Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Claus Torben Haug

Konzeptionierung und Umsetzung eines Werkzeugs zum Vergleichen verschiedener Versionen einer Swagger-Definition

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Claus Torben Haug

Konzeptionierung und Umsetzung eines Werkzeugs zum Vergleichen verschiedener Versionen einer Swagger-Definition

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr. Ulrike Steffens

Eingereicht am: 27. Juli 2018

Claus Torben Haug

Thema der Arbeit
Konzeptionierung und Umsetzung eines Werkzeugs zum Vergleichen verschiedener Versionen einer Swagger-Definition

Stichworte
Swagger, OpenAPI, Java, TypeScript, REST

Kurzzusammenfassung
Diese Bachelorarbeit behandelt die Vergleichbarkeit von zwei Swagger-Definitionen. Es wird gezeigt, dass es möglich ist, zwei Swagger-Definitionen maschinell zu vergleichen und die Änderungen auf ihre Abwärtskompatibilität hin zu überprüfen und darsrustellen. Dafür wird zunächst die OpenAPI-Spezifikation untersucht und anschließend auf Basis der daraus gewonnen Erkenntnisse ein Prototyp entwickelt.

Claus Torben Haug

Title of the paper
Conceptual design and implementation of a tool for comparing different versions of a Swagger definition

Keywords
Swagger, OpenAPI, Java, TypeScript, REST

Abstract
This bachelor thesis deals with the comparability of two Swagger definitions. It is shown, that it is possible to compare two Swagger definitions programmatically and check the changes for their backward compatibility. First of all, the OpenAPI specification is examined and then a prototype is developed on the basis of the findings gained from it.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Im folgenden Kapitel wird ein Überblick über die Themenbeschreibung (siehe: Abschnitt 1.1), sowie die Zielsetzung (siehe: Abschnitt 1.2) und Struktur der Arbeit (siehe: Abschnitt 1.3) gegeben.

1.1 Themenbeschreibung

Der Microservice-Architekturstil erfreut sich immer größerer Beliebtheit (vgl.: Amaral u. a. (2015) Kapitel II.C). Sam Newman beantwortet die Frage, was Microservices sind, in seinem Buch „Microservices Konzeption und Design“ folgendermaßen: „Microservices sind kleine, eigenständige Services, die kollaborieren bzw. sich gegenseitig zuarbeiten“ (siehe: Newman und Lorenzen (2015) Kapitel 1.1). Um zu kollaborieren müssen diese Microservices über Schnittstellen mit einander kommunizieren. Eine Möglichkeit hierfür bietet REST (vgl.: Newman und Lorenzen (2015) Kapitel 4.1). Diese Schnittstellen sollten dokumentiert und versioniert werden, hierfür bietet Swagger ein umfangreiches Toolkit (vgl.: SmartBear Software (2018a)). Die Dokumentation in Swagger erfolgt in Swagger-Definitionen.

Nach Änderungen an den externen Schnittstellen eines Microservice ist es sowohl für die Entwickler des Service als auch Entwickler von Clients für den Service interessant, welche Änderungen erfolgten und ob diese rückwärts kompatibel sind. Um auch bei umfangreichen Swagger-Definitionen die Auswirkungen schnell erkennen zu können, wäre ein Werkzeug nützlich, dass die Änderungen aus zwei Swagger-Definitionen extrahiert und visuell übersichtlich einem Entwickler präsentiert und auch vor nicht rückwärts kompatiblen Änderungen gewarnt werden.

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist es zu zeigen, dass es maschinell möglich ist die Änderungen aus zwei Swagger-Definitionen, die gemäß der OpenAPI-Spezifikation Version 3.0.0 (vgl.: Miller und Ron (2017)) erstellt wurden, zu extrahieren, auf ihre Abwärtskompatibilität hin zu prüfen und

dies zu visualisieren. Um dies zu zeigen, soll ein Prototyp eines solchen Vergleichstools erstellt werden.

1.2.1 Abgrenzung

Ziel dieser Arbeit ist es nicht, wissenschaftlich zu untersuchen, welche Attribute einer Swagger-Definition zu einer Abwärtsinkompatibilität führen können. Des weiteren ist es nicht Ziel dieser Arbeit, dass der Prototyp die Swagger-Definitionen vollständig untersucht.

1.3 Struktur der Arbeit

Diese Arbeit ist in fünf Kapitel gegliedert, die wie folgt Aufgebaut sind:

- **Kapitel 1, Einleitung:** Es wird ein Überblick über die Themenbeschreibung, sowie die Zielsetzung und Struktur der Arbeit gegeben.
- **Kapitel 2, Grundlagen:** Es wird auf die Grundlagen, die für diese Arbeit notwendig sind, eingegangen.
- **Kapitel 3, Analyse:** Es wird die OpenAPI-Spezifikation analysiert, bestehende Software auf ihre Verwendbarkeit hin geprüft und vorhandene Swaggerbibliotheken auf ihren Nutzen für diese Arbeit untersucht.
- **Kapitel 4, Architektur und Umsetzung:** Es wird ein Architekturstil ausgewählt und dieser dann näher betrachtet. Es werden die Programmiersprache und Frameworks für das Front- und Backend, sowie der Build- und Infrastrukturtools ausgewählt. Außerdem wird die Architektur und Umsetzung des technischen und fachlichen Prototypen gezeigt.
- **Kapitel 5, Fazit:** Es gibt eine abschließende Zusammenfassung der Arbeit wieder, inklusive einer Bewertung des Ergebnisses und eines Ausblicks.

2 Grundlagen

Im folgenden Kapitel wird auf die Grundlagen, die für diese Arbeit notwendig sind, eingegangen. In Abschnitt 2.1 werden REST und HTTP und in Abschnitt 2.2 die Formate JSON und YAML erläutert. In Abschnitt 2.3 wird auf OpenAPI und Swagger und in Abschnitt 2.4 auf die Programmiersprachen Java und Typescript eingegangen.

2.1 REST und HTTP

In diesem Abschnitt werden der Architekturstil REST in Unterabschnitt 2.1.1 und das HTTP-Protokoll in Unterabschnitt 2.1.2 erläutert.

2.1.1 REST

Bei REST handelt es sich um einen Architekturstil, den Roy Thomas Fielding erstmals im Jahr 2000 in seiner Dissertation veröffentlichte (vgl.: Fielding (2000)). Bei dem Architekturstil handelt es sich um ein Client-Server-Architekturstil (vgl.: Fielding (2000) Kapitel 5.1.2), der auf 5 Grundprinzipien basiert(siehe: Tilkov (2015) Kapitel 2.2):

- Statuslose Kommunikation
- Ressourcen mit eindeutiger Identifikation
- Verknüpfungen/Hypermedia
- Unterschiedliche Repräsentationen
- Standardmethoden

Ein Übertragungsprotokoll hat ist in dem Architekturstil nicht festgelegt. Wird im weiteren Teil der Arbeit jedoch von REST gesprochen, ist damit im wesentlichen Restfull HTTP gemeint.

2.1.2 HTTP

Bei HTTP handelt es sich um ein weitgehend REST-Konformes Übertragungsprotokoll. Wenn in der Arbeit von HTTP gesprochen wird, ist vorrangig HTTP/1.1 (siehe.: Fielding u. a. (1999)) gemeint. Geschichtlich gesehen sind REST und HTTP eng verwandt, denn der REST-Architekturstil wurde von Roy Thomas Fielding entworfen, weil er die Notwendigkeit dazu während der Standardisierung von HTTP/1.1 sah (vgl.: Fielding (2000) CONCLUSIONS).
HTTP:

- ist Zustandslos (vgl.: Fielding u. a. (1999) Abstract)
- identifiziert Ressourcen eindeutig mittels URI (vgl.: Fielding u. a. (1999) Kapitel 1.1)
- Verknüpfungen können über URLs realisiert werden (vgl.: Fielding u. a. (1999) Kapitel 3.2.2)
- Http lässt offen, welcher Repräsentationstyp im Body übertragen wird, es ist dem Entwickler überlassen eine entsprechende Repräsentation zu wählen.
- HTTP stellt folgende Standardmethoden bereit (vgl.: Fielding u. a. (1999) Kapitel 9):
  - OPTIONS
  - GET
  - HEAD
  - POST
  - PUT
  - DELETE
  - TRACE
  - CONNECT
  - PATCH (wurde 1999 aus dem Standart entfernt (vgl.: Fielding u. a. (1999)))

2.2 JSON und YAML

In diesem Abschnitt werden die Formate JSON in Unterabschnitt 2.2.1 und YAML in Unterabschnitt 2.2.2 erläutert.

2.2.1 JSON

„JSON (JavaScript Object Notation) ist ein schlankes Datenaustauschformat, das für Menschen einfach zu lesen und zu schreiben und für Maschinen einfach zu parsen (Analysieren von Datenstrukturen) und zu generieren ist. Es basierd auf einer Untermenge der JavaScript Programmiersprache, Standard ECMA-262 dritte Edition - Dezember 1999.

Bei JSON handelt es sich um ein Textformat, das komplett unabhängig von Programmiersprachen ist, aber vielen Konventionen folgt, die Programmieren aus der Familie der C-basierten Sprachen (inklusive C, C++, C#, Java, JavaScript, Perl, Python und vielen anderen) bekannt sind“ (siehe: JSon.org (2018)).

2.2.2 YAML

„YAML Ain’t Markup Language“ (kurz YAML) ist eine Datenaustauschformat, das konzeptioniert wurde, um gut lesbar zu sein und mit modernen Programmiersprachen für alltägliche Aufgaben zusammenzuarbeiten (vgl.: Ben-Kiki u. a. (2009)). Im Vergleich zu JSON ist YAML stärker auf die Lesbarkeit durch Menschen ausgelegt, man kann aber sagen, das YAML eine Obermenge von JSON darstellt (vgl.: Ben-Kiki u. a. (2009)).

2.3 OpenAPI und Swagger

In diesem Abschnitt wird erläutert, was die „OpenAPI Initiative“ (siehe: Unterabschnitt 2.3.1) und „OpenAPI Specification“ (siehe: Unterabschnitt 2.3.2) ist. Des Weiteren wird in Unterabschnitt 2.3.3 festgelegt, was in dieser Arbeit unter einer Swagger-Definition verstanden wird.

2.3.1 OpenAPI Initiative

Die OpenAPI Initiative (OAI) wurde von einem Konsortium zukunftsorientierter Branchenexperten ins Leben gerufen, die den immensen Wert einer Standardisierung der Beschreibung von REST-APIs erkannt haben (vgl.: OpenAPI Initiative (2018)). Als „open governance structure“ unter der Linux Foundation konzentriert sich das OAI auf die Erstellung, Entwicklung und Förderung eines herstellerneutralen Beschreibungsformats (vgl.: OpenAPI Initiative (2018)).

2.3.2 OpenAPI-Spezifikation

Die OpenAPI-Spezifikation (kurz: OAS), früher als Swagger-Spezifikation bezeichnet, ist der weltweite Standard für die Definition von RESTful-APIs (vgl.: SmartBear Software (2018d)). Mit der OAS können Entwickler eine technologieunabhängige Schnittstelle entwickeln, die die Grundlage für die Entwicklung und den Gebrauch ihrer API bildet (vgl.: SmartBear Software (2018d)).

2.3.3 Swagger-Definition

Als Swagger-Definition wird in dieser Arbeit ein Dokument im JSON- oder YAML-Format verstanden, welches gemäß der OpenAPI-Spezifikation erstellt wurde.

2.4 Programmiersprachen

In diesem Abschnitt werden die, für diese Arbeit relevanten, Programmiersprachen erläutert. In Unterabschnitt 2.4.1 wird auf die Programmiersprache Java und in Unterabschnitt 2.4.2 auf die Programmiersprache Typescript eingegangen.

2.4.1 Java

„Nach fast 20 Jahren hat sich Java als Plattform etabliert. Über 9 Millionen Softwareentwickler verdienen weltweit mit der Sprache ihre Brötchen, 3 Milliarden Geräte führen Java-Programme aus, 1,1 Milliarden Desktops und alle Blu-ray-Player. Es gibt 10 Millionen Downloads von Oracles Laufzeitumgebung in jeder Woche, was fast 1 Milliarde Downloads pro Jahr ergibt.

[...] Als robuste objektorientierte Programmiersprache mit einem großen Satz von Bibliotheken ist Java als Sprache für Softwareentwicklung im Großen angekommen und im Bereich plattformunabhängiger Programmiersprachen konkurrenzlos“ (siehe.: Ullenboom (2014).

2.4.2 TypeScript

"TypeScript ist ein Open-Source-Projekt von Microsoft [...]. Es verfügt in Ergänzung zu Javascript über ein stärkeres Typsystem, wie man es aus Programmiersprachen wie Java oder C# kennt. Durch die starke Typisierung kann der Compiler Funktionsaufrufe besser prüfen und es können Fehler bereits zur Compilezeit erkannt werden. Außerdem können Tools den Code genauer analysieren. Dies ermöglicht Komfortfunktionen automatische Vervollständigung, Navigation zwischen Methoden und Klassen sowie eine solide Refactoring-Unterstützung.

TypeScript kompiliert in reines JavaScript [...] und ist dadurch in allen Browsern und auf allen Plattformen ausführbar"(siehe: Woiwode u. a. (2017) Kapitel 4.1).

3 Analyse

Im folgenden Kapitel wird in Abschnitt 3.1 die OpenAPI-Spezifikation analysiert, in Abschnitt 3.2 bestehende Software auf ihre Verwendbarkeit hin geprüft und in Abschnitt 3.3 vorhandene Swaggerbibliotheken auf ihren Nutzen für diese Arbeit untersucht.

3.1 Analyse der OpenAPI-Spezifikation

In diesem Abschnitt wird die OpenAPI-Spezifikation analysiert. Zunächst wird in Unterabschnitt 3.1.1 auf das Dateiformat eingegangen, in dem Swagger-Definitionen gespeichert werden. Daraufhin wird in Unterabschnitt 3.1.2 die Struktur einer Swagger-Definition erläutert und in Unterabschnitt 3.1.3 die möglichen Operationen auf diese Baumstruktur analysiert. In Unterabschnitt 3.1.4 werden dann die Kritikalitätslevel festgelegt, die für einen Knoten oder ein Blatt nach dem Anwenden einer Operation möglich sind und in Unterabschnitt 3.1.5 analysiert, welche Arten der Knotenreferenzierung innerhalb einer Swagger-Definition existieren. Abschließend wird in Unterabschnitt 3.1.6 die Baumstruktur analysiert.

3.1.1 Dateiformat

OpenAPI-Dokumente werden im JSON- oder YAML-Format abgespeichert (siehe Unterabschnitt 2.2.1 und Unterabschnitt 2.2.2) (vgl. Miller und Ron (2017)).

3.1.2 Struktur

Ein OpenAPI-Dokument ist Baumartig aufgebaut. Es gibt ein Wurzelknoten, von dem weiter Knoten bzw. Blätter abzweigen. Von jedem Knoten können wiederum weitere Knoten bzw. Blätter abzweigen. Ein Knoten muss hierbei allerdings nicht alle gemäß Spezifikation möglichen Kinder enthalten (vgl. Miller und Ron (2017)). Die Kinder eines Knotens können in einem Knoten eindeutig, an ihrem in der Spezifikation festgelegten Namen, referenziert werden (vgl. Miller und Ron (2017)). In Unterabschnitt 3.1.6 wird näher auf den Aufbau dieses Baumes eingegangen und in Unterabschnitt 3.1.3 werden mögliche Operationen auf diesen Baum betrachtet.

3.1.3 Operationen

Es gibt drei mögliche Operationen, die auf einen Knoten oder ein Blatt eines OpenAPI-Baums angewendet werden können. Ein Knoten oder ein Blatt kann gelöscht, erstellt oder verändert werden. Daraus ergeben sich vier mögliche Zustände, die ein Knoten oder ein Blatt haben kann. Ein Knoten oder Blatt kann unverändert (siehe: Unterunterabschnitt 3.1.3.1), gelöscht (siehe: Unterunterabschnitt 3.1.3.2), erstellt (siehe: Unterunterabschnitt 3.1.3.3) oder verändert (siehe: Unterunterabschnitt 3.1.3.4) sein. Es wird im weiteren davon ausgegangen, dass der linke Baum, ein Baum vor dem Anwenden keiner, einer oder mehrerer Operationen ist und der rechte Baum, der linke Baum nach dem Anwenden dieser Operationen ist.

3.1.3.1 Unverändert

Ein Knoten gilt als unverändert, wenn alle Blätter aller Äste, die von dem Knoten abzweigen, identisch sind. Des Weiteren gilt, dass keine Blätter in den Ästen hinzugekommen oder entfernt wurden. Ein Blatt gilt als unverändert, wenn sein Wert und seine Position in beiden Bäumen identisch ist. Abbildung 3.1 zeigt dies grafisch.

