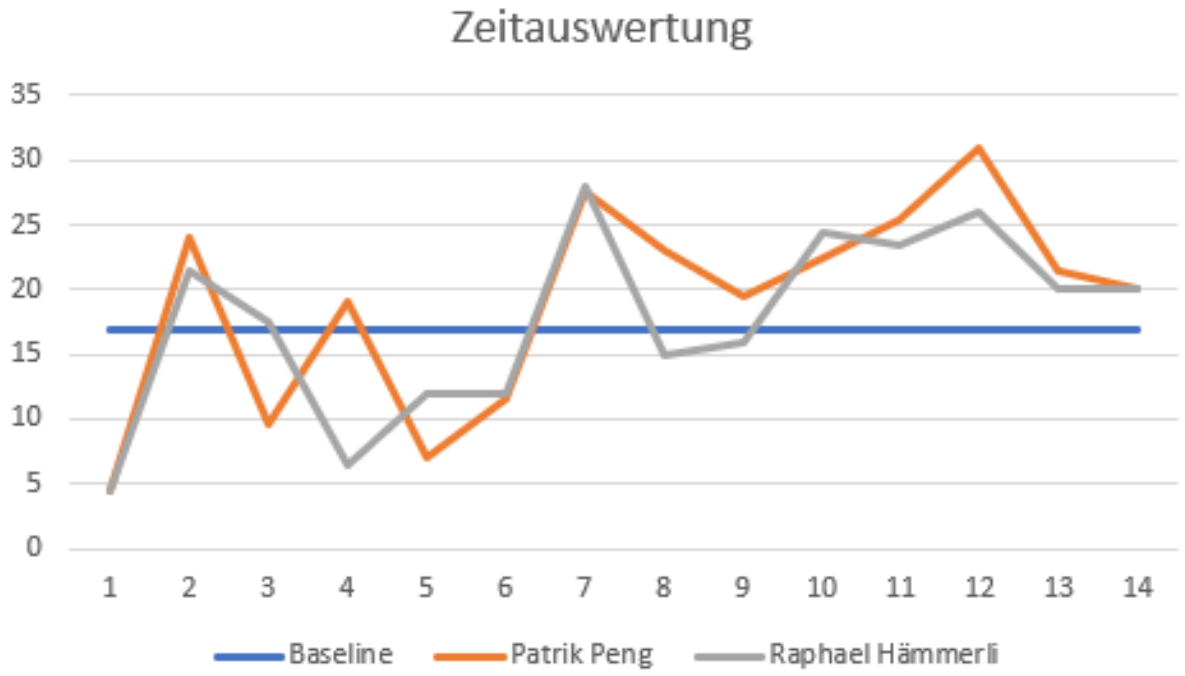

Abbildung 40: Zeitauswertung

Am Ende des Projekts hatte Patrik Peng 261.5 Stunden und Raphael Hämmerli 242.5 Stunden aufgewendet. Da Herr Stettler, unser Betreuer, in der ersten Woche nicht anwesend war und wir noch zu wenig Informationen hatten, konnten wir in dieser Woche nicht viel arbeiten. In der vierten, respektive fünften Woche hatten wir beide jeweils private Termine, weswegen nicht viel Zeit in die Arbeit investiert wurde. Dafür konnte ab Woche sieben richtig losgelegt werden, womit beide die vorgegebene Zeit erreicht haben, Patrik hat diese sogar um 20 Stunden übertroffen.

9 Schlussfolgerungen

Im Verlauf dieser Studienarbeit änderten sich verschiedene Aspekte wie Anforderungen, Technologie-Entscheidungen, die Softwarearchitektur oder Arbeitsprozesse, welche schlussendlich Einflüsse auf das Endprodukt hatten. In diesem Abschnitt der Arbeit werden diese Änderungen reflektiert, um einen Einblick zu ermöglichen, wie sich das Projekt im Laufe des Semesters verändert und entwickelt hat, sowie was beim nächsten Mal besser gemacht werden könnte.

9.1 Architektur

Im Bezug auf die Architektur der Applikation und deren Umsetzung sind ein paar Schönheitsfehler aufgetaucht, welche bei einem Neubeginn anders gelöst werden könnten.

Base Device auf Layer 1

Der Device-Controller im Rails, welcher für das CRUD der Devices verantwortlich ist, wurde layerunabhängig implementiert, sodass nicht für jeden Layer ein neuer Controller implementiert werden muss. Gleiches gilt auch für weitere Controller.

Auf jedem Layer kann ein Device ein Base Device besitzen, damit z.B. bekannt ist, auf welchem L1 Device ein L2 Device basiert. Für Layer 1 ist dies jedoch anders. Da dies der unterste Layer ist, gibt es hier keine Base Devices. Durch den layerunabhängigen Controller muss jedoch auch für Layer 1 Devices ein Base Device gesetzt werden können. Wir haben das Problem so gelöst, dass bei Layer 1 Devices das Base Device auf das Meta Device verlinkt. Dies macht von der Struktur der (Sub)Layer eigentlich Sinn, entspricht jedoch nicht hundertprozentig der Architekturdefinition.

Eine Alternative für das Problem wäre eine Fallunterscheidung im Controller gewesen, sodass Layer 1 Devices im Controller anders behandelt werden. Dies hätte dazu geführt, dass im Controller wieder layerspezifischer Code enthalten ist, was wir vermeiden wollten.

Um das Problem grundsätzlich zu lösen, sind Anpassungen an der Architektur nötig. Eine Idee ist, einen Base-Layer einzuführen, welcher den untersten Layer abbildet und die Meta Devices enthält. Layer 1 Devices hätten dann ein Base Device in diesem Base Layer verlinkt. Dies hätte jedoch zu viel Zeit in Anspruch genommen und wurde verworfen.

Verschlüsselung der Zugangsdaten

Bereits im Frühstadium der Arbeit wurde als nicht funktionale Anforderung festgelegt, dass die entwickelte Applikation die gängigen Sicherheitsstandards, welche wir bis dahin im Studium gelernt haben, einhalten muss. Das bedeutet, dass z.B. sensitive Informationen, wie Zugangsdaten für ein Netzwerkgerät, nicht im Klartext in der Datenbank gespeichert werden dürfen. Im Normalfall werden diese Daten (z.B. Benutzerpasswörter) mittels Hashfunktion und Salt gehashed und dieser Hash in der Datenbank gespeichert. Da die Hashfunktion jedoch eine Einwegfunktion ist, kann vom Hash nicht wieder auf das Passwort geschlossen werden. In unserem Fall ist dies jedoch von Nöten, da Username und Passwort bei jedem Request an ein Netzwerkgerät mitgesendet werden müssen. Daher werden diese Daten stattdessen verschlüsselt in der Datenbank gespeichert und zur Laufzeit wieder entschlüsselt, wenn sie benötigt werden.

Ursprünglich war die Idee, dass der Prozess der Ver- und Entschlüsselung möglichst unauffällig stattfindet, sodass der Entwickler normal auf diese Attribute im Code zugreifen kann. Damit dies möglich ist, müssen im Model entsprechende Callbacks definiert werden, welche ausgeführt werden, sobald

dessen Informationen aus der DB gelesen, oder in die DB geschrieben werden. Aus uns nicht bekannten Gründen, werden ein Teil dieser Callbacks nicht ausgelöst, was die gewünschte Umsetzung nicht ermöglichte. Wir vermuten, dass es sich hierbei um einen Bug in der Neo4jrb Implementierung handelt.

Das Problem wurde schlussendlich umgangen, indem Wrappermethoden implementiert wurden, welche anstelle der Callbacks aufgerufen werden. Der Unterschied zeigt sich im Code so, dass nicht mittels .password auf ein Attribut zugegriffen wird, sondern mittels .decrypted_password. Dies bringt letztendlich die gleiche Funktionalität, ausser dass der Entwickler nun eine separate Wrappermethode verwenden muss.

CRUD als Sub-Ressource der Layers

Dadurch, dass auf jedem Layer Devices, Interfaces und Connections verwaltet werden können, wird in der jetzigen Implementierung der Layer bei jedem Request an den Server als separater GET-Parameter mitgesendet. Dies kann vereinfacht werden, wenn diese Objekte als Subressource eines Layers in den Rails-Routen definiert werden. In diesem Fall wäre der Layer automatisch in der URL des Requests inkludiert und müsste nicht als separater Parameter mitgesendet werden. Diese Änderungen wären nicht schwer umsetzbar, wurden aus Zeitgründen jedoch ausgelassen.

9.2 Projektmanagement

Die relativ offene und vorerst nicht detailliert beschriebene Aufgabenstellung führte dazu, dass zu beginn des Projekts viel Zeit durch Abtasten des Scopes und Anforderungsanalyse konsumiert werden musste. Es war uns nicht klar, was das Produkt schlussendlich an Funktionen besitzen soll. Dies führte dazu, dass der komplette Projektplan und die Meilensteine mehrfach überarbeitet werden mussten und schlussendlich nicht massgebend eingehalten werden konnte.