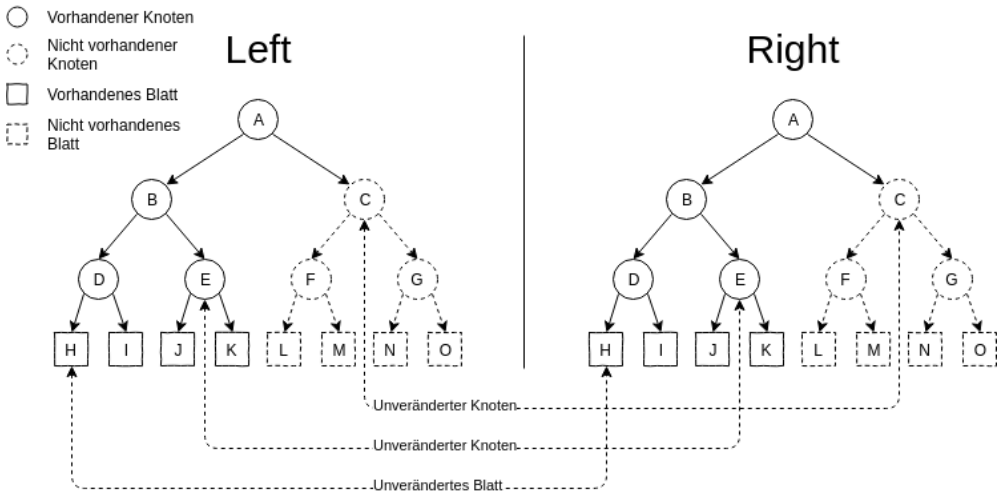

Abbildung 3.1: Unveränderter Baum

3.1.3.2 Gelöscht

Ein Knoten gilt als gelöscht, wenn alle Blätter aller Äste, die von dem Knoten abzweigen, gelöscht sind. Des Weiteren gilt, dass keine Blätter in den Ästen hinzugekommen sind. Ein

Blatt gilt als gelöscht, wenn ihm im linken Baum ein, im rechten aber kein Wert zugeordnet werden kann. Abbildung 3.2 zeigt dies grafisch.

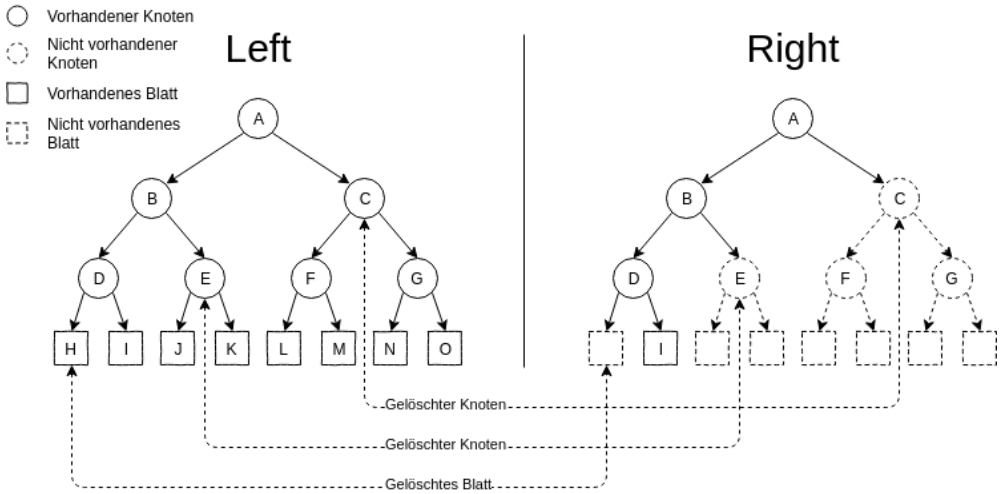

Abbildung 3.2: Baum mit gelöschten Knoten und Blättern

3.1.3.3 Erstellt

Ein Knoten gilt als erstellt, wenn alle Blätter aller Äste, die von dem Knoten abzweigen erstellt wurden. Ein Blatt gilt als erstellt, wenn ihm im linken Baum kein, im rechten aber ein Wert zugeordnet werden kann. Abbildung 3.3 zeigt dies grafisch.

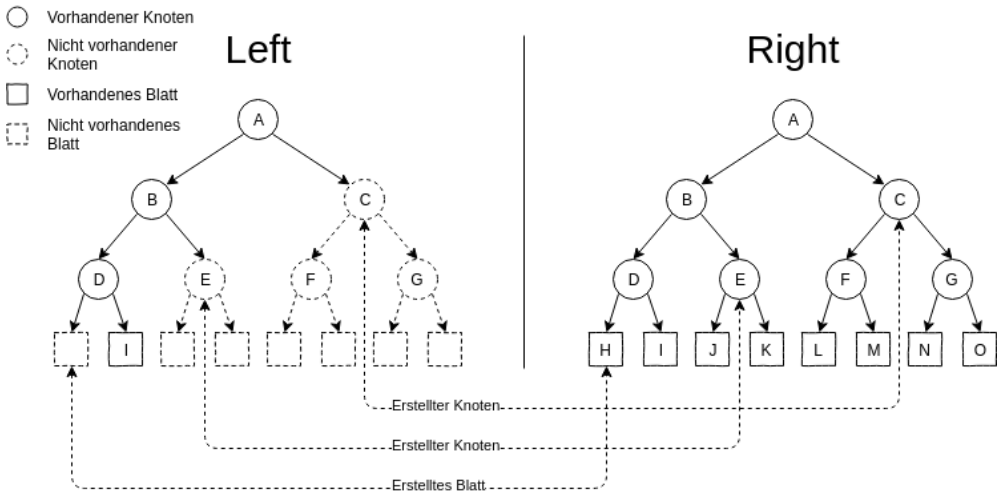

Abbildung 3.3: Baum mit erstellten Knoten und Blättern

3.1.3.4 Verändert

Ein Knoten gilt als verändert, wenn eine oder mehrere der folgenden Bedingungen gilt:

- Mindestens ein Blatt des Knotens wurde geändert.
- Mindestens ein Kind des Knotens wurde geändert.
- Mindestens ein, aber nicht alle Kinder des Knotens wurden gelöscht.
- Mindestens ein, aber nicht alle Kinder des Knotens wurden erstellt.

Im Allgemeinen kann man sagen, ein Knoten eines Baumes gilt als verändert, wenn er nicht unverändert, gelöscht oder erstellt ist. Ein Blatt gilt als verändert, wenn der Wert, der ihm zugeordnet werden kann, im linken Baum ein anderer ist, als im rechten. Abbildung 3.4 zeigt dies grafisch.

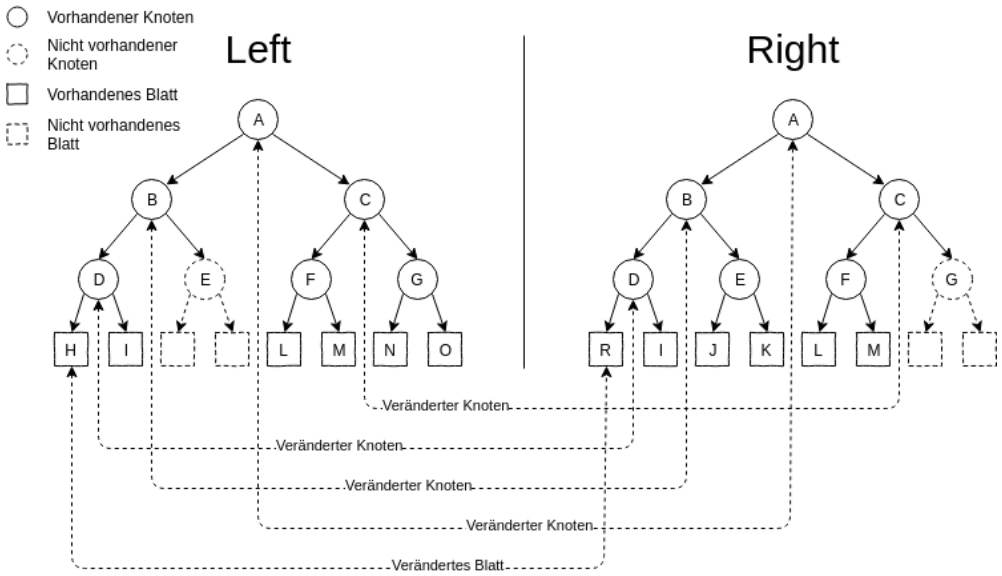

Abbildung 3.4: Baum mit veränderten Knoten und Blättern

3.1.4 Kritikalitäten

In Unterabschnitt 3.1.3 wurde beschrieben, welche Operationen auf eine OpenAPI-Baum angewendet werden können und festgelegt, in welchem Zustand sich ein Knoten oder ein Blatt des Baumes nach dem Anwenden einer Operation befindet. In diesem Abschnitt wird beschrieben, wie die Auswirkungen dieser Zustandsänderungen benannt und ermittelt werden. Zunächst werden in Unterunterabschnitt 3.1.4.1 die Kritikalitätslevel festgelegt und danach in Unterunterabschnitt 3.1.4.2 aufgezeigt, wie für Knoten und Blätter das Kritikalitätslevel ermittelt wird. Bei der Einteilung und Ermittlung der Kritikalitäten wird von der Sicht ausgegangen, dass ein bestehender Consumer, der gegen die Swagger-Definition des linken Baumes entwickelt wurde, auf die Schnittstelle eines Producers zugreift, der gemäß der Swagger-Definition des rechten Baumes entwickelt wurde.

3.1.4.1 Kritikalitätstypen

Die Kritikalitäten werden in vier Level unterteilt. Es gibt die Level None, Info, Warning und Critical, die im Weiteren näher erläutert werden.

- **None:** Es wurde keine Zustandsänderung an einem Knoten oder Blatt erkannt, oder die erkannte Zustandsänderung ist gemäß der OpenAPI-Spezifikation (siehe: Miller und Ron

(2017)) nicht als Änderung zu werten (z.B.: das setzten eines Blattes von nicht gesetzt, auf den Defaultwert).

- **Info:** Es wurde eine Zustandsänderung an einem Knoten oder Blatt erkannt, diese hat jedoch keine technischen Auswirkungen beim Zugriff auf die Schnittstelle. Der Nutzer sollte aber auf diese Änderung hingewiesen werden (z.B.: die Änderung einer Beschreibung).
- **Warning:** Es wurde eine Zustandsänderung an einem Knoten oder Blatt erkannt, diese hat jedoch wahrscheinlich keine technischen Auswirkungen beim Zugriff auf die Schnittstelle, es könnte aber Implementationsabhängig jetzt oder zukünftig zu Problemen führen (z.B.: das Ändern des Attributs Deprecated von False auf True).
- **Critical:** Es wurde eine Zustandsänderung an einem Knoten oder Blatt erkannt und diese hat technische Auswirkungen beim Zugriff auf die Schnittstelle (z.B.: das Löschen eines Pfades).

3.1.4.2 Kritikalität von Knoten und Blättern

In diesem Abschnitt wird gezeigt, wie die Kritikalität von Knoten und Blättern des rechten Baumes ermittelt wird.

3.1.4.2.1 Kritikalität von Blättern

Die Kritikalität von Blättern wird daran gemessen, welche Auswirkung die auf das Blatt erfolgte Operation auf den Elternknoten hat.

3.1.4.2.2 Kritikalität von Knoten

Die Kritikalität von Knoten wird, je nachdem in welchem Zustand sich der Knoten nach einer Operation (siehe: Unterabschnitt 3.1.3) befindet, unterschiedlich bemessen. Befindet sich der Knoten im Zustand Unverändert oder Verändert, ist die Kritikalität des Knotens die höchste Kritikalität aller Kinder dieses Knotens. Befindet sich der Knoten im Zustand Gelöscht oder Verändert, wird die Kritikalität des Knotens daran gemessen, welche Auswirkungen die erfolgte Operation auf den Elternknoten dieses Knotens hat.

3.1.5 Knotenreferenzarten

In der OpenAPI-Spezifikation lassen sich drei unterschiedliche Typen von Knotenreferenzierungsarten ermitteln (vgl. Miller und Ron (2017)). Im folgenden werden diese als direkte

Referenzierung (siehe: Unterunterabschnitt 3.1.5.1), Listenreferenzierung (siehe: Unterunterabschnitt 3.1.5.2) und Mapreferenzierung (siehe: Unterunterabschnitt 3.1.5.3) bezeichnet.

3.1.5.1 Direkte Referenzierung

Als direkte Referenzierung wird in diesem Dokument eine Referenzierung verstanden, bei der die Referenzierung eines Kindknotens im Elternknoten direkt erfolgt. Im „Path Item Object“ wird mit „get“ beispielsweise direkt ein „Operation Object“ referenziert (vgl.: Miller und Ron (2017)). Listing 3.1 zeigt ein entsprechendes Beispiel.

```
[...]
get:
    description: Beschreibung des Operation Object
    summary: Kurze Beschreibung des Operation Object
    operationId: ID des Beschreibung des Operation Object
[...]
```

Listing 3.1: Direkte Referenzierung Beispiel

3.1.5.2 Listenreferenzierung

Als Listenreferenzierung wird in diesem Dokument eine Referenzierung verstanden, bei der die Referenz im Elternknoten auf eine Liste von Kindknoten zeigt und die Kinder in dieser Liste nicht direkt eindeutig identifizierbar sind. Im weiteren werden Knoten, die diese Art der Referenzierung nutzen auch als ListNode bezeichnet. Im „Path Item Object“ wird mit „parameters“ beispielsweise eine Liste von „Parameter Object“ referenziert (vgl.: Miller und Ron (2017)). In der Regel lässt sich eine Listenreferenzierung aber zu einer Mapreferenzierung (siehe: Unterunterabschnitt 3.1.5.3) umwandeln, in dem Elemente, die den Kindknoten eindeutig machen, als Key verwendet. Dadurch werden allerdings Knoten, bei denen diese eindeutigen Elemente als Key verwendet wurden und sich eines dieser Elemente ändert, als gelöscht und neu erstellt erkannt. Im Falle des „Parameter Object“ ist dies Beispielsweise eine Kombination der Blätter „name“ und „in“ (vgl.: Miller und Ron (2017)). Listing 3.2 zeigt ein entsprechendes Beispiel.

```
[...]
parameters:
    - name: id
      in: path
      description: Kurze Beschreibung
```

```
[...]
    - name: name
      in: path
      description: Kurze Beschreibung
[...]
```

Listing 3.2: Listenreferenzierung Beispiel

3.1.5.3 Mapreferenzierung

Als Mapreferenzierung wird in diesem Dokument eine Referenzierung verstanden, bei der die Referenz im Elternknoten auf eine Key-Value-Map zeigt, wobei jedem Key in der Map eindeutig ein Kindknoten zugewiesen ist. Im weiteren werden Knoten, die diese Art der Referenzierung nutzen auch als MapNode bezeichnet. Im „Components Object“ wird mit „schemas“ beispielsweise eine Map von „Schema Object“ referenziert (vgl.: Miller und Ron (2017)). Listing 3.3 zeigt ein entsprechendes Beispiel.

```
schemas:
    Category:
        type: object
        properties:
            id:
                type: integer
                format: int64
            name:
                type: string
    Tag:
        type: object
        properties:
            id:
                type: integer
                format: int64
            name:
                type: string
```

Listing 3.3: Mapreferenzierung Beispiel

3.1.6 Aufbau des OpenAPI-Baums und Kritikalitätszuordnung im OpenAPI-Baum

In diesem Abschnitt werden die einzelnen Knoten und Blätter des OpenAPI-Baumes näher betrachtet. Es wird der Aufbau des Baumes gezeigt und die Kritikalität von Blättern und Knoten ermittelt. Die Ermittlung erfolgt anhand der in Unterabschnitt 3.1.4 festgelegten Kritikalitätsleveln und anhand der in Unterabschnitt 3.1.3 ermittelten Zuständen.

Da es nicht Teil dieser Arbeit ist, wissenschaftlich zu Untersuchen, welche Attribute einer Swagger-Definition zu einer Abwärtsinkompatibilität führen (siehe: Unterabschnitt 1.2.1), wird die Korrektheit der Zuordnung der Kritikalitätslevel angenommen, aber nicht überprüft! Des Weiteren ist es nicht Teil dieser Arbeit, dass der Prototyp eine Swagger-Definition vollständig auswertet (siehe: Unterabschnitt 1.2.1), daher wird in diesem Abschnitt nur auf die für den Prototypen relevanten Knoten und Blätter eingegangen.