Ursprünglich wurden Meilensteine eingeführt, welche den Abschluss einer jeweiligen Layerimplementation definierten. Bei Beginn der Implementationsphase wurde aber klar, dass vorerst die Grundimplementation mit dem Layer 1 korrekt umgesetzt werden muss. Erst anschliessend kann auf diesem aufgebaut werden und weitere Layer hinzugefügt werden.

Die Organisation im Team hat jedoch sehr gut funktioniert. Die Arbeit wurde immer auf die zwei Teammitglieder aufgeteilt und nach Erledigung zusammengeführt und überprüft. Zu jeder Zeit war klar, wer an welchem Teil arbeitet und was das Ziel bis am Ende des Tages, oder Ende der Woche ist.

9.3 Sonstige Probleme

Für die Studienarbeit wurden uns von der Schule Arbeitsplätze zur Verfügung gestellt. Diese Plätze wurden zudem mit Desktop Workstations ausgerüstet. Die zusätzlichen Monitore konnten wir an unsere Entwicklungsgeräte anschliessen und so effizienter arbeiten. Der Desktop-Computer haben wir als Production und CI Server verwendet. Diese Hardware war jedoch erst kurz vor Mitte des Projekts verfügbar.

Des Weiteren war die Verwendung von Restconf und NeXt UI nicht immer einfach. Für beide Produkte war die Dokumentation entweder nicht in grossem Ausmass vorhanden, inkorrekt, oder schwer zu finden. Ob wir beide Produkte nochmals verwenden würden, kann nicht mit Gewissheit gesagt werden.

9.4 Projektfazit

Ziel war ursprünglich ein Produkt zu erstellen, welches am Schluss fertig ist und produktiv genutzt werden kann. Dies wurde nicht erreicht. Doch wenn man beachtet, wieviel Zeit bereits die Analyse und

der Architekturentwurf benötigt haben, ist dies nicht verwunderlich. Für eine Studienarbeit war es von Beginn an eine grosse Aufgabe.

Trotzdem sind wir mit dem Stand der Dinge zufrieden. Es wurde eine solide Architektur entworfen, welche benutzbar ist. Das, was implementiert wurde, funktioniert zuverlässig und der Code wurde so umgesetzt, dass weitere Layer einfach hinzugefügt werden können. In manchen Situationen während des Projekts wurde mehr Zeit investiert und mehr Sorgfalt eingesetzt, als unbedingt nötig gewesen wäre. Dies hat sich aber bereits für uns zum Teil wieder ausbezahlt. Unser Arbeitsmotto lautete stets: „Lieber weniger umsetzen, dafür aber korrekt“.

Bezüglich dem Projektmanagement kann argumentiert werden, dass nicht alles perfekt ablief und dass im Vorhinein besser geplant hätte werden müssen. Dem stimmen wir zu, doch durch viele Unklarheiten und Technologien in welche eingearbeitet werden musste, war dies für uns nicht einfach umsetzbar.

Für die Zukunft sehen wir Verbesserungspotenzial in der Kommunikation zwischen Betreuer und Teammitglieder. Gerade zu Beginn des Projekts sollte der Hauptbetreuer bei jedem Meeting anwesend sein und von den Teammitgliedern möglichst spezifische Fragen gestellt werden. Somit soll verhindert werden, dass zu einem späteren Zeitpunkt bereits erbrachte Arbeit durch Änderungen an den Anforderungen nutzlos wird und Fehler in der Planung frühzeitig behoben werden können.

9.5 Persönliche Berichte

Raphael Hämmerli

Die Studienarbeit hat sehr viel Zeit und Mühen über das Semester benötigt, dennoch hat die ganze investierte Arbeit sich gelohnt. In den ersten paar Wochen war für mich noch nicht klar wohin die Reise gehen sollte. Spätestens ab der Hälfte waren die meisten Fragen jedoch geklärt und ab dann hatte ich viel Motivation und Spass an der Arbeit. Die Zusammenarbeit mit Patrik funktionierte reibungslos, da wir bereits vor der Studienarbeit durch verschiedene Gruppenprojekte viel Arbeit und Zeit in unsere Teambildung investiert hatten, wie zum Beispiel das Engineering Projekt. Abschliessend hat mir diese Arbeit sehr viel Spass und Freude bereitet und es war eine lehrreiche Erfahrung.

Patrik Peng

Auch wenn die Studienarbeit viel Zeit und Nerven in Anspruch genommen hat, war es dennoch eine sehr spannende und lehrreiche Erfahrung. Das Projekt bot uns die Möglichkeit, das erlernte Wissen des Studiums unter realistischen Bedingungen in der Praxis anzuwenden. Die Zusammenarbeit mit Raphael hat mir viel Freude bereitet und ich bin der Meinung, dass wir eine vernünftige Architektur entworfen und ein brauchbares Proof of Concept implementiert haben. Selbst wenn nicht alles perfekt lief, bin ich stolz auf unser Endresultat.

10 Glossar

In diesem Dokument werden oft Abkürzungen und spezifische Begriffe verwendet. Diese werden hier kurz erläutert:

Begriff	**Erklärung**
SA	Studienarbeit
Layer	Layer nach OSI-Modell, sofern nicht anders spezifiziert
Sublayer	Unterlayer eines Layers nach OSI-Modell, wird für die Informationsspeicherung verwendet
UC01	Use Case Nummer 1
NFA	Nicht funktionale Anforderung
CRUD	Die Operationen Create, Read, Update und Delete
Netzwerkknoten	Eine Komponente des Netzwerks (Switch, Router)
Kante	Eine Verbindung zwischen zwei Netzwerkinterfaces (logisch oder physisch)
Interface	Die Verbindung von Kante und Netzwerkknoten (Netzwerkport)
Einheit	Eine Einheit auf dem Layer (Netzwerkknoten, Kante oder Interface)
Job	Die Definition einer Aufgabe
Task	Eine aktuelle Ausführung eines Jobs
TaskJob	Der Name des Objekts das die Informationen zu einem Task beinhaltet
Meta Device	Repräsentiert ein physikalisches Gerät im Digital Twin
Device	Repräsentiert ein Netzwerkgerät im Digital Twin auf einem Layer
Interface	Repräsentiert ein Netzwerkinterface im Digital Twin auf einem Layer
Connection	Repräsentiert eine Verbindung zwischen zwei, oder mehr Interfaces im Digital Twin auf einem Layer

Quellen

[Bc18] Bozhidar Batsov und RuboCop contributors. *RuboCop Documentation*. 2018. URL: https://rubocop.readthedocs.io/en/latest/.

[Bri18] James Britt. *Ruby 2.5.3 API Documentation*. 2018. URL: https://ruby-doc.org/core-2.5.3/.

[BV18] CMS Distribution B.V. *Solarwinds Network Topology Mapper*. 2018. URL: http://www.cmsdistribution.com/nl/product/solarwinds-network-topology-mapper/.

[Com18] RSpec Community. *RSpec*. 2018. URL: http://rspec.info/.

[GU18] Chris Grigg und Brian Underwood. *Neo4j.rb Documentation*. 2018. URL: https://neo4jrb.readthedocs.io/en/9.2.x/.

[Han18] David Heinemeier Hansson. *Ruby on Rails Guides*. 2018. URL: https://edgeguides.rubyonrails.org/.

[Har16] Michael Hartl. *Ruby on Rails™ Tutorial: Learn Web Development with Rails, Fourth Edition*. Addison-Wesley Professional, 2016.

[Inc18a] GitHub Inc. *GitHub*. 2018. URL: https://github.com/.

[Inc18b] NGINX Inc. *NGINX*. 2018. URL: https://www.nginx.com/.

[Inc18c] Red Hat Inc. *Fedora*. 2018. URL: https://getfedora.org/de/.

[Inc18d] Stack Exchange Inc. *stack overflow*. 2018. URL: https://stackoverflow.com/.

[Jet18a] JetBrains. *RubyMine*. 2018. URL: https://www.jetbrains.com/ruby/.

[Jet18b] JetBrains. *TeamCity Documentation*. 2018. URL: https://confluence.jetbrains.com/display/TCD18/TeamCity+Documentation.

[Mar18] Myron Marston. *RSpec Core 3.8 Documentation*. 2018. URL: https://relishapp.com/rspec/rspec-core/v/3-8/docs.

[Neo18] Inc. Neo4j. *Neo4j*. 2018. URL: https://neo4j.com/.

[Opp10] Priscilla Oppenheimer. *Top-Down Network Design, Third Edition*. Cisco Press, 2010.

[Sch18] Ralph Schmieder. *Open Device Programmability*. 2018. URL: https://de.slideshare.net/CiscoDevNet/open-device-programmability-handson-intro-to-restconf-and-a-bit-of-netconf.

[Sof18] Business Skills & Software. *Network Diagram Software & Mapping Tools*. 2018. URL: https://www.businessphrases.net/network-diagram-software/.

[Zve18] Al Zverev. *NeXt-UI Tutorials*. 2018. URL: https://github.com/NeXt-UI/next-tutorials/blob/master/tutorials/tutorial-000.md.

11 Anhang

A Aufgabenstellung

Netzwerke wurden in den letzten Jahren immer grösser und neue Protokolle auf verschiedenen Layern haben die Komplexität des Gesamtsystems massiv erhöht. So ist denn auch für einen Netzwerk-Ingenieur zunehmend schwierig, auftretende Fehler im Netzwerk zu finden und zu beheben. Eine grosse Hilfe könnte ein digitaler Zwilling des Soll Zustandes sein, welcher das Netz in einwandfreiem Zustand und mit korrekter Konfiguration abbildet. In dieser Arbeit soll dafür ein generischer Ansatz entwickelt werden, wie der Zustand eines realen Netzwerkes abgebildet und in einer DB gespeichert werden kann. Treten später Fehler auf, kann der Ist-Zustand des Netzes mit dem gespeicherten Soll-Zustand des digitale Zwillings verglichen werden und so die Ursache von Fehlern schneller gefunden werden.

Diese Studienarbeit beschäftigt sich im ersten Teil mit der Abstrahierung eines realen Netzwerkes und deren Speicherung in einer Datenbank. Informationen über Komponenten, über Verbindungen zwischen Komponenten und über die Zustände der eingesetzten Protokolle sollen nach OSI Layer strukturiert abgebildet werden. Der so entwickelte Ansatz soll danach in Software umgesetzt und an einem konkreten Netzwerk überprüft werden.

Die zu erstellende Lösung soll die folgenden Funktionen unterstützen:

- Zustände und Konfiguration sollen aus realen Infrastrukturen ausgelesen, abstrahiert und in einer Datenbank abgespeichert werden.
- Der so erstellte Digitale Zwilling muss bearbeitet werden können, z.B. wenn neue Geräte hinzukommen oder Konfigurationsänderungen nötig sind.
- Der Digitale Zwilling soll mit dem echten Netzwerk verglichen und Differenzen sollen angezeigt werden können.
- Die Lösung soll über ein einfach nutzbares User-Interface verfügen.
- Der Digitale Zwilling soll dabei geräteunabhängig sein, sodass dieser nicht bei jedem Hard- oder Softwarewechsel aktualisiert werden muss.

Voraussetzung:

- Freude an Netzwerktechnologien und Protokollen.

Ort, Datum: Rapperswil, 17.17.18

Unterschrift Betreuer:

B Eigenständigkeitserklärung

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt sind oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe,
- dass ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort	**Datum**
Rapperswil	18. Dezember 2018

Unterschrift

Raphael Hämmerli	Patrik Peng
	

C Anleitungen

C.1 Hinzufügen eines neuen Layers

Um dem Digital Twin einen neuen Layer hinzuzufügen, müssen neue Klassen hinzugefügt werden. Die Klassen sind alle um den Layer-Manager gruppiert. Es müssen keine Änderungen an sonstigen Teilen des Digital Twins geändert werden. Sämtliche betroffene Klassen sind im unteren Diagramm Rot gefärbt.

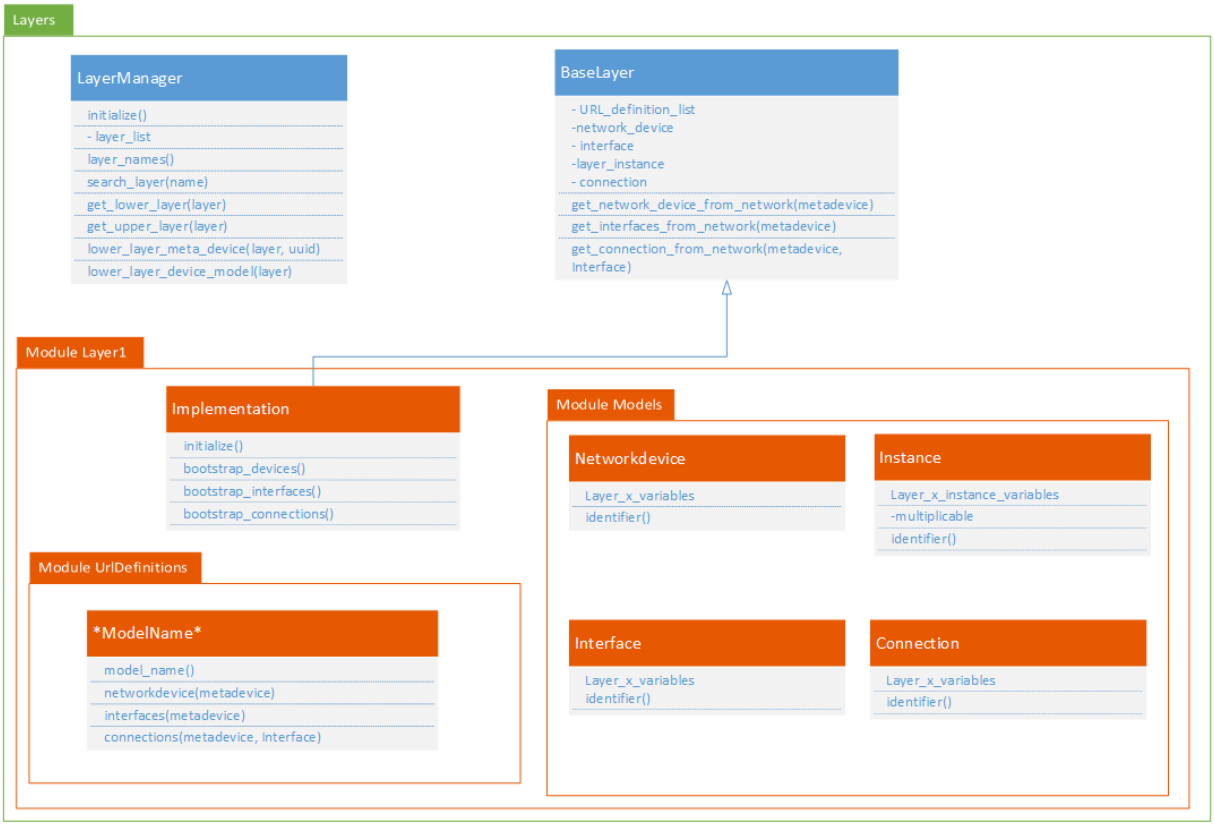

Abbildung 41: Die Klassen des Layer-Managers

C.1.1 Beschreibung neue Klassen

Die Klassen wurden schon im Architekturteil beschrieben, deswegen wird nur kurz erklärt welche Klasse was beinhalten muss. Es können auch zusätzliche Methoden oder Felder hinzugefügt werden, falls notwendig.

Module Models Das Modul *models* muss die vier Klassen *Connection*, *Device*, *Instance* und *Interface* beinhalten. Die Felder dieser Klassen können je nach Informationen die benötigt werden beliebig angepasst werden. Diese werden dann im Frontend automatisch dargestellt.

Module URLDefinitions Die URL_Definitions ermöglichen das Kommunizieren per Restconf. Pro Yang Model muss eine separate Definition erstellt werden. Die Funktion *model_name*, die den Name des Models beinhaltet ist Pflicht, die anderen sind optional.

Implementation Die Klasse *Implementation* fasst die übrigen Klassen zusammen und stellt sie den anderen Komponenten zur Verfügung. In der *initialize* Methode werden die Variabeln *network_device*, *interface*, *connection* und *layer_instance* den jeweiligen Klassen zugeordnet. Zudem wird der „name“ gesetzt und die *url_definition_list* wird mit je einer Instanz aller *URL_Definitions* gefüllt. Die drei Bootstrap-Methoden kümmern sich um die erstmalige Erstellung der Geräte über das Netzwerk, sie sind optional.

Layer Manager Der LayerManager muss nicht für jeden Layer neu erstellt werden, er dient dazu, den neuen Layer in den Rest des Digital Twins zu integrieren. Dazu muss nur in der *initialize* Methode eine neue Instanz der *Implementation*-Klasse des neuen Layers erstellt werden und der *layer_list* hinzugefügt werden.

C.1.2 Ordnerstruktur und Namespaces

Die Layer befinden sich im Ordner „\src\app\layers“, wobei jeder OSI-Layer seinen eigenen Ordner besitzt. Die Ordner der jeweiligen Sublayer sind in den OSI-Layer-Ordner verschachtelt, um für bessere Übersicht zu sorgen. Eine Orderstruktur mit den beiden OSI-Layer „Layer 1“, „Layer 2“ und dem Sublayer „Link Aggregation“ sieht wie folgt aus.