Wenn sich ein Knoten oder ein Blatt im Zustand „Unverändert“ befindet, ist das Kritikalitätslevel immer „None“, da keine Änderung eingetreten ist und wird im Weiteren nicht betrachtet. Außerdem wird bei Knoten der Zustand „Verändert“ nicht betrachtet, da das Kritikalitätslevel sich in diesem Zustand aus den Kindern ergibt(siehe: Unterunterabschnitt 3.1.3.4). Wird im weiteren von allen Zuständen gesprochen, sind bei Knoten damit die Zustände „Gelöscht“ und „Erstellt“, bzw. bei Blättern die Zustände „Gelöscht“, „Erstellt“ und „Verändert“ gemeint. Der Wurzelknoten ist das „OpenAPI Object“ (siehe Unterunterabschnitt 3.1.6.1). Des Weiteren gibt es einige Blätter, die wiederholt in diversen Knoten auftreten. Wenn dies in der Beschreibung der einzelnen Knoten nicht anders definiert wird, gilt folgendes:

- **description:** Bei diesem Blatt handelt es sich um eine kurze Beschreibung des Knotens, welche mittels „Markdown“ formatiert werden kann (vgl.: Miller und Ron (2017)). Die Kritikalität dieses Blattes ist in allen Zuständen „Info“, da es sich um eine Beschreibung handelt, die nicht in der API verarbeitet wird.
- **summary:** Bei diesem Blatt handelt es sich um eine kurze Beschreibung des Knotens (vgl.: Miller und Ron (2017)). Die Kritikalität dieses Blattes ist in allen Zuständen „Info“, da es sich um eine Beschreibung handelt, die nicht in der API verarbeitet wird.
- **$ref:** Bei diesem Blatt handelt es sich um eine Referenz auf ein in den „components“ definiertes Object (vgl.: Miller und Ron (2017)). Alle Operationen auf dieses Blatt sind kritisch (Critical), eine Änderung dieser Referenz die API erheblich verändern kann.

Des Weiteren gilt, wenn ein Blatt gemäß Spezifikation vorhanden sein muss, wird auf das Löschen und Hinzufügen dieses Blattes nicht näher eingegangen, da es sich hierbei um nicht OpenAPI konforme Operationen handelt und diese Zustände somit immer die Kritikalität „Critical“ aufweisen.

3.1.6.1 OpenAPI Object

- **openapi:** Bei diesem Blatt handelt es sich um die genutzte OpenAPI -Version (vgl.: Miller und Ron (2017)). Dieses Blatt wird in dieser Arbeit nicht betrachtet.
- **info:** Dieser Knoten stellt Metadaten über die API bereit (vgl.: Miller und Ron (2017)) und ist somit nicht relevant für den Vergleich zweier Apis und wird in dieser Arbeit nicht weiter betrachtet.
- **servers:** Bei diesem Knoten handelt es sich um ein ListNode von „Server Object“ (siehe: Unterunterabschnitt 3.1.6.2) (vgl.: Miller und Ron (2017)). Als Key kann das Blatt „url“ verwendet werden. Wird ein „Server Object“ aus diesem ListNode gelöscht, ist dies eine kritische Änderung (Critical), da zu vermuten ist, dass der entsprechende Server nicht mehr existiert und vorhandene Consumer nicht mehr auf diesen zugreifen können. Das Hinzufügen eines „Server Object“ zu diesem ListNode ist jedoch unkritisch (Info), da sich noch auf die bestehenden Server berufen werden kann.
- **paths:** Bei diesem Knoten handelt es sich um ein „Paths Object“ (vgl.: Miller und Ron (2017)). Bei dem „Paths Object“ (siehe: Unterunterabschnitt 3.1.6.5) handelt es sich um eine Sonderform des MapNodes und enthält „Path Item Objects“ (siehe: Unterunterabschnitt 3.1.6.6). Das Hinzufügen von einem „Path Item Object“ zu diesem Knoten ist unkritisch (Info), da es sich hierbei um eine Erweiterung der API handelt und bestehende Consumer nicht beeinflusst. Das Löschen von einem „Path Item Object“ aus diesem Knoten hingegen ist kritisch (Critical), da bestehende Consumer diesen Pfad nutzen könnten.
- **components:** Bei diesem Knoten handelt es sich um ein „Components Object“ (siehe: Unterunterabschnitt 3.1.6.4). Es enthält wiederverwendbare Komponenten der API (vgl.: Miller und Ron (2017)). Das Hinzufügen dieses Knotens ist unkritisch (Info), da bestehende Consumer nicht darauf basieren. Das Löschen wiederum ist kritisch (Critical), da bestehende Consumer Objekte aus diesem Container verwenden könnten.

- **security:** Dieser Knoten stellt Metadaten bezüglich der Zugriffssicherung für die API bereit und ist somit zunächst nicht für die eigentliche Kompatibilität zweier Apis relevant und wird in dieser Arbeit nicht weiter betrachtet.
- **tags:** Dieser Knoten stellt Metadaten über die API bereit (vgl.: Miller und Ron (2017)) und ist somit nicht relevant für den Vergleich zweier Apis und wird in dieser Arbeit nicht weiter betrachtet.
- **externalDocs:** Bei diesem Knoten handelt es sich um ein „External Documentation Object“ (siehe: Unterunterabschnitt 3.1.6.21) (vgl.: Miller und Ron (2017)). Das Erstellen oder Löschen dieses Knotens ist unkritisch (Info), da es sich lediglich um Dokumentation handelt.

3.1.6.2 Server Object

- **url:** Bei diesem Blatt handelt es sich um einen String, der die Adresse, unter welcher der Server zu erreichen ist, darstellt (vgl.: Miller und Ron (2017)). Alle Operationen auf diesem Blatt sind kritisch (Critical), da eine Änderung der Adresse sich möglicherweise auf bestehende Consumer auswirkt.
- **variables:** Bei diesem Knoten handelt es sich um einen MapNode von „Server Variable Object“ (siehe: Unterunterabschnitt 3.1.6.3). Die Anzahl und Namen der Objekte geht hier aus dem Blatt „url“ hervor (vgl.: Miller und Ron (2017)), daher ist ein Löschen oder Erstellen eines „Server Variable Object“ nicht vorgesehen und wird somit als kritisch (Critical) betrachtet.

3.1.6.3 Server Variable Object

- **enum:** Dieses Blatt enthält eine Liste von Strings, die als Option verwendet werden können (vgl.: Miller und Ron (2017)). Das Löschen und Ändern dieser Liste ist kritisch (Critical), da bestehende Consumer diese Liste verwenden könnten. Das Hinzufügen dieses Blattes ist wahrscheinlich nicht kritisch (Warning), da bestehende Consumer vermutlich das Blatt „default“ verwenden.
- **default:** Dieses Blatt ist ein String, der den Defaultwert dieser Variable darstellt (vgl.: Miller und Ron (2017)). Alle Operationen auf diesem Blatt sind kritisch (Critical), da bestehende Consumer dieses Blatt verwenden könnten.

3.1.6.4 Components Object

- **schemas:** Bei diesem Knoten handelt es sich um ein MapNode von „Schema Object“ (siehe: Unterunterabschnitt 3.1.6.10). Dieser stellt wiederverwendbare „Schema Objects“ bereit (vgl.: Miller und Ron (2017)). Das Hinzufügen von einem „Schema Object“ zu diesem Knoten ist unkritisch (Info), da ein bestehender Consumer diesen nicht nutzt. Das Löschen ist jedoch kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **responses:** Bei diesem Knoten handelt es sich um ein MapNode von „Response Object“ (siehe: Unterunterabschnitt 3.1.6.12). Dieser stellt wiederverwendbare „Response Objects“ bereit (vgl.: Miller und Ron (2017)). Das Hinzufügen von einem „Response Object“ zu diesem Knoten ist unkritisch (Info), da ein bestehender Consumer diesen nicht nutzt. Das Löschen ist jedoch kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **parameters:** Bei diesem Knoten handelt es sich um ein MapNode von „Parameter Object“ (siehe: Unterunterabschnitt 3.1.6.8). Dieser stellt wiederverwendbare „Parameter Objects“ bereit (vgl.: Miller und Ron (2017)). Das Hinzufügen von einem „Parameter Object“ zu diesem Knoten ist unkritisch (Info), da ein bestehender Consumer diesen nicht nutzt. Das Löschen ist jedoch kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **examples:** Bei diesem Knoten handelt es sich um ein MapNode von „Example Object“ (siehe: Unterunterabschnitt 3.1.6.13). Dieser stellt wiederverwendbare „Example Objects“ bereit (vgl.: Miller und Ron (2017)). Das Löschen und Hinzufügen von einem „Example Object“ zu diesem Knoten ist unkritisch (Info), da es sich um ein Beispiel handelt und dieses nicht beim Zugriff auf eineAPIverwendet werden sollte.

- **requestBodies:** Bei diesem Knoten handelt es sich um ein MapNode von „Request Body Object“ (siehe: Unterunterabschnitt 3.1.6.14). Dieser stellt wiederverwendbare „Request Body Objects“ bereit (vgl.: Miller und Ron (2017)). Das Hinzufügen von einem „Request Body Object“ zu diesem Knoten ist unkritisch (Info), da ein bestehender Consumer diesen nicht nutzt. Das Löschen ist jedoch kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **headers:** Bei diesem Knoten handelt es sich um ein MapNode von „Header Object“ (siehe: Unterunterabschnitt 3.1.6.15). Dieser stellt wiederverwendbare „Header Objects“ bereit (vgl.: Miller und Ron (2017)). Das Hinzufügen von einem „Header Object“ zu diesem Knoten ist unkritisch (Info), da ein bestehender Consumer diesen nicht nutzt. Das Löschen ist jedoch kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **securitySchemes:** Bei diesem Knoten handelt es sich um ein MapNode von „Security Scheme Object“ (siehe: Unterunterabschnitt 3.1.6.16). Dieser stellt wiederverwendbare „Security Scheme Objects“ bereit (vgl.: Miller und Ron (2017)). Das Hinzufügen von einem „Security Scheme Object“ zu diesem Knoten ist unkritisch (Info), da ein bestehender Consumer diesen nicht nutzt. Das Löschen ist jedoch kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **links:** Bei diesem Knoten handelt es sich um ein MapNode von „Link Object“ (siehe: Unterunterabschnitt 3.1.6.19). Dieser stellt wiederverwendbare „Link Objects“ bereit (vgl.: Miller und Ron (2017)). Das Hinzufügen von einem „Link Object“ zu diesem Knoten ist unkritisch (Info), da ein bestehender Consumer diesen nicht nutzt. Das Löschen ist jedoch kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **callbacks:** Bei diesem Knoten handelt es sich um ein MapNode von „Callback Object“ (siehe: Unterunterabschnitt 3.1.6.20). Dieser stellt wiederverwendbare „Callback Objects“ bereit (vgl.: Miller und Ron (2017)). Das Hinzufügen von einem „Callback Object“ zu diesem Knoten ist unkritisch (Info), da ein bestehender Consumer diesen nicht nutzt. Das Löschen ist jedoch kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

3.1.6.5 Paths Object

Beim „Paths Object“ handelt es sich um eine Abwandlung des MapNodes. Bei diesem Typen stellt der Key die relative URI zur Ressource dar. Der Key beginnt und endet immer mit einem „/“. Er enthält beliebig viele Konstanten, die durch einfache Zeichenketten repräsentiert werden, und beliebig viele Parameter, die durch einfache Zeichenketten in geschweiften Klammern repräsentiert werden (vgl.: Miller und Ron (2017)). In Listing Listing 3.4 werden beispielhaft einige Pfade gezeigt.

```
1 /const1/
2 /const1/{var1}
3 /const2/{var1}/const1/{var2}
```

Listing 3.4: Path Example

Variablen sind allerdings nur durch Positionierung in dem Key eindeutig (vgl.: Miller und Ron (2017)). In Listing 3.5 wird dies beispielhaft dargestellt.

```
1 Different Pathes:
2 /const2/{var1}/{var2}/const1
3 /const2/{var1}/{var2}
```

```
4
5  Different Pathes, but with undefined result:
6  /const2/{var1}/{var2}/const1
7  /const2/{var1}/const1/{var2}
8
9  Equal Pathes:
10 /const2/{var1}/const1/{var2}
11 /const2/{var2}/const1/{var1}
```

Listing 3.5: Path Equality

Zwei Keys gelten als gleich, wenn sowohl alle Konstanten identisch sind und sich an der gleichen Position in der URI befinden. Außerdem müssen sich die Variablen an der gleichen Position befinden, ein Übereinstimmen der Variablennamen ist nicht notwendig (vgl.: Miller und Ron (2017)). Ändert sich eine Konstante, ist zunächst nicht unterscheidbar, ob es sich um eine Änderung oder ein Löschen und neu Hinzufügen handelt. Daher wird im weiteren davon ausgegangen, dass, wenn sich eine Konstante im Pfad ändert, es sich um ein Löschen und neu Hinzufügen handelt.

3.1.6.6 Path Item Object

- **get:** Dieser Knoten enthält ein „Operation Object“ (siehe: Unterunterabschnitt 3.1.6.9). Über dieses Objekt wird definiert, wie sich die API beim Aufrufen des Pfades mit der Http-Get-Methode (siehe: Unterabschnitt 2.1.2) verhält (vgl.: Miller und Ron (2017)). Das Hinzufügen dieses Knotens ist unkritisch (Info), da bestehende Consumer diesen nicht nutzen. Das Löschen dieses Knotens hingegen ist kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **put:** Dieser Knoten enthält ein „Operation Object“ (siehe: Unterunterabschnitt 3.1.6.9).Über dieses Objekt wird definiert, wie sich die API beim Aufrufen des Pfades mit der Http-Put-Methode (siehe: Unterabschnitt 2.1.2) verhält (vgl.: Miller und Ron (2017)). Das Hinzufügen dieses Knotens ist unkritisch (Info), da bestehende Consumer diesen nicht nutzen. Das Löschen dieses Knotens hingegen ist kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **post:** Dieser Knoten enthält ein „Operation Object“ (siehe: Unterunterabschnitt 3.1.6.9). Über dieses Objekt wird definiert, wie sich die API beim Aufrufen des Pfades mit der Http-Post-Methode (siehe: Unterabschnitt 2.1.2) verhält (vgl.: Miller und Ron (2017)). Das Hinzufügen dieses Knotens ist unkritisch (Info), da bestehende Consumer diesen

nicht nutzen. Das Löschen dieses Knotens hingegen ist kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **delete:** Dieser Knoten enthält ein „Operation Object“ (siehe: Unterunterabschnitt 3.1.6.9). Über dieses Objekt wird definiert, wie sich die API beim Aufrufen des Pfades mit der Http-Delete-Methode (siehe: Unterabschnitt 2.1.2) verhält (vgl.: Miller und Ron (2017)). Das Hinzufügen dieses Knotens ist unkritisch (Info), da bestehende Consumer diesen nicht nutzen. Das Löschen dieses Knotens hingegen ist kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **options:** Dieser Knoten enthält ein „Operation Object“ (siehe: Unterunterabschnitt 3.1.6.9). Über dieses Objekt wird definiert, wie sich die API beim Aufrufen des Pfades mit der Http-Options-Methode (siehe: Unterabschnitt 2.1.2) verhält (vgl.: Miller und Ron (2017)). Das Hinzufügen dieses Knotens ist unkritisch (Info), da bestehende Consumer diesen nicht nutzen. Das Löschen dieses Knotens hingegen ist kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **head:** Dieser Knoten enthält ein „Operation Object“ (siehe: Unterunterabschnitt 3.1.6.9). Über dieses Objekt wird definiert, wie sich die API beim Aufrufen des Pfades mit der Http-Head-Methode (siehe: Unterabschnitt 2.1.2) verhält (vgl.: Miller und Ron (2017)). Das Hinzufügen dieses Knotens ist unkritisch (Info), da bestehende Consumer diesen nicht nutzen. Das Löschen dieses Knotens hingegen ist kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **patch:** Dieser Knoten enthält ein „Operation Object“ (siehe: Unterunterabschnitt 3.1.6.9).Über dieses Objekt wird definiert, wie sich die API beim Aufrufen des Pfades mit der Http-Patch-Methode (siehe: Unterabschnitt 2.1.2) verhält (vgl.: Miller und Ron (2017)). Das Hinzufügen dieses Knotens ist unkritisch (Info), da bestehende Consumer diesen nicht nutzen. Das Löschen dieses Knotens hingegen ist kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **trace:** Dieser Knoten enthält ein „Operation Object“ (siehe: Unterunterabschnitt 3.1.6.9). Über dieses Objekt wird definiert, wie sich die API beim Aufrufen des Pfades mit der Http-Trace-Methode (siehe: Unterabschnitt 2.1.2) verhält (vgl.: Miller und Ron (2017)). Das Hinzufügen dieses Knotens ist unkritisch (Info), da bestehende Consumer diesen nicht nutzen. Das Löschen dieses Knotens hingegen ist kritisch (Critical), da bestehende Consumer diesen nutzen könnten.

- **servers:** Bei diesem Knoten handelt es sich um eine ListNode von „Server Object“ (siehe: Unterunterabschnitt 3.1.6.2). Als Key kann das Blatt „url“ verwendet werden (vgl.: Miller und Ron (2017)). Wird ein „Server Object“ aus diesem ListNode gelöscht, ist dies eine kritische Änderung (Critical), da zu vermuten ist, das der entsprechende Server nicht mehr existiert und vorhandene Consumer nicht mehr auf diesen zugreifen können. Das Hinzufügen von einem „Server Object“ zu diesem ListNode ist jedoch unkritisch (Info), da sich noch auf die bestehenden Server berufen werden kann.