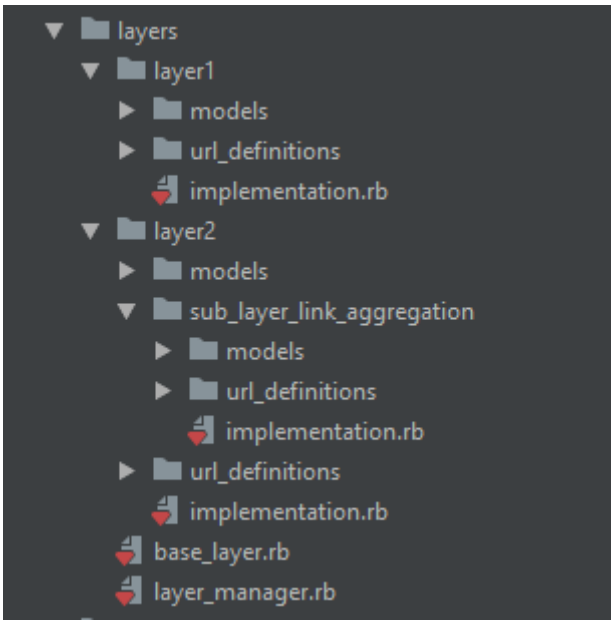

Abbildung 42: Eine Beispielstruktur

Jede Klasse wird einem Namespace zugewiesen, der vom Speicherort der Klasse abhängig ist. Zum Beispiel, die Klasse "Device"des Layer 1 ist im Pfad „\layers\Layer1\Models“, folglich besitzt es den Namespace Layer1::Models::Device. Dadurch kann jeder Layer eine Klasse „Device“ haben, da diese per Namespaces getrennt sind.

C.2 Installation & Betrieb

C.2.1 Voraussetzungen

Folgende Voraussetzungen müssen vor der Installation des Digital Twin erfüllt sein.

1. Linux oder MacOS als Betriebsystem
2. Ruby Version >= 2.5.3 (Installation via rbenv empfohlen)

C.2.2 Installation

Bei dieser Installationsanleitung wird von Ubuntu 18.04 1 LTS als Betriebsystem ausgegangen. Andere Versionen und Distributionen sollten funktionieren, wurden aber nicht getestet.

Für den Betrieb der Applikation sind gewisse Systempackages notwendig. Mit folgendem Befehl können diese unter Ubuntu installiert werden:

```
sudo apt install build-essential patch ruby-dev zlib1g-dev liblzma-dev nano
```

Anschliessend kann das ZIP-Archiv mit dem Code entpackt und mittels Kommandozeile in dessen Verzeichnis gewechselt werden.

In der Applikation ist ein Ruby-Skript enthalten, welches die Installation der Applikation vereinfachen soll. Dieses Skript kann mittels folgendem Befehl gestartet werden:

```
bin/setup
```

Development oder Production Environment

Zu Beginn wird gefragt, ob die Applikation für ein Development oder Production Environment eingerichtet werden soll. Bei einer lokalen Installation für die Entwicklung sollte das Development Environment, bei einem produktiven Betrieb das Production Environment gewählt werden. Der Unterschied zwischen den beiden Setupvarianten ist die Installation der Datenbank neo4j. Beim Produktionsbetrieb sollte diese aus Performancegründen alleinstehend installiert werden.

Anschliessend kann man den Aufforderungen des Skripts folgen. Sollte bei einem Schritt ein Fehler auftauchen, muss überprüft werden, ob die vorherigen Schritte korrekt befolgt wurden. Falls der Fehler weiterhin besteht, kann es hilfreich sein, das komplette Verzeichnis zu löschen und neu zu entpacken.

Wenn das Skript komplett und fehlerfrei durchläuft, werden am Schluss Informationen über das weitere Vorgehen gezeigt. Sind diese befolgt worden, sollte die Applikation korrekt installiert und betriebsbereit sein. Mittels folgendem Befehl können die Test gestartet und überprüft werden:

```
rspec
```

C.2.3 Betrieb der Applikation

Für den Betrieb der Applikation wird ebenfalls zwischen den beiden Environments unterschieden. Folgende Befehlszeilenkommandos sind bei der Verwendung der App von Nutzen:

Development Environment

bundle install	Neue Dependencies (Gems) installieren
rails s	Rails Server starten
rails c	Rails Konsole starten
rspec	Tests starten
rails neo4j:start	neo4j DB starten
rails neo4j:stop	neo4j DB stoppen
rails neo4j:schema:load	Datenbank Schema laden
rails neo4j:seed	Grunddaten in Datenbank laden
rails neo4j:reset_yes_i_am_sure	Datenbank komplett resetten

Production Environment

rails s -e production	Rails Server starten
RAILS_ENV=production rails c	Rails Konsole starten
RAILS_ENV=production rails neo4j:schema:load	Datenbank Schema laden
RAILS_ENV=production rails neo4j:seed	Grunddaten in Datenbank laden

Es wird empfohlen die Applikation hinter einem Reverse Proxy wie z.B. Nginx zu betreiben, damit die SSL Terminierung einfach gelöst werden kann.

Da die Datenbank standalone installiert wurde, ist das Management der Datenbank unterschiedlich und abhängig von der Installationsweise. Weitere Informationen über das Management können auf der Website von neo4j eingesehen werden.

Auf dem Produktionsserver kann die Railsapp, sowie die Datenbank mittels Systemd automatisch gestartet werden. Beispielkonfigurationen hierfür sind im Verzeichnis bin/systemd-examples gefunden werden.

C.2.4 Verwendung Digital Twin

In diesem Abschnitt wird die Verwendung des Digital Twin kurz erklärt. Es wird nicht jede Funktion im Detail erläutert, da die Bedienoberfläche relativ selbsterklärend sein sollte.

Startseite Einstiegspunkt der Applikation ist die Startseite. Es kann der gewünschte Layer per Dropdownmenü aussgewählt werden und die entsprechenden können von dieser Seite gestartet werden. Bevor diese Tasks genutzt werden können, müssen vorerst physikalische Netzwerkgeräte in Form von Meta Devices erfasst werden. Wenn die Buttons für die Bootstrap-Tasks grau sind, stehen für diesen Layer keine Bootstrap-Funktionen zur Verfügung.

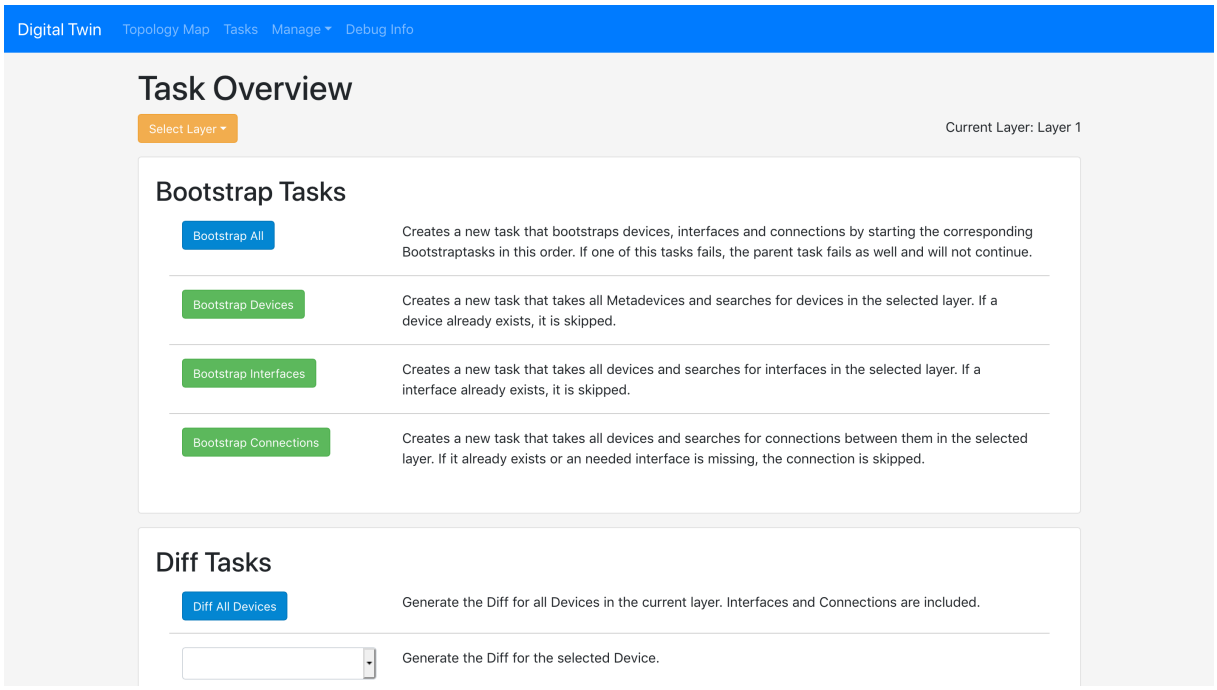

Abbildung 43: Startseite Digital Twin mit Auflistung der Bootstrap Tasks

Meta Devices erfassen In der Navigationsleiste kann unter dem Punkt Manage -> Meta Devices die Übersicht der erfassten Meta Devices eingesehen werden.

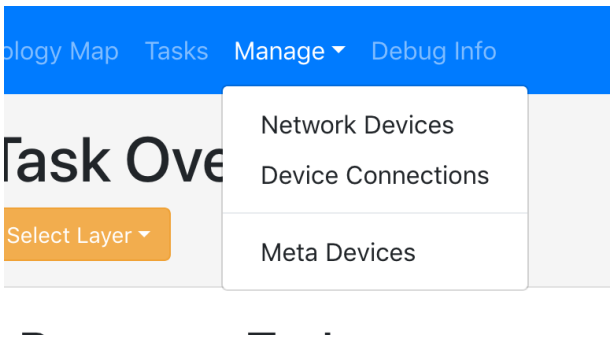

Abbildung 44: Management von Meta Devices

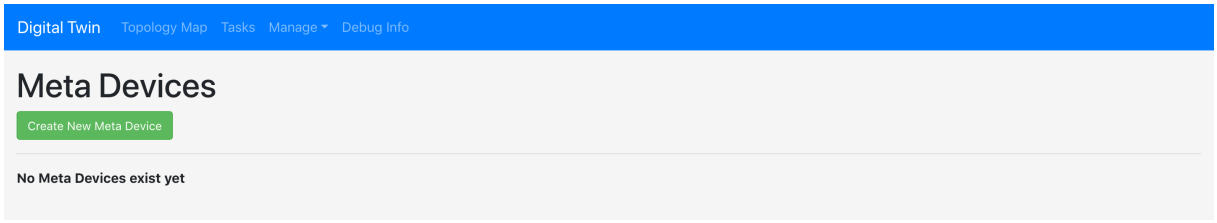

Abbildung 45: Leere Auflistung der Meta Devices

Mit Klick auf die grüne Schaltfläche kann ein neues Meta Device erfasst werden.

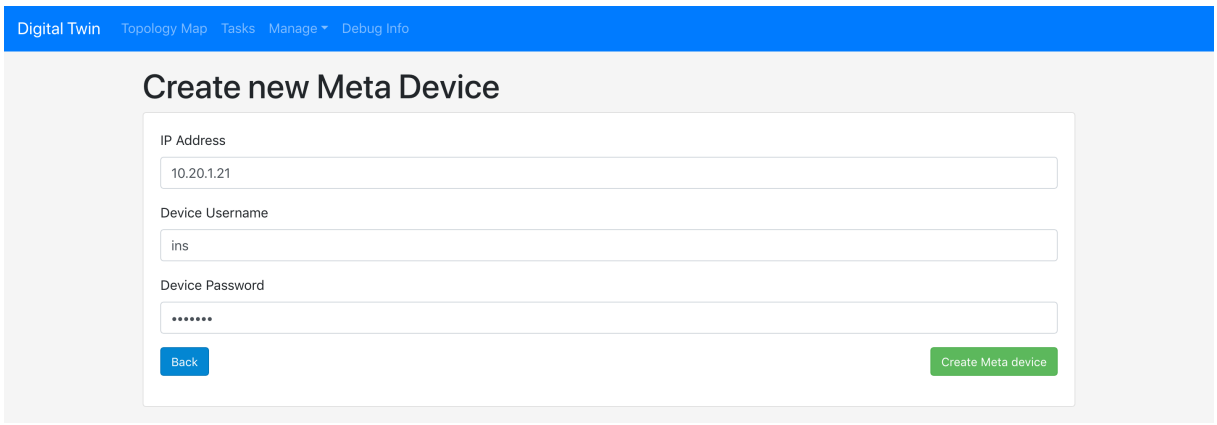

Abbildung 46: Neues Meta Device erfassen

Anschliessend werden die erstellten Meta Devices in der Übersicht aufgelistet. Zusätzlich wird überprüft, ob die Geräte vom Server via Ping erreichbar sind.

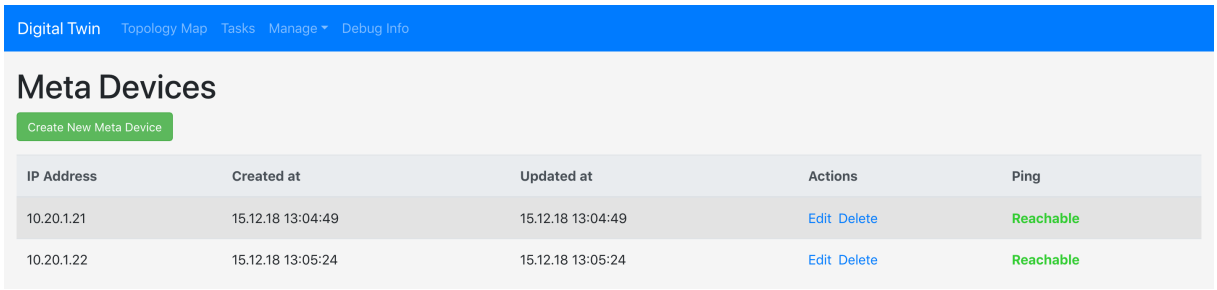

Abbildung 47: Auflistung der erfassten Meta Devices

Die Meta Devices können mittels der unter Actions dargestellten Links zusätzlich bearbeitet, oder gelöscht werden. Beim Löschen ist jedoch Vorsicht geboten, denn es werden alle Devices mit Interfaces und allfälligen Connections auch gelöscht, welche auf diesem Meta Device basieren.

Erfassung von Devices, Interfaces und Connections Wenn Meta Devices erfasst sind, können Devices, Interfaces und Connections in den entsprechenden Layern erfasst werden. Dies kann entweder automatisch mittels den bereits erwähnten Bootstrap Tasks, oder manuell via Oberfläche erreicht werden.

Die manuelle Erstellung von Devices läuft analog zu den Meta Devices ähnlich ab und kann unter Manage -> Network Devices gefunden werden. Auch hier kann mittels der grünen Schaltfläche ein neues Objekt erstellt werden. Wichtig ist hier die korrekte Auswahl des Base Devices.

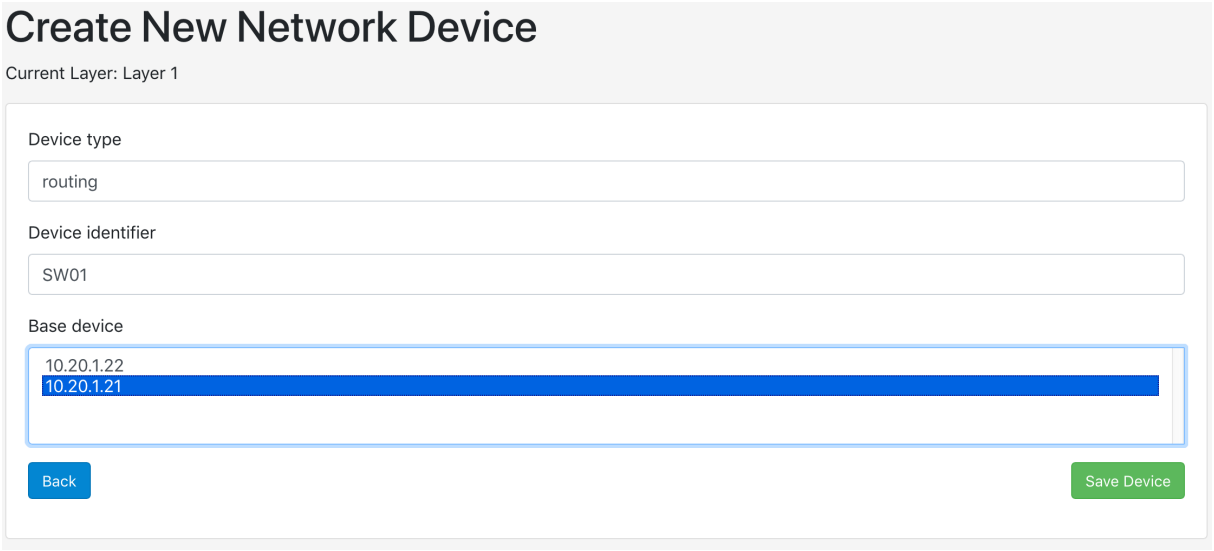

Abbildung 48: Erstellung neues Network Device

Nach der Erstellung wird das neue Device in der Übersicht dargestellt. Mittels Manage Interfaces Link in der Actions Spalte des Network Device können nun die zugehörigen Interfaces verwaltet werden.