- **parameters:** Dieser Knoten ist ein ListNode von „Parameter Object“ (siehe: Unterunterabschnitt 3.1.6.8) (vgl.: Miller und Ron (2017)). Die Besonderheiten dieses Knotens werden in Unterunterabschnitt 3.1.6.7 näher betrachtet. Unter der Annahme, dass es nicht standardisiert ist, wie sich ein Producer verhält, wenn er zu viele Parameter übergeben bekommt, ist das Löschen von einem „Parameter Object“ aus diesem Knoten kritisch (Critical). Das Hinzufügen von Parametern ist kritisch (Critical), wenn der Parameter nicht optional ist, ansonsten ist das Hinzufügen unkritisch. Es könnte sich aber die Funktionalität verändert haben, wenn ein nicht optionaler Parameter hinzukommt (Warning).

3.1.6.7 Parameters Object

Das „Parameters Object“ ist nicht Bestandteil der OpenAPI-Spezifikation (vgl.: Miller und Ron (2017)), es dient hier zur Veranschaulichung des MapNodes, der aus einem ListNode von „Parameter Object“ entsteht. Im Allgemeinen kann man sagen, dass als Key für den MapNode eine Kombination aus den Blättern „name“ und „in“ genutzt werden kann. Handelt es sich bei dem „Parameter Object“ allerdings um einen Pfadparameter, gibt es zwei möglich Ansätze, dieses Objekt zu Referenzieren. Einerseits kann man von dem Standpunkt ausgehen, dass die Position des Parameternamens im Pfad (siehe: Unterunterabschnitt 3.1.6.5) ausschlaggebend zur Identifizierung von einem „Parameter Object“ ist. Denn wenn ein Parameter an der gleichen Stelle des Pfades im linken Baum einen anderen Namen hat, als im rechten Baum, kann es sich trotzdem um den gleichen Parameter handeln. Listing 3.6 und Listing 3.7 zeigen Beispielhaft, dass bei einer Namensänderung eines Parameters eigentlich keine Änderung der API eingetreten ist.

```
1 [...]
2 paths:
3     /api/{test}:
4 [...]
5         parameters:
```

```
            - in: path
              name: test
[...]
```

Listing 3.6: Linker Baum bei der Referenzierung von Parametern über ihre Position

```
[...]
paths:
    /api/{test2}:
[...]
        parameters:
            - in: path
              name: test2
[...]
```

Listing 3.7: Rechter Baum bei der Referenzierung von Parametern über ihre Position

Andererseits kann man aber auch von dem Standpunkt ausgehen, dass der Name das Ausschlaggebende für die Identifizierung eines Pfadparameters ist. Listing 3.8 und Listing 3.9 zeigen beispielhaft, dass sich lediglich die Position der Parameter im Pfad, aber nicht die Parameter verändert haben. Geht man bei diesem Beispiel allerdings davon aus, dass die Referenzierung der Parameter über ihre Position im Pfad erfolgt, würden beide Parameter als verändert erkannt werden. Allerdings würde der Pfad auch nicht als verändert erkannt werden (siehe: Unterunterabschnitt 3.1.6.5).

```
[...]
paths:
    /api/{test1}/{test2}:
[...]
        parameters:
            - in: path
              name: test1
[...]
            - in: path
              name: test2
[...]
```

Listing 3.8: Linker Baum bei der Referenzierung von Parametern über ihren Namen

```
[...]
paths:
```

```
3     /api/{test2}/{test1}:
4 [...]
5         parameters:
6             - in: path
7               name: test1
8 [...]
9             - in: path
10              name: test2
11 [...]
```

Listing 3.9: Rechter Baum bei der Referenzierung von Parametern über ihren v

Da, wenn man die Pfadparameter über ihren Name referenziert, erhebliche Änderungen an der API nicht erkannt würden (z.B., wie im obigen Beispiel gezeigt, das Vertauschen von Parametern im Pfad), wird im Weiteren davon ausgegangen, dass die Pfadparameter aufgrund ihrer Position im Pfad referenziert werden.

3.1.6.8 Parameter Object

- **name:** Dieses Blatt enthält einen String, der den Namen des Parameters beschreibt, der beim Zugriff auf die API verwendet wird (vgl.: Miller und Ron (2017)). Enthält das Blatt „in“ den Wert „path“, sind Änderungen an diesem Blatt unkritisch (Info), da die Zuordnung des Parameters über die Position im Pfad erfolgt. Andernfalls sind Änderungen an diesem Blatt kritisch (Critical), da der Parameter nicht mehr unter seinem alten Namen zuzuordnen ist.

- **in:** Dieses Blatt bestimmt, über welchen Weg ein Parameter übergeben wird (vgl.: Miller und Ron (2017)). Änderungen an diesem Blatt sind kritisch, da sich der Übertragungsweg geändert hat.

- **required:** Dieses Blatt enthält einen Boolean, der bestimmt, ob ein Parameter gesetzt werden muss. Wenn dieses Blatt nicht existiert, ist sein Wert „False“ (vgl.: Miller und Ron (2017)). Änderungen von „False“ auf „True“ sind kritisch (Critical), da bestehende Consumer die API möglicherweise nicht mehr korrekt bedienen können. Änderungen von „True“ auf „False“ sind unkritisch (Info), da bestehende Consumer die API weiterhin korrekt bedienen können.

- **deprecated:** Dieses Blatt enthält einen Boolean, der bestimmt, ob ein Parameter veraltet ist. Aus der OpenAPI-Spezifikation geht kein Default-Wert hervor (vgl.: Miller und Ron

(2017)). Im Weiteren wird davon ausgegangen, dass wenn dieses Blatt nicht existiert, sein Wert „False“ ist. Änderungen von „False“ auf „True“ sind zwar unkritisch, können aber später technische Probleme nach sich ziehen (Warning), da bestehende Consumer die API zwar in dieser Version noch korrekt bedienen können, in einer späteren aber möglicherweise nicht mehr. Änderungen von „True“ auf „False“ sind unkritisch (Info), da bestehende Consumer die API weiterhin korrekt bedienen können.

- **allowEmptyValue:** Dieses Blatt enthält einen Boolean, der bestimmt, ob ein Parameter einen leeren Wert enthalten darf. Wenn dieses Blatt nicht existiert, ist sein Wert „False“ (vgl.: Miller und Ron (2017)). Änderungen von „True“ auf „False“ sind kritisch (Critical), da bestehende Consumer die API möglicherweise nicht mehr korrekt bedienen können. Änderungen von „False“ auf „True“ sind unkritisch (Info), da bestehende Consumer die API weiterhin korrekt bedienen können.

- **style:** Dieses Blatt bestimmt, in welcher Form der Parameter an den Producer übergeben wird (vgl.: Miller und Ron (2017)). Das Erstellen oder Löschen dieses Blattes mit seinem Default-Wert in Abhängigkeit zum „in“-Blatt ist unkritisch (Info), alle anderen Operationen sind kritisch (Critical), da bestehende Consumer die API möglicherweise nicht mehr korrekt bedienen können.

- **explode:** Dieses Blatt enthält einen Boolean, der bestimmt, ob ein Parameter, der eine Map oder ein Array enthält, zu einzelnen Parametern, je in dem Objekt enthaltenem Wert, umgewandelt wird (vgl.: Miller und Ron (2017)). Da es sich hierbei um eine rein serverseitige Option handelt, sind alle Operationen auf diesem Blatt unkritisch (Info).

- **allowReserved:** Dieses Blatt enthält einen Boolean, der bestimmt, ob Sonderzeichen in dem Wert des Parameters escaped werden müssen. Der Default-Wert dieses Blattes ist „false“ (vgl.: Miller und Ron (2017)). Das Löschen und Hinzufügen dieses Blattes ist unkritisch (Info) wenn der Wert dem Default-Wert entspricht. Alle anderen Operationen sind kritisch, da der Consumer die API möglicherweise nicht mehr korrekt bedienen kann.

- **schema:** Dieser Knoten enthält ein „Schema Object“ (siehe: Unterunterabschnitt 3.1.6.10), welches den Typen des Parameters definiert (vgl.: Miller und Ron (2017)). Das Löschen und Hinzufügen dieses Knotens sind kritisch (Critical), da dies eine erhebliche Änderung an der API zur Folge haben kann.

- **example:** Dieses Blatt enthält ein beliebiges Objekt, welches ein Beispiel für die Nutzung dieses Parameters darstellt (vgl.: Miller und Ron (2017)). Alle Operationen auf diesem Blatt sind unkritisch (Info), da das Beispiel keinen Einfluss auf die API hat.
- **examples:** Dieser Knoten ist ein MapNode von „Example Object“ (siehe: Unterunterabschnitt 3.1.6.13), welcher ein Beispiel für die Nutzung dieses Parameters enthält (vgl.: Miller und Ron (2017)). Alle Operationen auf diesem Knoten sind unkritisch (Info), da die Beispiele keinen Einfluss auf die API haben.
- **content:** Dieser Knoten ist ein MapNode von „Media Type Object“ (siehe: Unterunterabschnitt 3.1.6.22), welcher eine Art erweiterten Satz an Regeln für das Serialisieren von Parametern bereitstellt (vgl.: Miller und Ron (2017)). Alle Operationen auf diesem Knoten sind kritisch (Critical), da das Hinzufügen oder Entfernen dieses Knotens erheblichen Einfluss auf die API haben kann.

3.1.6.9 Operation Object

- **tags:** Dieses Blatt enthält eine Liste von Strings (vgl.: Miller und Ron (2017)). Alle Operationen auf diesem Blatt sind unkritisch (Info), da es sich hierbei lediglich um Metadaten handelt.
- **externalDocs:** Bei diesem Knoten handelt es sich um ein „External Documentation Object“ (siehe: Unterunterabschnitt 3.1.6.21) (vgl.: Miller und Ron (2017)). Das Erstellen oder Löschen dieses Knotens ist unkritisch (Info), da es sich lediglich um Dokumentation handelt.
- „operationId:“ Bei diesem Blatt handelt es sich um einen eindeutigen Identifier für diese Operation (vgl.: Miller und Ron (2017)). Alle Operationen auf diesem Blatt sind für den Consumer zunächst unkritisch. Da dieses Blatt aber von externen Tools genutzt werden könnte, ist sein Kritikalitätslevel „Warning“.
- **parameters:** Dieser Knoten ist ein ListNode von „Parameter Object“ (siehe: Unterunterabschnitt 3.1.6.8) (vgl.: Miller und Ron (2017)). Die Besonderheiten dieses Knotens werden in Unterunterabschnitt 3.1.6.7 näher betrachtet. Unter der Annahme, dass es nicht standardisiert ist, wie sich ein Producer verhält, wenn er zu viele Parameter übergeben bekommt, ist das Löschen eines „Parameter Object“ aus diesem Knoten kritisch (Critical). Das Hinzufügen von Parametern ist kritisch (Critical), wenn der Parameter nicht optional ist, ansonsten ist das Hinzufügen unkritisch. Es könnte sich aber die Funktionalität verändert haben, wenn ein nicht optionaler Parameter hinzukommt (Warning).

- **requestBody:** Dieser Knoten enthält ein „Request Body Object“ welches das Format des Bodys bei Anfragen an die API bestimmt (vgl.: Miller und Ron (2017)). Alle Operationen auf diesen Knoten sind kritisch (Critical), da das Hinzufügen dieses Knotens die API erheblich verändert. Unter der Annahme, dass nicht näher definiert ist, wie sich ein Producer verhält, wenn er einen „Request Body“ übergeben bekommt, obwohl keiner erwartet wird, ist das Löschen auch kritisch (Critical).

- **responses:** Dieser Knoten enthält ein „Responses Object“ (siehe: Unterunterabschnitt 3.1.6.11), wobei das „Responses Object“ eine Sonderform des MapNodes von „Response Object“ (siehe: Unterunterabschnitt 3.1.6.12) ist (vgl.: Miller und Ron (2017)). Das Hinzufügen von einem „Response Object“ zu diesem Knoten ist kritisch (Critical), da bestehende Consumer diesen Response möglicherweise nicht verarbeiten können. Das Löschen eines „Response Object“ aus diesem Knoten ist unkritisch (Info), da bestehende Consumer dadurch nicht beeinflusst werden.

- **callbacks:** Dieser Knoten ist ein MapNode von „Callback Object“ und enthällt die möglichen Callbacks, die ein Producer ausführen kann (vgl.: Miller und Ron (2017)). Das Löschen von einem „Callback Object“ aus diesem Knoten ist kritisch, da bestehende Consumer möglicherweise auf den Callback warten könnten. Das Hinzufügen ist ebenfalls kritisch, da bestehende Consumer nicht auf diesen Callback reagieren könnten.

- **deprecated:** Dieses Blatt enthält einen Boolean, der bestimmt, ob ein „Operation Object“ veraltet ist. Aus der OpenAPI-Spezifikation geht kein Default-Wert hervor. Im Weiteren wird davon ausgegangen, dass wenn dieses Blatt nicht existiert, sein Wert „False“ ist (vgl.: Miller und Ron (2017)). Änderungen von „False“ auf „True“ sind zwar unkritisch, können aber später technische Probleme nach sich ziehen (Warning), da bestehende Consumer die API zwar in dieser Version noch korrekt bedienen können, in einer späteren aber möglicherweise nicht mehr. Änderungen von „True“ auf „False“ sind unkritisch (Info), da bestehende Consumer die API weiterhin korrekt bedienen können.

- **security:** Dieser Knoten enthält eine Liste von „Security Requirement Object“ (vgl.: Miller und Ron (2017)). Das Vergleichen von Sicherheitsaspekten ist nicht Teil dieser Arbeit, daher wird angenommen, dass es sich bei diesem Blatt um einen Knoten Handelt und alle Operationen kritisch (Critical) sind.

- **servers:** Bei diesem Knoten handelt es sich um eine ListNode von „Server Object“ (siehe: Unterunterabschnitt 3.1.6.2). Als Key kann das Blatt „url“ verwendet werden (vgl.: Miller und Ron (2017)). Wird ein „Server Object“ aus diesem ListNode gelöscht, ist dies eine

kritische Änderung (Critical), da zu vermuten ist, dass der entsprechende Server nicht mehr existiert und vorhandene Consumer nicht mehr auf diesen zugreifen können. Das Hinzufügen von einem „Server Object“ zu diesem ListNode ist jedoch unkritisch (Info), da sich noch auf die bestehenden Server berufen werden kann.

3.1.6.10 Schema Object

Das „Schema Object“ dient zur Definition von Ein- und Ausgabedatentypen (vgl.: Miller und Ron (2017)). In dieser Arbeit wird angenommen, das es sich hierbei um ein Blatt handelt. Die Kritikalitätslevel für das Erstellen und Löschen dieses Blattes werden in den Elternobjekten festgelegt. Ändert sich ein bestehendes „Schema Object“, ist das Kritikalitätslevel „Critical“, da möglicherweise eine erhebliche Änderung an der API durchgeführt wurde.

3.1.6.11 Responses Object

Das „Responses Object“ ist eine Sonderform des MapNodes. Bei diesem MapNode wird mit einem Key ein „Respone Object“ (siehe: Unterunterabschnitt 3.1.6.12) referenziert. Der Key ist dabei entweder der String „default“ oder ein „HTTP Status Code“ (siehe: Fielding u. a. (1999)) (vgl.: Miller und Ron (2017)).

3.1.6.12 Response Object

- **headers:** Bei diesem Knoten handelt es sich um einen MapNode von „Header Object“ (siehe: Unterunterabschnitt 3.1.6.15) (vgl.: Miller und Ron (2017)). Das Löschen von einem „Header Object“ aus diesem Node ist unkritisch (Info), da bestehende Consumer zwar noch auf dieses „Header Object“ reagieren können, der Producer jedoch nicht mehr sendet. Das Hinzufügen eines „Header Object“ ist wiederum kritisch (Critical), da bestehende Consumer dieses wahrscheinlich noch nicht verarbeiten können.

- **content:** Bei diesem Knoten handelt es sich um einen MapNode von „Media Type Object“ (siehe: Unterunterabschnitt 3.1.6.22) (vgl.: Miller und Ron (2017)). Das Löschen von einem „Media Type Object“ aus diesem Node ist unkritisch (Info), da bestehende Consumer zwar noch auf dieses „Media Type Object“ reagieren können, der Producer es jedoch nicht mehr sendet. Das Hinzufügen eines „Media Type Object“ ist wiederum kritisch (Critical), da bestehende Consumer dieses wahrscheinlich noch nicht verarbeiten können.

- **links:** Bei diesem Knoten handelt es sich um einen MapNode von „Link Object“ (siehe: Unterunterabschnitt 3.1.6.19) (vgl.: Miller und Ron (2017)). Das Löschen von einem „Link

Object“ aus diesem Node ist kritisch (Critical), da bestehende Consumer möglicherweise einen Link suchen, der gar nicht mehr existiert. Das Hinzufügen von einem „Link Object“ ist wiederum unkritisch (Info), da bestehende Consumer dieses wahrscheinlich noch nicht verarbeitet haben.

3.1.6.13 Example Object

- **value:** Dieses Blatt enthält ein beliebiges Objekt, das als Beispiel dient (vgl.: Miller und Ron (2017)). Alle Operationen auf diesem Blatt sind unkritisch (Info), da es keine Auswirkungen auf bestehende APIs hat.
- **externalValue:** Dieses Blatt enthält eine Url, die auf ein Beispiel zeigt (vgl.: Miller und Ron (2017)). Alle Operationen auf diesem Blatt sind unkritisch (Info), da es keine Auswirkungen auf bestehende APIs hat.

3.1.6.14 Request Body Object

- **content:** Bei diesem Knoten handelt es sich um einen MapNode von „Media Type Object“ (siehe: Unterunterabschnitt 3.1.6.22) (vgl.: Miller und Ron (2017)). Das Löschen von einem „Media Type Object“ aus diesem Knoten ist kritisch (Critical), da bestehende Consumer noch auf diesen zugreifen könnten. Das Hinzufügen von einem „Media Type Object“ ist wiederum unkritisch (Info), da bestehende Consumer zwar nicht auf diesen zugreifen, die API aber weiterhin korrekt bedienen.
- **required:** Dieses Blatt enthält einen Boolean, der bestimmt, ob ein Body gesendet werden muss. Wenn dieses Blatt nicht existiert, ist sein Wert „False“ (vgl.: Miller und Ron (2017)). Änderungen von „False“ auf „True“ sind kritisch (Critical), da bestehende Consumer die API möglicherweise nicht mehr korrekt bedienen können. Änderungen von „True“ auf „False“ sind unkritisch (Info), da bestehende Consumer die API weiterhin korrekt bedienen können.

3.1.6.15 Header Object

Das „Header Object“ dient zur Definition von Header-Parametern (vgl.: Miller und Ron (2017)). In dieser Arbeit wird angenommen, das es sich hierbei um ein Blatt handelt. Die Kritikalitätslevel für das Erstellen und Löschen dieses Blattes werden in den Elternobjekten festgelegt. Ändert sich ein bestehendes „Header Object“, ist das Kritikalitätslevel „Critical“, da möglicherweise eine erhebliche Änderung an der API durchgeführt wurde.

3.1.6.16 Security Scheme Object

Das „Security Scheme Object“ dient zur Festlegung von Sicherheitsmechanismen für ein „Operation Object“ (vgl.: Miller und Ron (2017)). Alle Operationen auf seinen Blätter und dem Knoten „flows“ vom Typ „OAuth Flows Object“ (siehe: Unterunterabschnitt 3.1.6.17 werden zunächst als kritisch (Critical) betrachtet, da die Consumer bei Änderungen an diesem Knoten möglicherweise nicht mehr auf die Teile der oder die ganze API zugreifen können.

3.1.6.17 OAuth Flows Object

Das „OAuth Flows Object“ dient zur Festlegung der Konfiguration der OAuth Flows (vgl.: Miller und Ron (2017)). Alle Operationen auf seine Knoten „implicit“, „password“, „clientCredentials“, „authorizationCode“ vom Typ „OAuth Flow Object“ (siehe: Unterunterabschnitt 3.1.6.18 werden zunächst als kritisch (Critical) betrachtet, da die Consumer bei Änderungen an diesem Knoten möglicherweise nicht mehr auf die Teile der oder die ganze API zugreifen können.

3.1.6.18 OAuth Flow Object

Das „OAuth Flow Object“ dient zur Festlegung der Konfigurationsdetails von einem OAuth Flow (vgl.: Miller und Ron (2017)). Alle Operationen auf seinen Blättern werden zunächst als kritisch (Critical) betrachtet, da die Consumer bei Änderungen an diesem Knoten möglicherweise nicht mehr auf die Teile der oder die ganze API zugreifen können.

3.1.6.19 Link Object

Das „Link Object“ dient zur Definition eines Links zu weiteren Ressourcen (vgl.: Miller und Ron (2017)). In dieser Arbeit wird angenommen, dass es sich hierbei um ein Blatt handelt. Die Kritikalitätslevel für das Erstellen und Löschen dieses Blattes werden in den Elternobjekten festgelegt. Ändert sich ein bestehendes „Link Object“, ist das Kritikalitätslevel „Critical“, da möglicherweise eine erhebliche Änderung an der API durchgeführt wurde.

3.1.6.20 Callback Object

Bei dem „Callback Object“ handelt es sich um einen MapNode von „Path Item Object“ (vgl.: Miller und Ron (2017)).

3.1.6.21 External Documentation Object

- **url:** Dieses Blatt enthält eine Url, die auf eine externe Dokumentation zeigt (vgl.: Miller und Ron (2017)). Änderungen an diesem Blatt sind unkritisch, da sie nicht die API betreffen.

3.1.6.22 Media Type Object

Das „Media Type Object“ dient zur Definition von Payloads (vgl.: Miller und Ron (2017)). In dieser Arbeit wird angenommen, das es sich hierbei um ein Blatt handelt. Die Kritikalitätslevel für das Erstellen und Löschen dieses Blattes werden in den Elternobjekten festgelegt. Ändert sich ein bestehendes „Media Type Object“, ist das Kritikalitätslevel „Critical“, da möglicherweise eine erhebliche Änderung an der API durchgeführt wurde.

3.2 Analyse bestehender Difftools auf ihre Verwendbarkeit

In diesem Abschnitt werden bestehende Lösungen für den Vergleich zweier Swagger-Definitionen betrachtet. In Unterabschnitt 3.2.1 werden zunächst die Kriterien aufgestellt, mittels welcher die bestehenden Tools auf ihre Verwendbarkeit hin getestet werden. In Unterabschnitt 3.2.2 wird geprüft, ob ein Vergleich mittels bestehender Textvergleichstools sinnvoll ist und in Unterabschnitt 3.2.3 werden vorhandene spezialisierte Lösungen betrachtet. Abschließend wird in Unterabschnitt 3.2.4 ein kurzes Fazit bezüglich der Verwendbarkeit bestehender Difftools gezogen.

3.2.1 Analysekriterien für die Verwendbarkeit bestehender Difftools

In Abschnitt 3.1 wurde ermittelt, welche Bedeutung die einzelnen Knoten einer Swagger-Definition, die gemäß der OpenAPI-Spezifikation Version 3.0.0 erstellt wurde, haben. Außerdem ist es Ziel dieser Arbeit zu zeigen, dass es maschinell möglich ist, die Änderungen aus zwei Swagger-Definitionen, die gemäß der OpenAPI-Spezifikation Version 3.0.0 (vgl.: Miller und Ron (2017)) erstellt wurden, zu extrahieren, auf ihre Abwärtskompatibilität hin zu prüfen und dies zu visualisieren. Darauf basierend, werden in diesem Abschnitt Kriterien festgelegt, anhand derer die möglichen Vergleichstools analysiert werden.

1. **Unterstützung von Definitionen gemäß OpenAPI-Spezifikation Version 3.0.0:** Aus dem Ziel der Arbeit geht hervor, dass Swagger-Definitionen, die gemäß der OpenAPI-Spezifikation-Version 3.0.0 (vgl.: Miller und Ron (2017)) erstellt wurden, unterstützt werden sollen.

2. **Erkennung von nicht geänderten Spezifikationen bei einer Reihenfolgeveränderungen:** Sowohl Spezifikationen, die im YAML-Format (siehe: Unterabschnitt 2.2.2), als auch im JSON-Format (siehe: Unterabschnitt 2.2.1) vorliegen, sind strukturierte Dokumente, bei denen es in vielen Teilen nicht auf die Reihenfolge ankommt, in der die einzelnen Knoten aufgeführt sind. Die Vergleichstools sollten also zwei Definitionen, bei denen lediglich die Reihenfolge von Knoten oder Blättern vertauscht wurde, die Schnittstelle sich dadurch jedoch nicht ändern würde, als unverändert erkennen. Dies wird anhand der in Listing 3.10 und Listing 3.11 gezeigten Swagger-Definitionen ermittelt. Diese enthalten ein „Paths Object“ mit drei „Path Item Objects“, die identisch sind, bei denen sich lediglich die Reihenfolge, in der sie aufgeführt sind, umgekehrt hat.

```
openapi: 3.0.0
info:
  title: 'TestAPI'
  version: 1.0.0
paths:
  /path1:
    get:
      summary: Pfad1 Summary
      operationId: pfad1Id
      parameters:
        - in: query
          name: input1
          required: false
          schema:
            type: string
      responses:
        '200':
          description: Pfad1 Response
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - id
                  properties:
                    id:
```

```
29                      type: string
30                      format: uuid
31                      example: d290f1ee-6c54-4b01-90e6-
                           d701748f0851
32  /path2:
33    get:
34      summary: Pfad1 Summary
35      operationId: pfad2id
36      parameters:
37        - in: query
38          name: input2
39          required: true
40          schema:
41            type: string
42      responses:
43        '200':
44          description: Pfad2 Response
45          content:
46            application/json:
47              schema:
48                type: array
49                items:
50                  type: object
51                  required:
52                    - name
53                  properties:
54                    name:
55                      type: string
56                      example: Widget Adapter
57  /path3:
58    get:
59      summary: Pfad1 Summary
60      operationId: pfad2id
61      parameters:
62        - in: query
63          name: searchString
64          required: true
65          schema:
66            type: string
67      responses:
```

```
68          '200':
69            description: Pfad2 Response
70            content:
71              application/json:
72                schema:
73                  type: array
74                  items:
75                    type: object
76                    required:
77                      - releaseDate
78                    properties:
79                      releaseDate:
80                        type: string
81                        format: int32
82                        example: '2016-08-29T09:12:33.001Z'
```

Listing 3.10: Swagger-Definition zur Erkennung von nicht geänderten Spezifikationen bei einer Reihenfolgeveränderungen linke Seite

```
1  openapi: 3.0.0
2  info:
3    title: 'TestAPI'
4    version: "2.0.0"
5  paths:
6    /path3:
7      get:
8        summary: Pfad1 Summary
9        operationId: pfad2id
10       parameters:
11         - in: query
12           name: searchString
13           required: true
14           schema:
15             type: string
16       responses:
17         '200':
18           description: Pfad2 Response
19           content:
20             application/json:
21               schema:
22                 type: array
```

```
23                items:
24                  type: object
25                  required:
26                    - releaseDate
27                  properties:
28                    releaseDate:
29                      type: string
30                      format: int32
31                      example: '2016-08-29T09:12:33.001Z'
32  /path2:
33    get:
34      summary: Pfad1 Summary
35      operationId: pfad2id
36      parameters:
37        - in: query
38          name: input2
39          required: true
40          schema:
41            type: string
42      responses:
43        '200':
44          description: Pfad2 Response
45          content:
46            application/json:
47              schema:
48                type: array
49                items:
50                  type: object
51                  required:
52                    - name
53                  properties:
54                    name:
55                      type: string
56                      example: Widget Adapter
57  /path1:
58    get:
59      summary: Pfad1 Summary
60      operationId: pfad1Id
61      parameters:
62        - in: query
```

```
63          name: input1
64          required: false
65          schema:
66            type: string
67      responses:
68        '200':
69          description: Pfad1 Response
70          content:
71            application/json:
72              schema:
73                type: array
74                items:
75                  type: object
76                  required:
77                    - id
78                  properties:
79                    id:
80                      type: string
81                      format: uuid
82                      example: d290f1ee-6c54-4b01-90e6-
                           d701748f0851
```

Listing 3.11: Swagger-Definition zur Erkennung von nicht geänderten Spezifikationen bei einer Reihenfolgeveränderungen rechte Seite

3. **Erkennung von nicht geänderten Spezifikationen beim Hinzufügen eines Attributes mit Default-Wert:** Ist ein Attribut einer Swagger-Definition gemäß der OpenAPI-Spezifikation mit einem Default-Wert versehen und wird dieses Attribut dann mit dem Default-Wert der Swagger-Definition hinzugefügt (siehe: Miller und Ron (2017)), ist keine Änderung an der Schnittstelle entstanden und diese ist somit abwärtskompatibel. Die Vergleichstools sollten also zwei Definitionen, bei denen ein Attribut mit seinem Default-Wert hinzugefügt wurde, als unverändert erkennen. Dies wird anhand der in Listing 3.12 und Listing 3.13 gezeigten Swagger-Definitionen ermittelt. Diese enthalten einen „Parameter Object“, bei dem das Attribut „required“ mit seinem Default-Wert hinzugefügt wurde.

```
1 openapi: 3.0.0
2 info:
3   title: 'TestAPI'
4   version: 1.0.0
```

```
paths:
  /path1:
    get:
      summary: Pfad1 Summary
      operationId: pfad1Id
      parameters:
        - in: query
          name: input1
          schema:
            type: string
      responses:
        '200':
          description: Pfad1 Response
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - id
                  properties:
                    id:
                      type: string
                      format: uuid
                      example: d290f1ee-6c54-4b01-90e6-
                         d701748f0851
```

Listing 3.12: Swagger-Definition zur Erkennung von nicht geänderten Spezifikationen beim Hinzufügen eines Attributes mit Default-Wert linke Seite

```
openapi: 3.0.0
info:
  title: 'TestAPI'
  version: "2.0.0"
paths:
  /path1:
    get:
      summary: Pfad1 Summary
      operationId: pfad1Id
      parameters:
```

```
11          - in: query
12            name: input1
13            required: false
14            schema:
15              type: string
16        responses:
17          '200':
18            description: Pfad1 Response
19            content:
20              application/json:
21                schema:
22                  type: array
23                  items:
24                    type: object
25                    required:
26                      - id
27                    properties:
28                      id:
29                        type: string
30                        format: uuid
31                        example: d290f1ee-6c54-4b01-90e6-
                             d701748f0851
```

Listing 3.13: Swagger-Definition zur Erkennung von nicht geänderten Spezifikationen beim Hinzufügen eines Attributes mit Default-Wert rechte Seite

4. **Erkennung von geänderten Spezifikationen beim Hinzufügen eines Attributes, wodurch eine kritische API-Änderung eintritt:** Wird der Definition ein Attribut hinzugefügt, welches eine kritische Änderung an der resultierenden API bewirkt, sollten die Vergleichstools also zwei Definitionen, bei denen solch ein Attribut hinzugefügt wurde, als verändert erkennen. Dies wird anhand der in Listing 3.14 und Listing 3.15 gezeigten Swagger-Definitionen ermittelt. Diese enthalten ein „Parameter Object“, bei dem das Attribut „required“ mit dem Wert „true“ hinzugefügt wurde.

```
1 openapi: 3.0.0
2 info:
3   title: 'TestAPI'
4   version: 1.0.0
5 paths:
6   /path1:
```

```
    get:
      summary: Pfad1 Summary
      operationId: pfad1Id
      parameters:
        - in: query
          name: input1
          schema:
            type: string
      responses:
        '200':
          description: Pfad1 Response
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - id
                  properties:
                    id:
                      type: string
                      format: uuid
                      example: d290f1ee-6c54-4b01-90e6-
                         d701748f0851
```

Listing 3.14: Swagger-Definition zur Erkennung von geänderten Spezifikationen beim Hinzufügen eines kritischen Attributes linke Seite

```
openapi: 3.0.0
info:
  title: 'TestAPI'
  version: "2.0.0"
paths:
  /path1:
    get:
      summary: Pfad1 Summary
      operationId: pfad1Id
      parameters:
        - in: query
          name: input1
```

```
13          required: false
14          schema:
15            type: string
16      responses:
17        '200':
18          description: Pfad1 Response
19          content:
20            application/json:
21              schema:
22                type: array
23                items:
24                  type: object
25                  required:
26                    - id
27                  properties:
28                    id:
29                      type: string
30                      format: uuid
31                      example: d290f1ee-6c54-4b01-90e6-
                           d701748f0851
```

Listing 3.15: Swagger-Definition zur Erkennung von geänderten Spezifikationen beim Hinzufügen eines kritischen Attribute rechte Seite

3.2.2 Analyse von Text-Diff-Tools zum Vergleich zweier Swagger-Definitionen

Mit Text-Diff-Tools lassen sich zwei Dateien, die im Textformat vorliegen auf Änderungen hin untersuchen. Für die Analyse wird exemplarisch das Text-Diff-Tool „Diffuse“ (siehe Moser und Menke (2018)) verwendet.

1. **Unterstützung von Definitionen gemäß OpenAPI-Spezifikation Version 3.0.0:** Da Swagger-Definitionen mittels JSON oder YAML definiert werden, liegen diese im Textformat vor. Somit können auch zwei Swagger-Definitionen, die gemäß der OpenAPI-Spezifikation Version 3.0.0 erstellt wurden, miteinander verglichen werden.

2. **Erkennung von nicht geänderten Spezifikationen bei einer Reihenfolgeveränderungen:** Bei diesem Test wird lediglich das „Path Item Object“ „/path1“ als unverändert erkannt. Abbildung 3.5 zeigt einen Screenshot von dem Vergleichsergebnis.