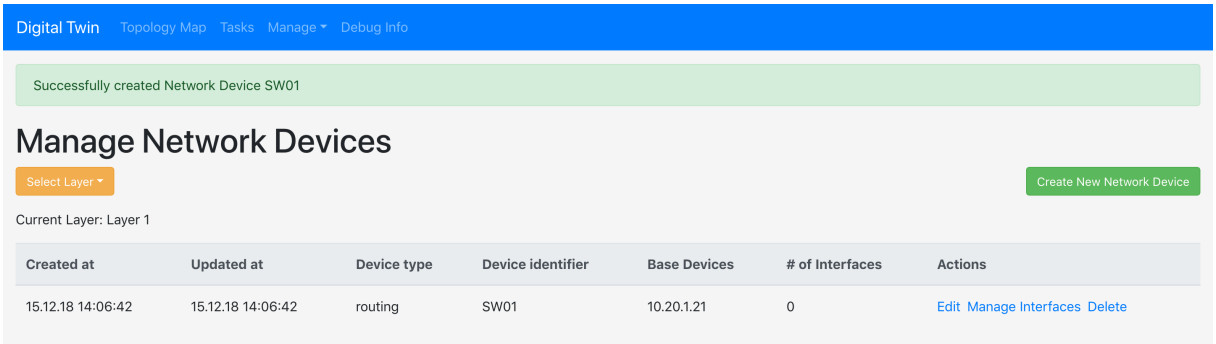

Abbildung 49: Neues Device ersichtlich, Link zu Manage Interfaces

Der weitere Prozess für das manuelle Management von Device Interfaces und Connections ist demjenigen der Network Devices identisch und wird hier nicht genäuer erläutert.

Topology Map Mittels Topology Map in der Navigationsleiste kann die erfasste Netzwerktopologie pro Layer betrachtet werden. Bei Mousehover über die Geräte und Verbindungen werden Kurzinformationen eingeblendet. Bei Mouseclick auf die Elemente werden unterhalb der Map detailierte Informationen über Devices, Interfaces und Connections eingeblendet.

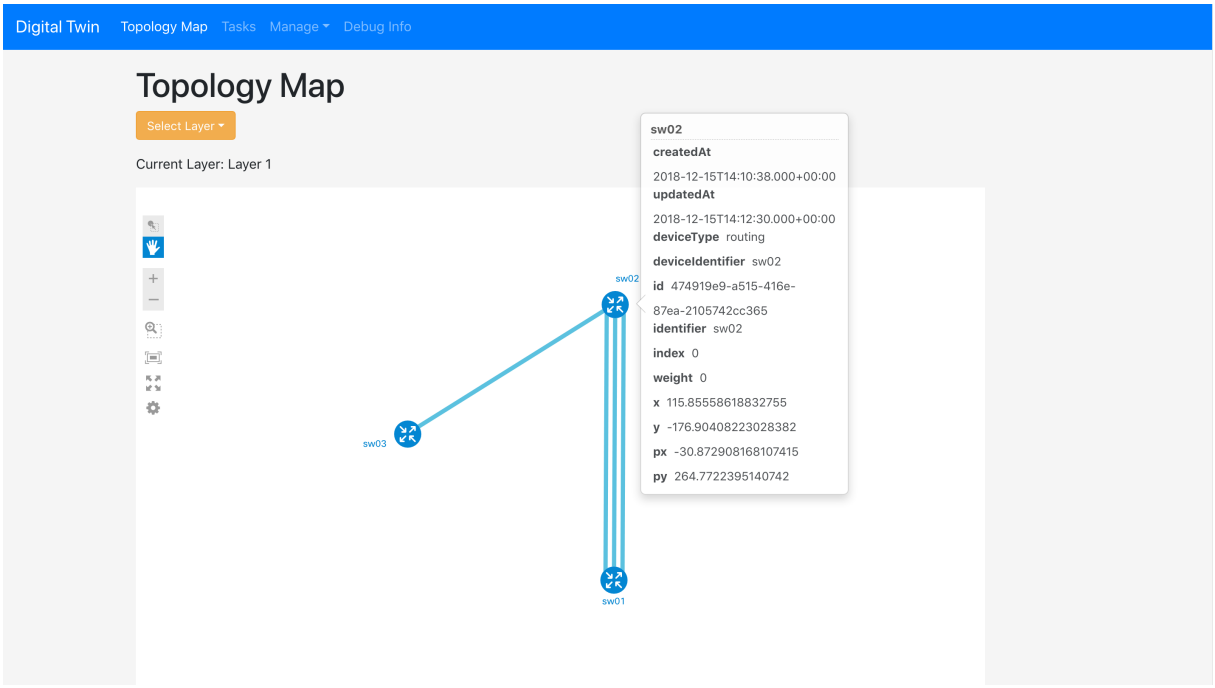

Abbildung 50: Kurzinformationen bei Mousehover

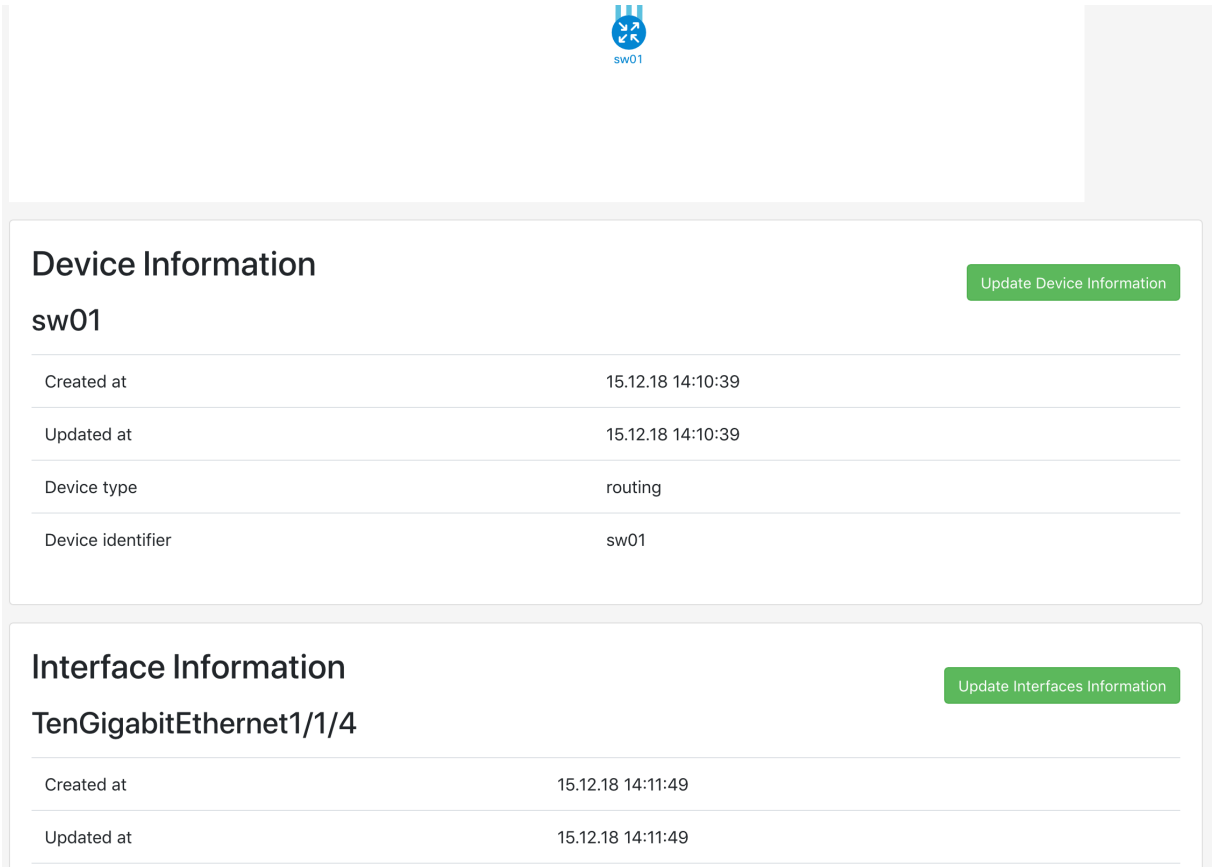

Abbildung 51: Informationen bei Mouseclick

Taskübersicht Informationen und Resultate der gestarteten Background Tasks können in der Taskübersicht eingesehen werden. Diese kann unter dem Punkt Tasks inder der Navigationsleiste erreicht werden. Mittels der Auto Refresh Checkbox wird der automatische Seitenneuaufbau alle fünf Sekunden ein oder ausgeschaltet werden.