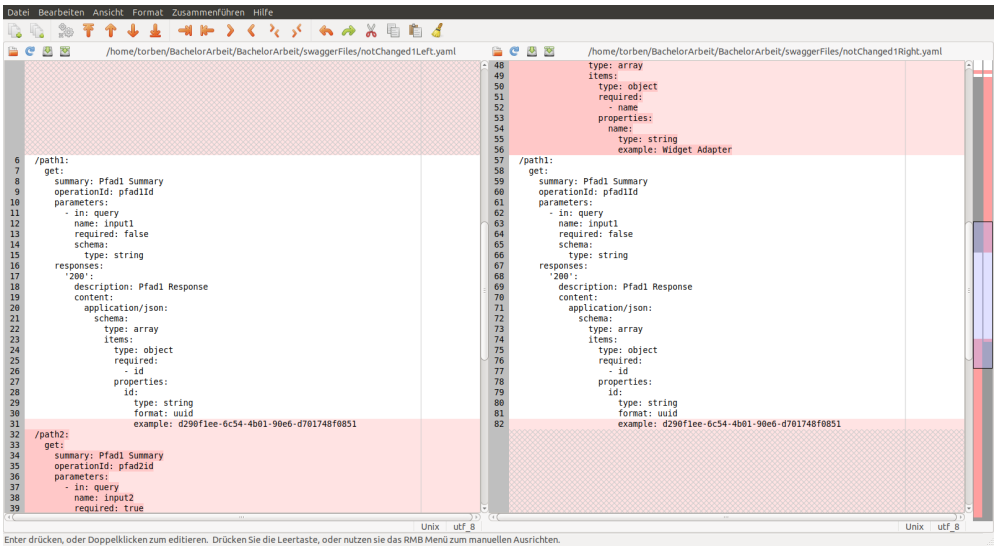

Abbildung 3.5: Screenshot von Diffuse beim Vergleich der Dateien Listing 3.10 und Listing 3.11

3. **Erkennung von nicht geänderten Spezifikationen beim Hinzufügen eines Attributes mit Default-Wert:** Nicht getestet, da bereits eine Erkennung von nicht geänderten Spezifikationen bei einer Reihenfolgeveränderungen nicht möglich war.

3.2.3 Analyse von spezialisierten Tools zum Vergleich zweier Swagger-Definitionen

Es konnten zu beginn dieser Arbeit folgende Tools gefunden werden, die speziell auf den Vergleich zweier Swagger-Definitionen ausgelegt sind:

1. **https://github.com/civisanalytics/swagger-diff:** Bei diesem Tool handelt es sich um ein Java-Programm, welches zwei Swagger-Definitionen miteinander vergleicht, die gemäß der OpenAPI-Spezifikation Version 2 entwickelt wurden (vgl.: Cousens (2018)). Daher wird dieses Tool nicht weiter betrachtet.

2. **https://github.com/Sayi/swagger-diff:** Bei diesem Tool handelt es sich um ein Ruby-Gem, welches zwei Swagger-Definitionen miteinander vergleicht, die gemäß der OpenAPI-Spezifikation Version 2 entwickelt wurden (vgl.: Sayi (2018)). Daher wird dieses Tool nicht weiter betrachtet.

3. **https://github.com/zallek/swagger-diff:** Bei diesem Tool handelt es sich um ein JavaScript-Programm, welches zwei Swagger-Definitionen miteinander vergleicht, die gemäß der

OpenAPI-Spezifikation Version 2 entwickelt wurden (vgl.: Fortin und Sevilla (2018)). Daher wird dieses Tool nicht weiter betrachtet.

4. **https://github.com/Azure/openapi-diff:** Bei diesem Tool handelt es sich um ein JavaScript-Programm, welches zwei Swagger-Definitionen miteinander vergleicht (vgl.: Barosan und Shah (2018)). Aus der README wird zunächst nicht klar, welche OpenAPI-Spezifikation das Programm unterstützt (vgl.: Barosan und Shah (2018)). Versucht man das Programm mit Swagger-Definitionen, die gemäß OpenAPI-Version 3.0.0 entwickelt wurden, aufzurufen, erhält man die in Listing 3.16 gezeigte Ausgabe, woraus ersichtlich wird, dass lediglich die OpenAPI-Version 2 unterstützt wird. Daher wird dieses Tool nicht weiter betrachtet.

```
1 FATAL: swagger-document/loader - FAILED
2 FATAL: Error: File 'file:///home/torben/BachelorArbeit/
     BachelorArbeit/swaggerFiles/notChanged1Left.yaml' is not a
     valid OpenAPI 2.0 definition (expected 'swagger: 2.0')
3   Error: File 'file:///home/torben/BachelorArbeit/BachelorArbeit
       /swaggerFiles/notChanged1Left.yaml' is not a valid OpenAPI
       2.0 definition (expected 'swagger: 2.0')
```

Listing 3.16: Test von https://github.com/Azure/openapi-diff mit Swagger-Definitionen gemäß OpenAPI-Spezifikation Version 3

5. **https://github.com/AllanHoejgaardJensen/open-api-diff:** Bei diesem Tool handelt es sich um ein Java-Programm, welches zwei Swagger-Definitionen miteinander vergleicht, die gemäß der OpenAPI-Spezifikation Version 2 entwickelt wurden (vgl.: Jensen und Krochmalski (2018)). Daher wird dieses Tool nicht weiter betrachtet.

6. **https://bitbucket.org/atlassian/openapi-diff:** Bei diesem Tool handelt es sich um ein TypeScript-Programm, welches zwei Swagger-Definitionen miteinander vergleicht, die gemäß der OpenAPI-Spezifikation Version 2 entwickelt und OpenAPI-Spezification Version 3 wurden (vgl.: Edo (2018a) und Edo (2018b)). Es werden jedoch nur geringe Teile der OpenAPI-Version 3 unterstützt. Führt man den Test „Erkennung von geänderten Spezifikationen beim Hinzufügen eines Attributes, wodurch eine kritische API-Änderung eintritt“ (siehe: Unterabschnitt 3.2.1) durch, wird, wie aus Listing 3.17 ersichtlich, ein False-Positiv ermittelt. Daher wird dieses Tool nicht weiter betrachtet.

```
1 * OpenAPI Diff v0.6.1 *
2
3
```

```
Inputs
------
Old spec: requiredValueLeft.yaml
New spec: requiredValueRight.yaml


Summary
-------
0 breaking changes found.
1 non-breaking changes found.
0 unclassified changes found.


Details
-------
Non-breaking: the path [info/version] was modified from '1.0.0'
   to '2.0.0'
```

Listing 3.17: Consolenausgabe von https://bitbucket.org/atlassian/openapi-diff beim Vergleich der Datei Listing 3.14 und Listing 3.15

7. **SwaggerHub Compare & Merge:** Dieses Tool wird in Unterunterabschnitt 3.2.3.1 näher betrachtet.

3.2.3.1 SwaggerHub Compare & Merge

Bei „SwaggerHub Compare & Merge“ handelt es sich um eine Funktion der Plattform „SwaggerHub“ (`https://app.swaggerhub.com`), die es ermöglicht, zwei Swagger-Definitionen miteinander zu vergleichen und zu mergen (vgl.: SmartBear Software (2018c)).

1. **Unterstützung von Definitionen gemäß OpenAPI-Spezifikation Version 3.0.0:** SwaggerHub unterstützt Definitionen gemäß OpenAPI-Spezifikation Version 3.0.0 (vgl.: SmartBear Software (2018b)).

2. **Erkennung von nicht geänderten Spezifikationen bei einer Reihenfolgeveränderungen:** Wie aus Abbildung 3.6 ersichtlich wird, werden Reihenfolgeveränderungen korrekt verarbeitet.

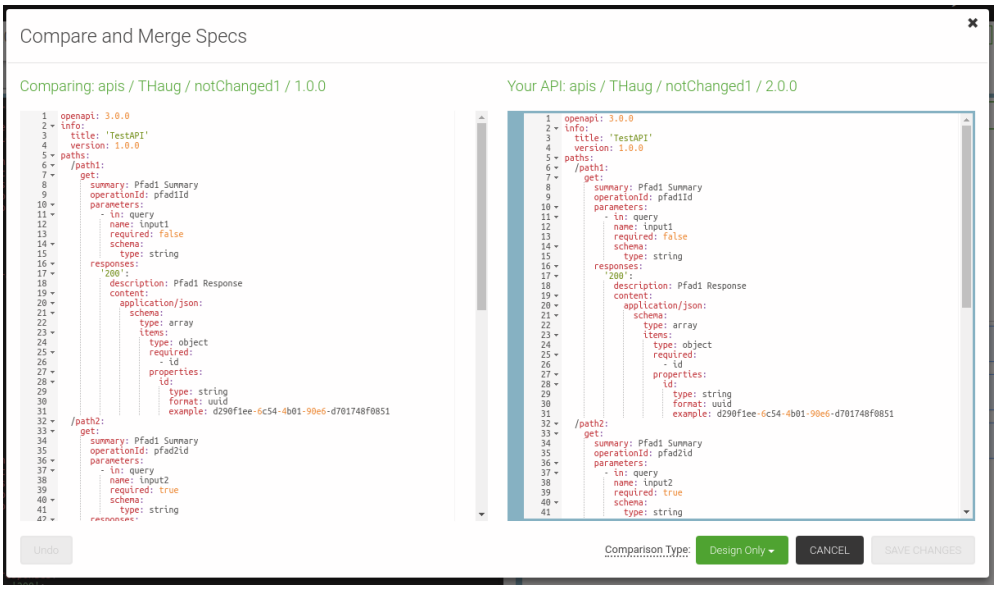

Abbildung 3.6: Screenshot von „SwaggerHub Compare & Merge“ beim Vergleich der Datei Listing 3.10 und Listing 3.11

3. **Erkennung von nicht geänderten Spezifikationen beim Hinzufügen eines Attributes mit Default-Wert:** Wie aus Abbildung 3.7 ersichtlich wird, erkennt „SwaggerHub Compare & Merge“ das Hinzufügen als Änderung.

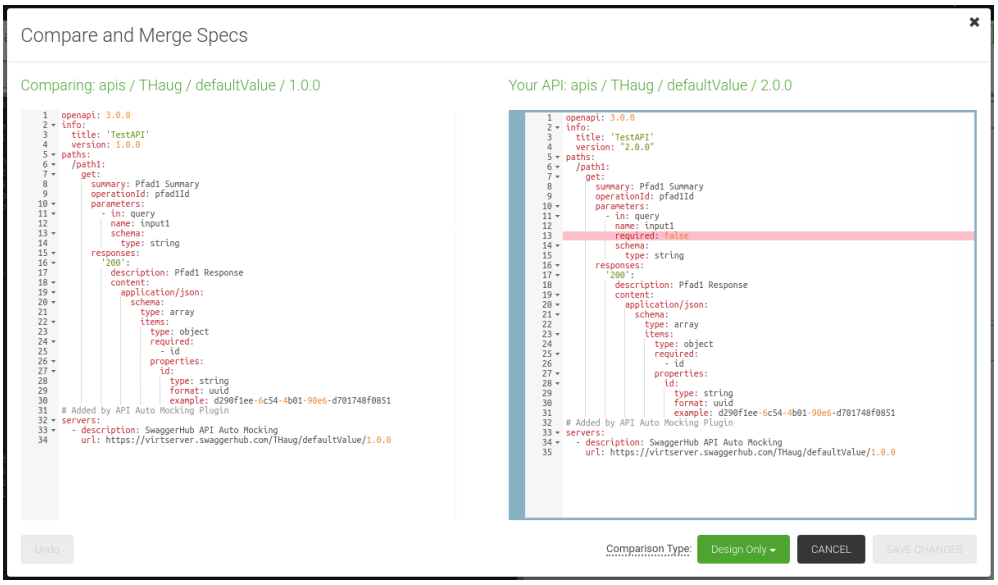

Abbildung 3.7: Screenshot von „SwaggerHub Compare & Merge“ beim Vergleich der Datei Listing 3.12 und Listing 3.13

4. **Erkennung von geänderten Spezifikationen beim Hinzufügen eines Attributes, wodurch eine kritische API-Änderung eintritt:** Wie aus Abbildung 3.8 und Abbildung 3.7 ersichtlich wird, erkennt „SwaggerHub Compare & Merge“ das Hinzufügen eines kritischen genauso, wie das Hinzufügen eines unkritischen Attributes. Es ist aber keine Unterscheidung möglich, ob es sich um eine kritische, oder unkritische Änderung handelt.

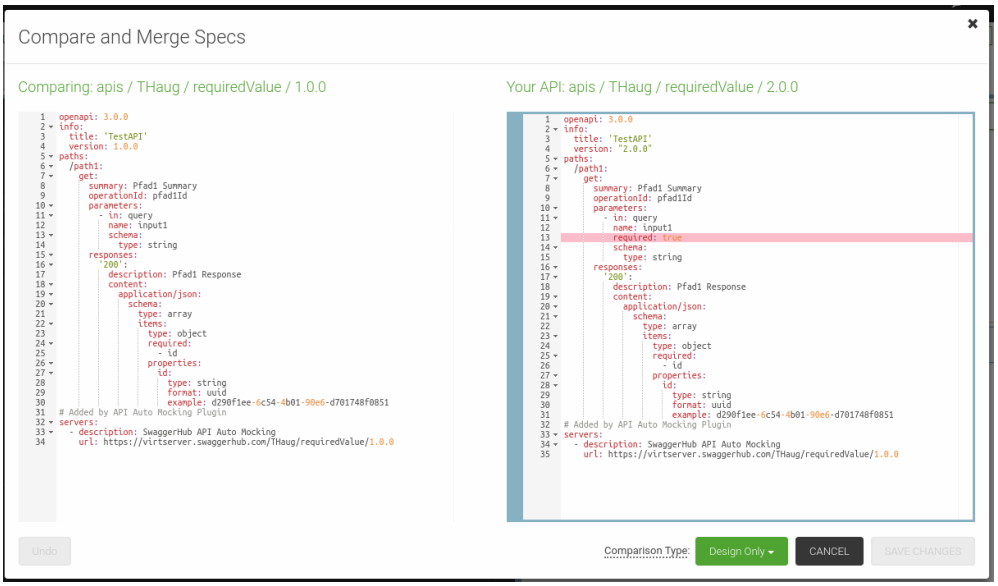

Abbildung 3.8: Screenshot von „SwaggerHub Compare & Merge“ beim Vergleich der Datei Listing 3.14 und Listing 3.15

Dieses Tool scheint somit nicht zur Erkennung von Veränderungen an der resultierenden API, sondern lediglich zur allgemeinen Erkennung von Änderungen zwischen zwei Swagger-Definitionen ausgelegt zu sein.

3.2.4 Fazit zur Analyse bestehender Difftools auf ihre Verwendbarkeit

Keines der bestehenden Tools erfüllt die in Unterabschnitt 3.2.1 aufgestellten Kriterien.

3.3 Analyse der vorhandenen Swaggerbibliotheken

Das Swaggerteam stellt einige Bibliotheken und Tools quelloffen zur Verfügung(vgl.: SmartBear Software (2018e)). Hierbei handelt es sich um diverse Tools, um mit Swagger-Definitionen zu arbeiten. Für diese Arbeit ist hiervon die einzig relevante Bibliothek der „Swagger-Parser“. Dieser wird in Unterabschnitt 3.3.1 näher betrachtet. Zudem gibt es viele weitere Bibliotheken, die durch freie Entwickler bereitgestellt werden. James Messinger stellt einen „Swagger-Parser“ für JavaScript als Bibliothek bereit (vgl.: Messinger (2018a)). Dieser unterstützt allerdings erst seit dem 25.05.2018 OpenAPI-Spezifikationen Version 3.0.0 (vgl.: Messinger (2018b)) und konnte daher nicht in die Architekturentscheidungen für diese Arbeit mit einfließen.

3.3.1 Swagger-Parser

Der Swagger-Parser ist eine Bibliothek mithilfe derer Swagger-Definitionen in Java POJOs umgewandelt werden können (vgl.: Tam u. a. (2018)). Der Swagger-Parser steht als Maven Modul zur Verfügung (vgl: Tam u. a. (2018)) und kann somit in Mavenprojekten genutzt werden. Listing 3.18 zeigt ein einfachen Beispiel, eines Mavenprojekts mit einer Abhängigkeit zu „Swagger-Parser“.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http:
             //maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>de.haug-dev.swagger-compare</groupId>
  <artifactId>swagger-parser-example</artifactId>
  <version>1.0.0</version>

  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>io.swagger.parser.v3</groupId>
      <artifactId>swagger-parser</artifactId>
      <version>2.0.0-rc2</version>
    </dependency>
  </dependencies>

</project>
```

Listing 3.18: Maven Swagger-Parser Beispiel

Die Bibliothek enthält die Klasse „OpenAPIV3Parser“, die das Einlesen von Swagger-Definitionen ermöglicht und diese dann als POJO zurück gibt. In Listing 3.19 wird gezeigt, wie eine lokale Datei eingelesen werden kann. Listing 3.20 zeigt die für dieses Beispiel verwendete Swagger-Definition und Listing 3.21 zeigt das resultierende POJO.