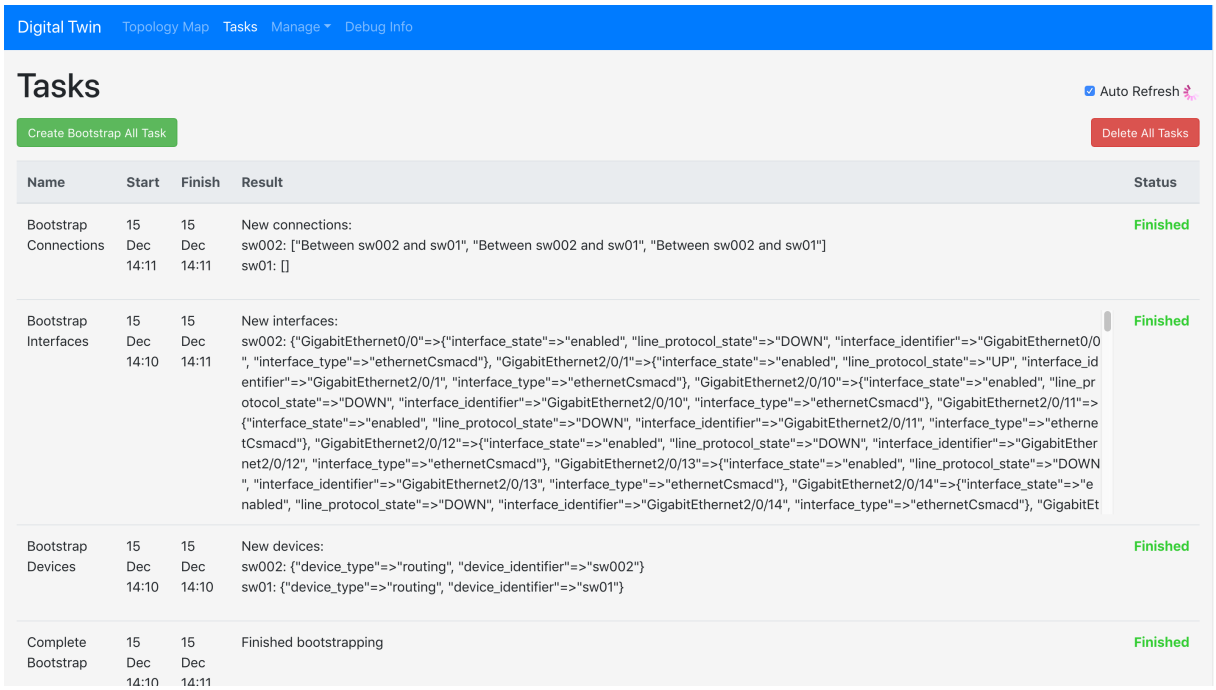

Abbildung 52: Taskübersicht

Debug Informationen Allgemeine Debug Informationen bezüglich der Applikation sind unter dem Menüpunkt Debug Info in der Navigationsleiste einsehbar.

Diese Informationen waren bei der Entwicklung und Installation der Applikation nützlich. Für den Digital Twin wurde nebst den herkömmlichen Railslogs zudem ein eigenes Logile erstellt, welches in Problemfällen behilflich sein kann. Dieses Logfile kann auf dem System unter dem Pfad ./log/digital_twin.log eingesehen werden.

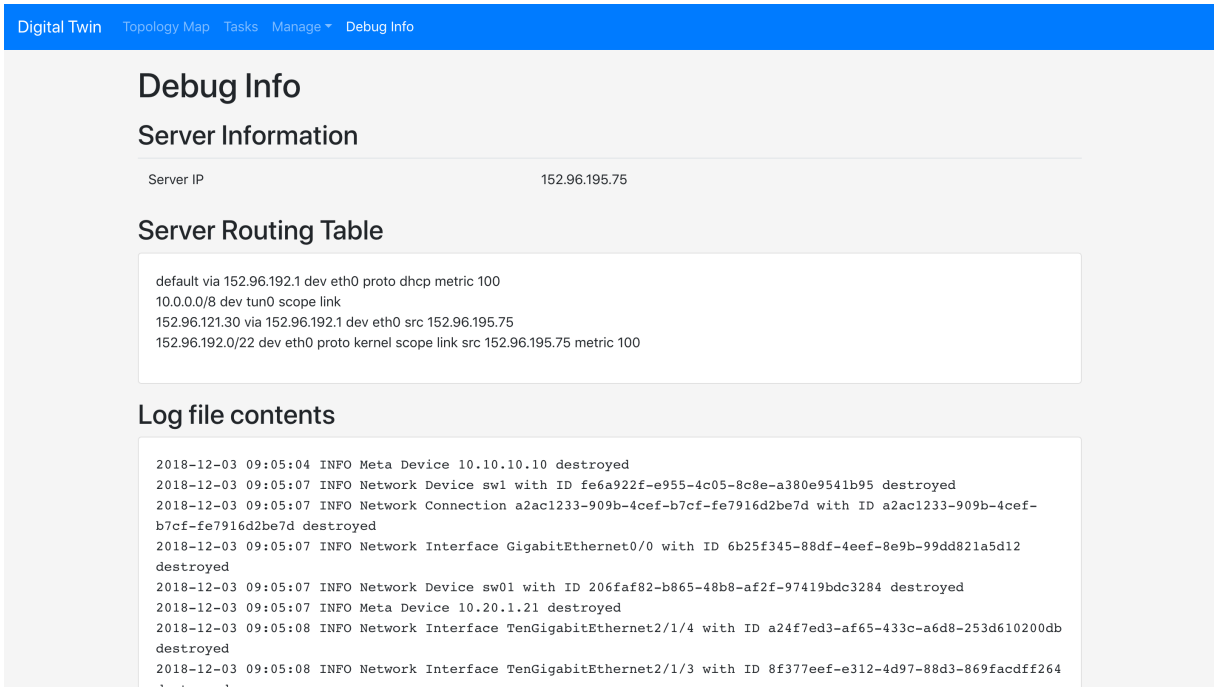

Abbildung 53: Debug Informationen

D Sitzungsprotokolle

Kick-Off

Mittwoch 19. September 11:30

Fragen vorab

- Worin liegt der Hauptfokus, das primäre Ziel?
  - Speicherung der Netzwerktopologie (1)
  - Automatisierte/manuelle Netzwerkerfassung (2)
- Kompatibilität auf Cisco beschränkt?
  - Vorerst ja
- Fancy GUI oder CLI
  - Abhängig von den Use Cases
- Falls Desktop/Mobile Clients: Cross-Plattform Support

Fazit

Es ist das Ziel das unter Umständen viele Daten pro Gerät abrufbar sein sollten: - Erlaubter Packet loss - BGP, OSPF, ... Neighbours - HSRP, Spanning Tree, VLANs - uvm.

Vermutlich Verwendung von Python als eine der Technologien

Weiteres Vorgehen

- Recherche über bereits bestehende ähnliche Lösungen
- Use Cases überlegen
- Projektplan
- Vorstellungen über Lab (was benötigen wir?)
- Mail von Stettler/Urs mit Infos zu Zeitpunkt für nächstes Meeting
- Gedanken über Art von Dokumentation

Woche 2 Meeting

Mittwoch 26. September 16:30

To discuss

- Review Projektplan
- Benötigte Hardware (Zugang zum Cisco Lab?)
- Termin nächstes Meeting (Mittwoch)
    - Nächsten Mittwoch um 9 Uhr im 8.262

Weiteres Vorgehen, was wird erwartet

- Problem analysieren und verstehen
    - Topologie aufzeichnen und überlegen was sollte gespeichert werden
    - Top-Down Network Design Lektüre (Chapter 5, 6-8)
- Analyse schreiben (Abgabe Freitag, Vorlage beachten)
- Zeitplan/Meilensteine überarbeiten (auf Montag)
- Prototyp definieren
- Research verfügbare Schnittstellen für Netzwerkgeräte

Woche 3 Meeting

Mittwoch 3. Oktober 2018 9:00 8.262

To discuss

- Raphael Zivi 22.10.2018
  - Erledigt
- Realistische Grössen von zu speichernden Netzwerken und Anzahl parallele Benutzer (NFA)
  - Wird in überarbeiteter Form von Betreuer zugesandt.
- Verfügbare Schnittstellen (Python Module, etc.)
  - NetConf, RESTconf, Fokus auf Zukunft (Urs am Nachmittag fragen)
- Zu speichernde Informationen pro Protokoll
  - Testgeräte
  - Heute Nachmittag mit Urs abklären
- Review Analyse

Review Anforderungen

- Use Cases genauer spezifizieren
  - Liste von Komponenten erstellen
  - Mandanten verwalten Netzwerke (ist aber eher Anforderung für Datenbankmodell)
  - Komponenteninformationen werden z.T. zuerst in DB eingetragen und dann erst überprüft
    Sprich Netzwerk wird zuerst in der DB gespeichert und nachfolgend mit frisch aufgebautem Netzwerk verglichen
  - Visuelle Darstellung doch im Fokus
- Abgrenzung präziser formulieren
  - Hilft bei Fehler ist zu generisch formuliert, besser zeigt nur Differenzen von Werten an
- Server client Architektur überdenken, kein zentraler Server, sondern eher standortspezifische Lösung mit Client Server Kombination, welche über Web UI ansprechbar ist
- Technische Risiken anpassen (auf neue Architektur)

Weiteres Vorgehen

- Raphael heute Nachmittag Gespräch mit Urs
  - Geräte, Fixer Arbeitsplatz, empfohlene Schnittstellen für Kommunikation zu Komponenten
- Nächste zwei Meetings am Montag nur mit Urs

Woche 4 Meeting

Montag 8. Oktober 2018 15:00 2.103

To discuss

- Grössen der zu speichernden Netzwerke (in Bezug auf NFA)
    - Grosse Netze (~6000 Komponenten)
- Review Sublayer Konzept
    - Wenn neuer Sublayer hinzukommt?
    - Änderungen im Code
    - DB Schema Migration
    - YAML File mit Sublayer Definitionen
- Review grobe Architektur (DB, Webserver, Hub)
- Benutzerauth?
    - TBD später

Technologien

Verwendung von restconf vorerst ok

Visualisierung

- neXt UI
- D3.js

Datenbank

- neo4j
- Postgres Plugin für Graphen
- DB & Yang model

Cisco IOS Version

- Welche Version supported welche Restconf Funktionen
- Restconf researchen

Weiteres Vorgehen

- Datenbanktechnologie researchen
- Gedanken über Architektur
    - Was und wie speichern wir (Sub)Layer Informationen
- YAML für Sublayerdefinition researchen

Woche 5 Meeting

Mittwoch 17. Oktober 2018 16:30 2.103

To discuss

Graph DB (neo4j)

- Architektur review
- Wie soll Vererbung gelöst werden?

Definition der SubLayer

- Wie?

Weiteres Vorgehen

- Definitiver Entscheid Technologien
- Architektur anpassen
- Dokumentation nachführen
- Prototyp weiter machen

Woche 6 Meeting

Montag 22. Oktober 2018 16:30 2.103

Anwesend: Herr Stettler, Herr Baumann, Raphael Hämmerli, Patrik Peng

To discuss

- Zeitplan diskutieren
- Review bisherige Architektur

Notizen vom Gespräch

- Sitzungsdokument genau verfassen und nach Gespräch an Betreuer senden

Architekturdokument

- Technologieauswahl genauer erörtern
- Klassen, Domainmodell, mehr Details
- Softwareentwicklungsdokumente beachten (Struktur)
    - Studienarbeitvorlagen auf dem Skripteserver
- Layer & Sublayer
    - Abhängigkeiten zwischen den Layer sind nicht genau erörtert
    - Grafiken wären schön
- Verwendung von Begriffen zu generisch (z.B. Node im Kontext von DB, oder im Kontext von Sublayer)
- Layer sauber abstrahieren, definieren, dokumentieren
- Zeitplan anpassen (Prototyp eine Woche nach hinten verschieben) -> Zeitplan senden

Weiteres Vorgehen

- Zeitplan überarbeiten, Prototyp Meilenstein eine Woche nach hinten verschieben, neue Version an Betreuer senden (bis Donnerstag)
- Architekturdokument überarbeiten
    1. Inhalt in Anforderungen verschieben und anpassen
    2. Analyse von Layer überarbeiten und klarer abstrahieren, definieren, dokumentieren
    3. Vorlagen konsultieren
- Weiterentwicklung am Prototyp
- Recherche UI Bibliothek für Darstellung im Browser

Woche 7 Meeting

Montag 29. Oktober 2018 15:00 2.103

Anwesend: Herr Stettler, Herr Baumann, Raphael Hämmerli, Patrik Peng

To discuss

- Abklärung verfügbare YANG Models für Geräte
  - Verfügbare Models vorher abrufen und verfügbare dynamisch auswählen
- SSH Timeout von Switches konfigurieren
  - Urs kümmert sich darum
- Zu speichernde Informationen pro (Sub)Layer (Review Layerdefinitionen.md)
  - MAC Adressen in eigenem Layer, da z.B. MPLS mit Labels arbeitet
  - Strukturierung der Daten graphisch visualisieren
- Interfaces ohne Link auch abspeichern?
  - Macht mehr Sinn zum diese auch zu erfassen
- Differenzierung Switchport, Routerport bei Layer 3 Switch
  - Als Layer 2 Interface Information
- Welche MAC soll bei einem Switchport (für STP) referenziert werden?
  - Management Interface MAC

Stand im Zeitplan

Betreuer hat Bedenken über Fortschritt des Projekts. Nächste Woche sollte Prototyp und Architektur fertig sein, was knapp wird.

Weiteres Vorgehen

- Nächstes Meeting am 5. November um 15:00 im 2.103
- Architektur fertigstellen
  - Sequenzdiagramm
  - Domainmodell
  - Architekturmodel
  - Layerdefinitionen sollen fertig sein
- Update Doku und an Betreuer senden
- Prototyp (Layer 1 implementation)

Woche 8 Meeting

Montag 5. November 2018 15:00 2.103

Anwesend: Herr Stettler, Herr Baumann, Raphael Hämmerli, Patrik Peng

To discuss

- Übertragungsprotokoll momentan als L1Connection und als L2Interface Attribut
  - Unterscheidung in Technologie und Enkapsulierung (Bsp. FrameRelay:HDLC)
  - Bei VLAN ist es dot1q auf L2

Feedback Architektur

- VLAN ID pro Knoten ist unnötig
- OSPF Priority pro Knoten und pro Interfaces
  - Default Wert kann überschrieben werden
  - OSPF Area handling überdenken (was wird gemapped wenn ein neues Interfaces mit neuer Area hinzugefügt wird?)
  - Generell: Layer nochmals überdenken
- Datenmodell Nord-Süd Assoziationen sind nicht gut ersichtlich

Weiteres Vorgehen

- Datenmodell anpassen, dass N-S Assoziationen ersichtlich sind
- OSPF Layer überarbeiten (was ist global, was ist pro Interface, siehe Issues oben)
- Urs sendet uns Capability-List von einem L2 Switch fürs Testing
- Build Pipeline (CI, DI)
  - Dokumentieren
- Nächstes Meeting in einer Woche selber Ort, selbe Zeit (12. November 15:00, 2.103)

Notiz bzgl. YANG Models: - Notieren welche geräteunabhängigen Models uns so fehlen

Woche 9 Meeting

Montag 12. November 2018 17:00 2.103

Anwesend: Herr Baumann, Raphael Hämmerli, Patrik Peng

To discuss

- VPN Account für Production Webserver
  - Wurde von Herr Baumann zur Verfügung gestellt
- Fokus auf Dokumentation

Weiteres Vorgehen

- Dokumentation auf den neusten Stand bringen
  - Dokumentation ganzes Buildsetup (TeamCity, GitHub Integration, Production Server)
  - Dokumentation Testing (Framework, Check in GitHub)
  - Dokumentation Linting
- Production Server fertig einrichten

Woche 10 Meeting

Montag 19. November 2018 16:30

Anwesend: Raphael Hämmerli, Patrik Peng, Herr Stettler (via Skype)

To discuss

- Diskussion momentaner Stand Dokumentation und Software

Weiteres Vorgehen

- Architektur Dokumentation anpassen und an Betreuer senden
- Weiterentwicklung am Produkt

Woche 11 Meeting

Montag 26. November 2018 16:00 2.154

Anwesend: Raphael Hämmerli, Patrik Peng, Herr Baumann, Herr Stettler (via Skype)

To discuss

- Wie sollen Änderungen an der Architektur dokumentiert werden? Die Anfangsversion anpassen (einfacher zu lesen) oder zusätzliches Kapitel mit Änderungen (zeigt Aufwand)?
    - Verschiedene Varianten möglich
    - Siehe Vorlagen, Beispiele
- Input Urs https://github.com/cisco-ie/anx

Weiteres Vorgehen

- As usual:
    - Weiterentwicklung DigitalTwin
    - Fortführung Dokumentation

Woche 12 Meeting

Montag 3. Dezember 2018 10:00 1.209

Anwesend: Herr Stettler, Raphael Hämmerli, Patrik Peng

To discuss

- Situation Meilensteine (nicht mehr realistisch)
- Definitive Requirements von Herr Stettler unterschrieben?
- Letzte Woche Entwicklungsphase

Weiteres Vorgehen

- Finalisierung Tasks
- Layer 2
- CRUD von Interfaces
- CRUD von Connections

Notiz Projektmanagement

- Zeitplan mit Meilensteinen nicht mehr so akkurat
- Meilensteine mehr basierend auf Funktionalität
- Beispielplan wie es hätte besser sein können
- Welche Fehler haben wir gemacht, was können wir bei nächsten Mal besser machen

Woche 13 Meeting

Montag 10. Dezember 2018 17:00 2.103

Anwesend: Urs Baumann, Raphael Hämmerli, Patrik Peng

To discuss

- Momentane Features

Weiteres Vorgehen

- Doku, Doku, Doku
- Aufwand der Arbeit in der Doku hervorheben

Woche 14 Meeting

Montag 17. Dezember 2018 11:30 1.262 (SA Zimmer)

Anwesend: Herr Stettler, Raphael Hämmerli, Patrik Peng

To discuss

- Benötigte Dokumente unterzeichnen
    - Aufgabenstellung
    - Rechtevereinbahrung
- Einverständniserklärung Publikation in SA?
    - Jawol
- Teamaufteilung explizit erwähnen?
    - Nein, so wie bereits erwähnt reicht
- Vorschau Abstract
    - Foto Raphael fehlt noch

Weiteres Vorgehen

- Finalisierung Arbeit