```
import io.swagger.v3.oas.models.OpenAPI;
```

```java
import io.swagger.v3.parser.OpenAPIV3Parser;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;


public class SwaggerParserExample {

    private static final OpenAPIV3Parser OPEN_API_V3_PARSER = new
        OpenAPIV3Parser();

    public static void main(String [ ] args) throws IOException {
        OpenAPI openAPI = OPEN_API_V3_PARSER.read("
            swaggerParserExample.yaml");
        try (BufferedWriter br = new BufferedWriter(new FileWriter("
            swaggerParserExampleOutput.txt"))){
            br.write(openAPI.toString().replace("\n\n", "\n"));
        }
    }
}
```

Listing 3.19: Java Swagger-Parser Beispiel

```yaml
openapi: 3.0.0
info:
  title: 'swaggerParserExample'
  version: 1.0.0
paths:
  /path1:
    get:
      summary: Pfad1 Summary
      operationId: pfad1Id
      parameters:
        - in: query
          name: input1
          schema:
            type: string
      responses:
        '200':
          $ref: '#/components/responses/testresponse'
```

```
components:
  responses:
    testresponse:
      description: Pfad1 Response
      content:
        application/json:
          schema:
            type: array
            items:
              type: object
              required:
                - id
              properties:
                id:
                  type: string
                  format: uuid
                  example: d290f1ee-6c54-4b01-90e6-d701748f0851
```

Listing 3.20: Swagger-Parser Beispiel Swagger-Definition

```
class OpenAPI {
  openapi: 3.0.0
  info: class Info {
    title: swaggerParserExample
    description: null
    termsOfService: null
    contact: null
    license: null
    version: 1.0.0
  }
  externalDocs: null
  servers: [class Server {
    url: /
    description: null
    variables: null
  }]
  security: null
  tags: null
  paths: class Paths {
    {/path1=class PathItem {
      summary: null
```

```
22      description: null
23      get: class Operation {
24        tags: null
25        summary: Pfad1 Summary
26        description: null
27        externalDocs: null
28        operationId: pfad1Id
29        parameters: [class QueryParameter {
30          class Parameter {
31            name: input1
32            in: null
33            description: null
34            required: false
35            deprecated: null
36            allowEmptyValue: null
37            style: form
38            explode: true
39            allowReserved: null
40            schema: class StringSchema {
41              class Schema {
42                title: null
43                multipleOf: null
44                maximum: null
45                exclusiveMaximum: null
46                minimum: null
47                exclusiveMinimum: null
48                maxLength: null
49                minLength: null
50                pattern: null
51                maxItems: null
52                minItems: null
53                uniqueItems: null
54                maxProperties: null
55                minProperties: null
56                required: null
57                type: null
58                not: null
59                properties: null
60                additionalProperties: null
61                description: null
```

```
62                format: null
63                $ref: null
64                nullable: null
65                readOnly: null
66                writeOnly: null
67                example: null
68                externalDocs: null
69                deprecated: null
70                discriminator: null
71                xml: null
72              }
73              type: string
74              _default: null
75              _enum: null
76            }
77            examples: null
78            example: null
79            content: null
80            $ref: null
81          }
82          in: query
83        }]
84        requestBody: null
85        responses: class ApiResponses {
86          {200=class ApiResponse {
87            description: null
88            headers: null
89            content: null
90            links: null
91            $ref: #/components/responses/testresponse
92          }}
93        }
94        callbacks: null
95        deprecated: null
96        security: null
97        servers: null
98      }
99      put: null
100     post: null
101     delete: null
```

```
    options: null
    head: null
    patch: null
    trace: null
    servers: null
    parameters: null
    $ref: null
  }}
}
components: class Components {
  schemas: null
  responses: class ApiResponses {
    {testresponse=class ApiResponse {
      description: Pfad1 Response
      headers: null
      content: class Content {
        {application/json=class MediaType {
          schema: class ArraySchema {
            class Schema {
              title: null
              multipleOf: null
              maximum: null
              exclusiveMaximum: null
              minimum: null
              exclusiveMinimum: null
              maxLength: null
              minLength: null
              pattern: null
              maxItems: null
              minItems: null
              uniqueItems: null
              maxProperties: null
              minProperties: null
              required: null
              type: null
              not: null
              properties: null
              additionalProperties: null
              description: null
              format: null
```

```
                $ref: null
                nullable: null
                readOnly: null
                writeOnly: null
                example: null
                externalDocs: null
                deprecated: null
                discriminator: null
                xml: null
              }
              type: array
              items: class ObjectSchema {
                class Schema {
                  title: null
                  multipleOf: null
                  maximum: null
                  exclusiveMaximum: null
                  minimum: null
                  exclusiveMinimum: null
                  maxLength: null
                  minLength: null
                  pattern: null
                  maxItems: null
                  minItems: null
                  uniqueItems: null
                  maxProperties: null
                  minProperties: null
                  required: [id]
                  type: null
                  not: null
                  properties: {id=class UUIDSchema {
                    class Schema {
                      title: null
                      multipleOf: null
                      maximum: null
                      exclusiveMaximum: null
                      minimum: null
                      exclusiveMinimum: null
                      maxLength: null
                      minLength: null
```

```
                        pattern: null
                        maxItems: null
                        minItems: null
                        uniqueItems: null
                        maxProperties: null
                        minProperties: null
                        required: null
                        type: null
                        not: null
                        properties: null
                        additionalProperties: null
                        description: null
                        format: null
                        $ref: null
                        nullable: null
                        readOnly: null
                        writeOnly: null
                        example: d290f1ee-6c54-4b01-90e6-d701748f0851
                        externalDocs: null
                        deprecated: null
                        discriminator: null
                        xml: null
                      }
                      type: string
                      format: uuid
                      _default: null
                      _enum: null
                    }}
                    additionalProperties: null
                    description: null
                    format: null
                    $ref: null
                    nullable: null
                    readOnly: null
                    writeOnly: null
                    example: null
                    externalDocs: null
                    deprecated: null
                    discriminator: null
                    xml: null
```

```
                }
                type: object
                defaultObject: null
              }
            }
            examples: null
            example: null
            encoding: null
          }}
        }
        links: null
        $ref: null
      }}
    }
    parameters: null
    examples: null
    requestBodies: null
    headers: null
    securitySchemes: null
    links: null
    callbacks: null
  }
}
```

Listing 3.21: Swagger-Parser Beispiel Ergebnis

Wie aus Listing 3.20 und Listing 3.21 ersichtlich ist, wird die Swagger-Definition eins zu eins in ein POJO umgewandelt. In der Swagger-Definition fehlende Blätter und Knoten werden durch „null“ ersetzt. Zudem erfolgt keine Auflösung der Blätter „$ref“.

4 Architektur und Umsetzung

In diesem Kapitel wird auf die Architekturentwicklung und Umsetzung des Prototypen eingegangen. Zunächst wird in Abschnitt 4.1 die Grundarchitektur ausgewählt und in Abschnitt 4.2 wird dann die Client-Server Architektur im Kontext dieser Arbeit erläutert. Danach werden in Abschnitt 4.3 die eingesetzten Programmiersprachen und Frameworks gezeigt und in Abschnitt 4.4 auf den Build und die Infrastruktur eingegangen. Abschließend wird in Abschnitt 4.5 die Umsetzung des technischen und in Abschnitt 4.6 des fachlichen Prototypen gezeigt.

4.1 Auswahl des Architekturstils

In diesem Abschnitt wird ermittelt, welcher Architekturstil für das Projekt am sinnvollsten ist. Näher betrachtet werden hierbei in Unterabschnitt 4.1.1 die Standalone Application, in Unterabschnitt 4.1.2 das Client-Server Modell und in Unterabschnitt 4.1.3 das Microservice Modell im Kontext dieser Arbeit. Abschließend wird in Unterabschnitt 4.1.4 ein Fazit gezogen, welcher der drei Architekturstile für diese Arbeit der sinnvollste ist.

4.1.1 Standalone Application

Abbildung 4.1 zeigt beispielhaft eine Standalone Application. Einerseits ist diese architektonisch sehr einfach, da Schnittstellen nur intern existieren und keine Kommunikation außerhalb des Anwendungskontextes notwendig ist. Außerdem werden sowohl GUI als auch Backend in der gleichen Programmiersprache entwickelt. Andererseits ist man aber auch gezwungen GUI und Backend in der gleichen Programmiersprache zu entwickeln. Zudem sind sowohl Backend, als auch Frontend an die gleich Laufzeitumgebung gebunden.

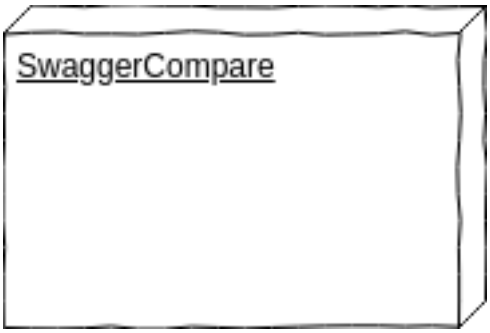

Abbildung 4.1: Swagger Compare Standalone

4.1.2 Client Server Modell

Wie Abbildung 4.2 zeigt, sind beim Client-Server-Modell Frontend und Backend getrennt. Dies erhöht einerseits die Komplexität der Architektur und des Programms im Gegensatz zur Standalone Application (siehe: Unterabschnitt 4.1.1), da Programmübergreifende Schnittstellen notwendig sind. Andererseits können Frontend und Backend in unterschiedlichen Programmiersprachen verfasst werden und sind somit nicht an die selbe Laufzeitumgebung gebunden.

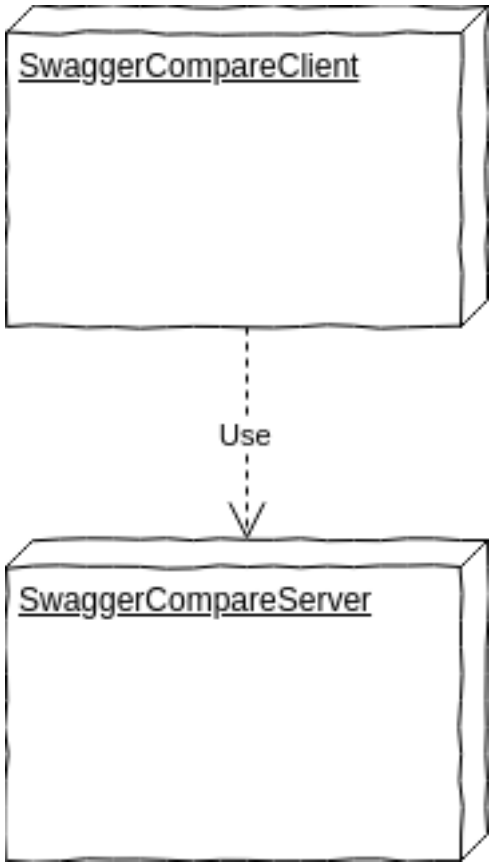

Abbildung 4.2: Swagger Compare Client-Server-Modell

4.1.3 Microservice Modell

In Abbildung 4.3 wird beispielhaft gezeigt, wie diese Arbeit in einer Microservicearchitektur umgesetzt werden könnte. Hierbei wird nicht näher auf gängige Pattern, wie zum Beispiel den Einsatz eines API-Gateways, eingegangen. Es wird ersichtlich, dass beim Microservice Modell, ähnlich wie beim Client-Server-Modell (siehe Unterabschnitt 4.1.2, Frontend und Backend getrennt sind. Somit können auch hier Frontend und Backend in unterschiedlichen Programmiersprachen entwickelt werden und können in verschiedenen Laufzeitumgebungen ausgeführt werden. Zudem bietet das Microservice Modell noch diverse weitere Vorteile, wie Belastbarkeit, Skalierbarkeit, und viele mehr (vgl.: Newman und Lorenzen (2015) S. 26). Allerdings steigt die Komplexität enorm an, da das Frontend nun nicht nur mit einem Service, sondern mit mehreren kommunizieren muss. Außerdem müssen auch die Services im Backend gegebenenfalls untereinander kommunizieren.

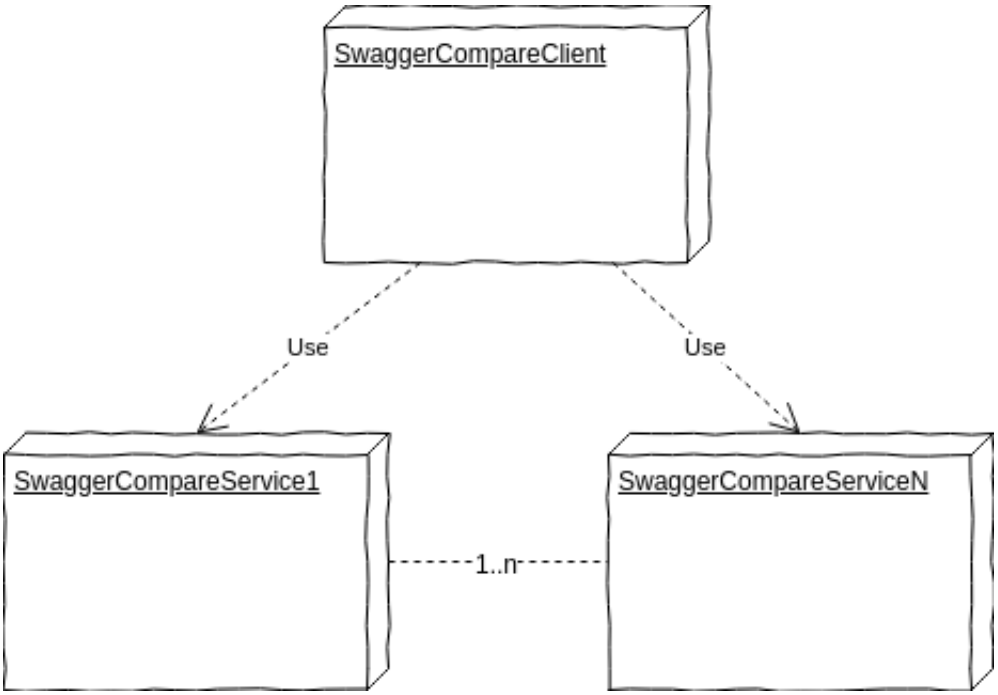

Abbildung 4.3: Swagger Compare Microservice-Modell

4.1.4 Fazit

Meines Erachtens ist der beste Architekturstil für das vorliegenden Projekt das Client-Server-Modell (siehe: Unterabschnitt 4.1.2). Zum einen ist das Backend an die Laufzeitumgebung „Java“ gebunden, da wichtige Komponenten zur Zeit nur als Java-Bibliotheken bereitstehen (siehe: Abschnitt 3.3) und somit bei der Standalone Application das gesamte Projekt an die Java-Laufzeitumgebung gebunden ist. Da die heutzutage wohl am meisten verbreitete Laufzeitumgebung nicht Java, sondern der Webbrowser ist, sollten meines Erachtens die auf dem Nutzer PC ausgeführten Programmteile im Webbrowser nutzbar sein. Um allerdings von den zusätzlichen Vorteilen einer Microserviceumgebung gegenüber des Client-Server-Modells (siehe: Unterabschnitt 4.1.3) zu profitieren, ist das Projekt meines Erachtens zu klein und die zusätzliche Komplexität würde die Vorteile nicht aufwiegen. Daher wird diese Arbeit nach dem Client-Server-Modell entwickelt.

4.2 Client-Server Architektur

In Unterabschnitt 4.1.4 wurde gezeigt, dass eine Client-Server Architektur für dieses Projekt der beste Ansatz ist. In diesem Abschnitt wird gezeigt, wie dieser umgesetzt werden soll.

4.2.1 Architekturstil

Wie aus Abbildung 4.4 ersichtlich ist, läuft die Kommunikation nach dem Prinzip des Request-Response ab. Eine einseitige Kommunikation vom Server zum Client ist nicht notwendig. Daher bietet sich die Rest-Architektur über HTTP (siehe: Unterabschnitt 2.1.1) als Architekturstil an, da dieser auf dem Request-Response Verfahren basiert.

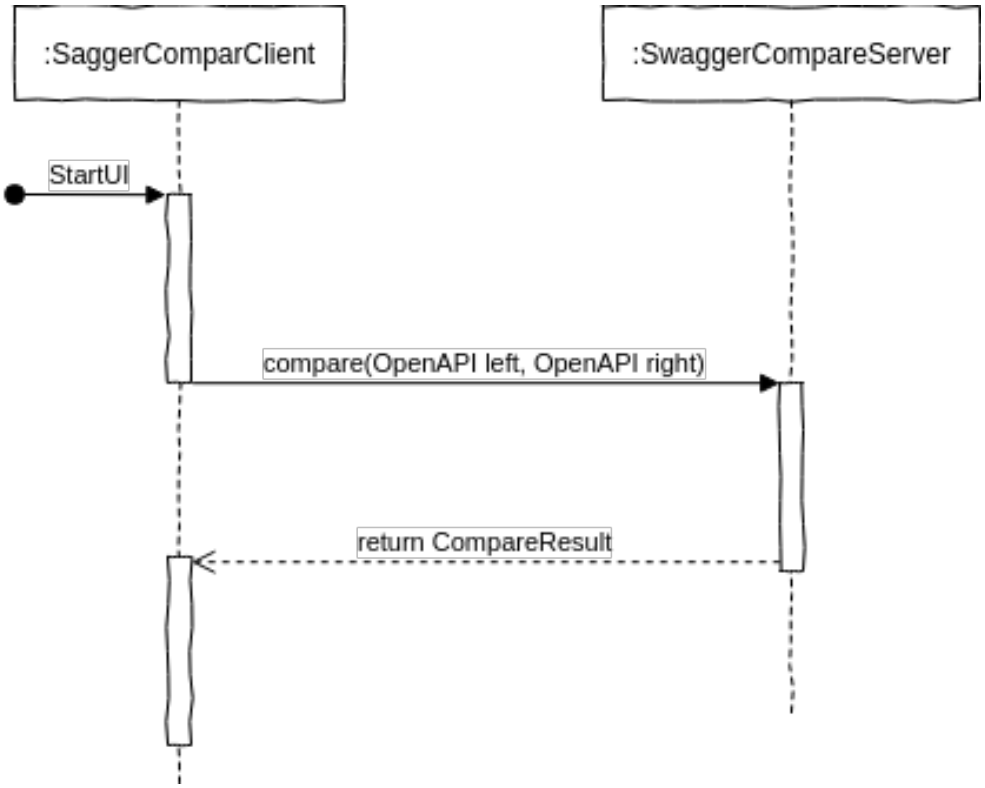

Abbildung 4.4: Kommunikation zwischen Client und Server

4.2.2 Highlevel View

Abbildung 4.5 zeigt den prinzipiellen Aufbau des Programms. Der Server stellt die UI zur Verfügung und führt den Backendcode aus. Mit einem Webbrowser kann die UI abgerufen und ausgeführt werden. UI und Backend kommunizieren dann im Request-Response-Verfahren.

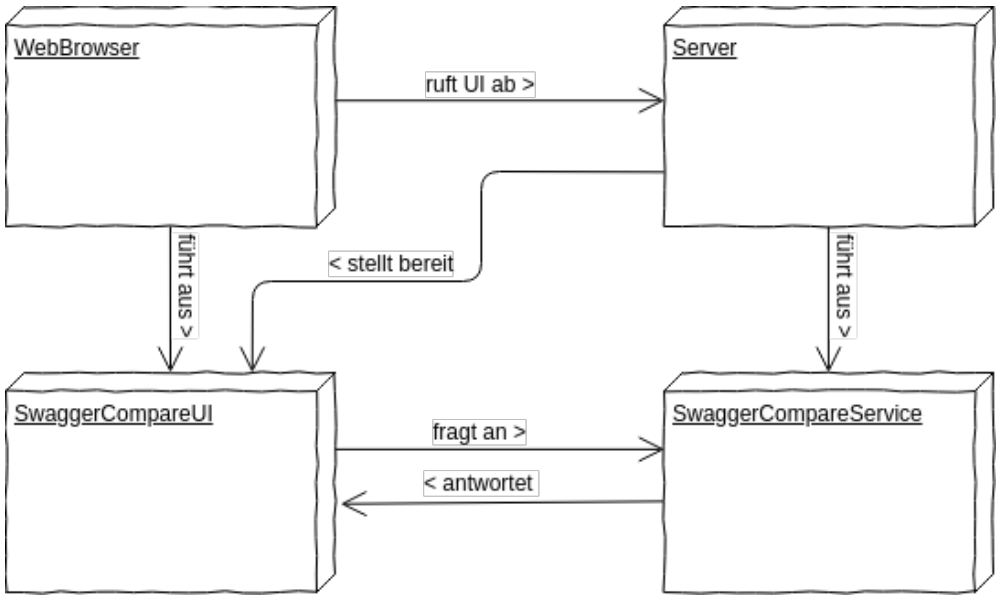

Abbildung 4.5: High Level View

4.3 Programmiersprachen und Frameworks

In diesem Abschnitt wird auf die eingesetzten Programmiersprachen und Frameworks eingegangen. Hierbei werden in Unterabschnitt 4.3.1 das Backend und in Unterabschnitt 4.3.2 das Frontend gesondert betrachtet.

4.3.1 Backend

In diesem Abschnitt wird in Unterunterabschnitt 4.3.1.1 auf die eingesetzte Programmiersprache und in Unterunterabschnitt 4.3.1.2 die eingesetzten Frameworks für das Backend eingegangen.

4.3.1.1 Programmiersprache

Wie aus Abschnitt 3.3 ersichtlich wird, steht der Swagger-Parser als Java-Bibliothek zur Verfügung. Da eine Neuimplementierung des Parsers in einer anderen Programmiersprache als

nicht zielführend erachtet wird, wird im Backend Java (siehe: Unterabschnitt 2.4.1) als Programmiersprache eingesetzt.

4.3.1.2 Frameworks

In diesem Unterabschnitt wird betrachtet, welche Frameworks eingesetzt werden. Diese werden kurz beschrieben und erläutert, wofür sie benötigt werden.

4.3.1.2.1 Swagger-Parser

Der Swagger-Parser (siehe: Unterabschnitt 3.3.1) dient zum Einlesen von Swagger-Definitionen und wandelt diese aus dem JSON-/YAML-Format in POJOs um. Da Java nativ keine Swagger-Definitionen einlesen kann, wird der Swagger-Parser hierfür verwendet.

4.3.1.2.2 Spring-Boot

Das Spring-Framework ist wohl der de facto Standard um Javaanwendungen zu entwickeln (vgl.: Walls (2016) S. 1). In diesem Projekt wird Spring Boot eingesetzt, um die Applikation als standalone Variante (mit integriertem Appserver) und als Webanwendung im WAR (Web Application Archive) Format bereit zu stellen. Zudem wird Spring als REST-Framework genutzt. Listing 4.1 zeigt einen minimalen Restcontroller in Spring.

```
1 @RestController
2 @RequestMapping("/mapping")
3 public class RestExampleController {
4
5   @GetMapping(value = "/function")
6   public String restFunction() {
7     return "Example";
8   }
9 }
```

Listing 4.1: Minimal Restcontroller Example

Zudem wird Spring für die Dependency Injection eingesetzt. Listing 4.2 zeigt eine Springcomponent, welche dann mittels Autowired, wie in Listing 4.3 oder Listing 4.4 gezeigt, genutzt werden kann. Da die Dependency Injection über den Konstruktor (wie in Listing 4.4 gezeigt) das Mocken (siehe: Absatz 4.3.1.2.4) einzelner oder mehrerer Komponenten in den Tests vereinfacht, wird dies der Injection über die Variablen (wie in Listing 4.3 gezeigt) vorgezogen.

```java
@Component
public class ExampleComponent {
  private String exampleValue = "Example";

  public String getExampleValue() {
    return exampleValue;
  }

  public void setExampleValue(String exampleValue) {
    this.exampleValue = exampleValue;
  }
}
```

Listing 4.2: Beispiel Spring Component

```java
public class AutwiredExampleController1 {
  @Autowired
  ExampleComponent exampleComponent;
}
```

Listing 4.3: Autowired Beispiel 1

```java
public class AutwiredExampleController2 {

  ExampleComponent exampleComponent;

  @Autowired
  public AutwiredExampleController2(
   ExampleComponent exampleComponent) {
    this.exampleComponent = exampleComponent;
  }
}
```

Listing 4.4: Autowired Beispiel 2

4.3.1.2.3 JUnit

JUnit ist die Standardbibliothek zum Schreiben automatisierter Tests in Java, denn „mehr als 60% aller quelloffenen Projekte unter GitHub referenzieren diese Bibliothek“ (siehe: Ullenboom (2014)). Auch in diesem Projekt wird JUnit für die automatisierten Tests eingesetzt.

4.3.1.2.4 Mockito

Bei Mockito handelt es sich um ein Mocking Framework (vgl. Faber (2018)). Es wird in diesem Projekt dafür eingesetzt, um in den JUnit Tests Abhängigkeiten zu mocken, um so nur die Funktionalität der zu testenden Methode zu prüfen, ohne direkt ein Integrationstest zu machen. Zudem wird es bei den Spring-Cloud-Contract-Tests (siehe: Absatz 4.3.1.2.5) eingesetzt, um einige Abhängigkeiten zu externen Bibliotheken zu mocken.

4.3.1.2.5 Spring Cloud Contract

Spring Cloud Contract ist ein Projekt, das helfen soll, nach dem Consumer Driven Contract Modell zu testen (vgl.: Pivotal Software, Inc. (2018)). In dieser Arbeit wird es eingesetzt um Consumer Driven Tests (kurz: CDT) des Backends umzusetzen. Listing 4.5 und Listing 4.6 zeigen beispielhaft einen CDT für den Restcontroller aus Listing 4.1.

```
1 public class exampleCdt {
2   @Before
3   public void setup() {
4     RestAssuredMockMvc.standaloneSetup(new RestExampleController());
5   }
6 }
```

Listing 4.5: Example Base CDC Test Class

```
1 org.springframework.cloud.contract.spec.Contract.make {
2   request {
3     method 'GET'
4     url '/mapping/function'
5   }
6   response {
7     status 200
8     body(
9       """Example"""
10    )
11  }
12 }
```

Listing 4.6: Example CDC Test

4.3.2 FrontEnd

In diesem Abschnitt wird in Unterunterabschnitt 4.3.2.1 auf die eingesetzte Programmiersprache und in Unterunterabschnitt 4.3.2.2 auf „Angular 5“ für das Frontend eingegangen.

4.3.2.1 Programmiersprache

In Unterabschnitt 4.1.4 wurde ermittelt, dass das Frontend im Webbrowser ausgeführt werden sollte. Unterabschnitt 4.2.2 zeigt die Unabhängigkeit von Client und Server in Bezug auf die Programmiersprache. Da als Framework für das Frontend Angular 5 (siehe: Unterunterabschnitt 4.3.2.2) eingesetzt wird, wird das Frontend in HTML, CSS und Typescript (siehe: Unterabschnitt 2.4.2) entwickelt.

4.3.2.2 Angular 5

„Angular ist ein Framework zur Entwicklung von Single-Page-Appliationen“, welches nativ das Konsumieren von Daten aus einer REST-Schnittstelle unterstützt (vgl.: Woiwode u. a. (2017) Vorwort). Da das Backend, wie in Absatz 4.3.1.2.2 gezeigt, eine REST-Schnittstelle bereitstellt, bietet sich Angular daher als Framework für das Frontend an.

4.4 Build und Infrastruktur

In diesem Abschnitt wird auf den Build und die Infrastruktur eingegangen. Es werden die verwendeten Tools aufgeführt und beschrieben, wofür sie benötigt werden.

4.4.1 Docker

Docker ist die weltweit führende Plattform, um Software in Containern auszuliefern (vgl.: Docker Inc. (2018)). In diesem Projekt wird Docker verwendet, um die Software einfach bereitstellen und deployen zu können.

4.4.2 Maven

In Unterunterabschnitt 4.3.1.1 wurde festgelegt, dass Java als Programmiersprache im Backend zum Einsatz kommt und in Unterunterabschnitt 4.3.2.2 wurde festgelegt, dass Angular 5 als Frontendframework zum Einsatz kommt. Daher wird ein Build-Tool benötigt, welches sowohl den Build von Java, als auch den Build des Angular-Frontends unterstützt. Maven unterstützt nativ den Build von Java-Programmen (vgl.: Apache Software Foundation (2018)).

Außerdem unterstützt Maven mit dem „frontend-maven-plugin“ den Build einer Angular-App (vgl.: Sletteberg (2018)). Des Weiteren unterstützt Maven mit dem „docker-maven-plugin“ den Build von Dockercontainern (vgl.: Culbertson u. a. (2018)).

4.4.3 Jenkins

Jenkins ist ein Opensource Server, welcher zur Automatisierung aller Aufgaben zum bauen, testen, ausliefern und installieren von Software genutzt werden kann (vgl.: Jenkins (2018) What is Jenkins?). Hierfür wird Jenkins auch in diesem Projekt eingesetzt.

4.5 Architektur und Umsetzung des technischen Prototyp

In diesem Abschnitt wird auf die Ziele, Architektur und Umsetzung des technischen Prototypen eingegangen. Hierfür werden zunächst in Unterabschnitt 4.5.1 die Ziele, die mit dem technischen Prototypen verfolgt werden, erläutert und in Unterabschnitt 4.5.2 Erfolgskriterien für diesen festgelegt. In Unterabschnitt 4.5.3 wird dann die Umsetzung des Builds und Deployments und in Unterabschnitt 4.5.4 und Unterabschnitt 4.5.5 die Umsetzung des Clients und Backends gezeigt. Zum Abschluss wird in Unterabschnitt 4.5.6 ein Resümee über den Erfolg des Prototypen gezogen.

4.5.1 Ziele des technischen Prototypen

Im technischen Prototypen soll zunächst gezeigt werden, dass das Projektziel, zwei verschiedener Versionen einer Swagger-Definition zu vergleichen, technisch umsetzbar ist. Dafür ist zu zeigen, dass mit den in Abschnitt 4.3 ausgewählten Programmiersprachen und Frameworks, sowie mit den in Abschnitt 4.4 gewählten Tools ein kleiner Teil des Zielprogramms umgesetzt werden kann. Dabei sollten alle Frameworks, Programmiersprachen und Tools mindestens einmal zum Einsatz kommen. Der technische Prototyp ist kein bugfreies, voll funktionstüchtiges, sauber implementiertes Programm! Er zeigt lediglich, dass es technisch möglich ist, das Projektziel zu erreichen. Jedoch sollte der Build und das Deployment bereits im finalen Zustand sein, damit sich diese während der eigentlichen Implementation des Projektes nicht mehr ändern.

4.5.2 Erfolgskriterien für den technischen Prototypen

1. Client:

   a) Es können zwei Swagger-Definitionen (die sich auf einem HTTP-Server befinden) ausgewählt werden.
   b) Die ausgewählten URLs zu den Definitionen werden per REST-Schnittstelle an den Server gesendet.
   c) Die Antwort des Servers wird grafisch aufgearbeitet und dargestellt.

2. Backend:
   a) Es können zwei URLs zu Swagger-Definitionen per Rest-Schnittstelle empfangen werden.
   b) Die URLs werden in POJOs umgewandelt.
   c) Die beiden OpenAPI-Dokumente werden verglichen und es wird ermittelt, welche Pfade gelöscht, hinzugefügt oder verändert wurden, bzw. gleich geblieben sind.
   d) Das Ergebnis wird per REST-Schnittstelle an den Client übertragen.

3. Build und Deployment:
   a) Das gesamte Projekt lässt sich mit Maven bauen.
   b) Ein Dockercontainer wird erstellt, der den Server bereitstellt.
   c) Das Projekt wird bei jedem Push auf das Repository vom Jenkins gebaut.
   d) Erfolgt der Push auf den Master, wird der Dockercontainer deployed.
   e) Schlägt ein Build fehl, wird durch den Buildserver eine Email versendet.

4.5.3 Build und Deployment

In Unterabschnitt 4.5.2 wurden folgende Kriterien an den Build und das Deployment gestellt:

1. Das gesamte Projekt lässt sich mit Maven bauen.
2. Ein Dockercontainer wird erstellt, der den Server bereitstellt.
3. Das Projekt wird bei jedem Push auf das Repository vom Jenkins gebaut.
4. Erfolgt der Push auf den Master, wird der Dockercontainer deployed.
5. Schlägt ein Build fehl, wird durch den Buildserver eine Email versendet.

Im folgenden wird gezeigt, wie dies mithilfe von Maven und Jenkins umgesetzt wurde.

4.5.3.1 Maven Build

In Abschnitt 4.4 wurde festgelegt, dass Maven als Build-Tool genutzt wird und festgestellt, dass Maven sowohl den Build von Docker-Containern, als auch des Angularfrontends unterstützt. Abbildung 4.6 zeigt die Struktur des Maven Multi Module Builds. Durch den Build des „swagger-compare-root“-Projektes werden auch alle abhängigen Projekte gebaut.
Abbildung 4.7 zeigt die Abhängigkeiten zwischen den einzelnen Modulen. Diese werden durch Mavenbuild in der richtigen Reihenfolge gebaut. Zusätzlich wird beim Build des „swagger-compare-ui“-Modules ein Angularbuild ausgeführt. Listing 4.7 zeigt den relevanten Ausschnitt aus der entsprechenden „pom.xml“. Zudem wird beim Build des „swagger-compare-standalone“-Modules ein Dockercontainer erzeugt. Der relevante Ausschnitt aus der „pom.xml“ ist in Listing 4.8 gezeigt.

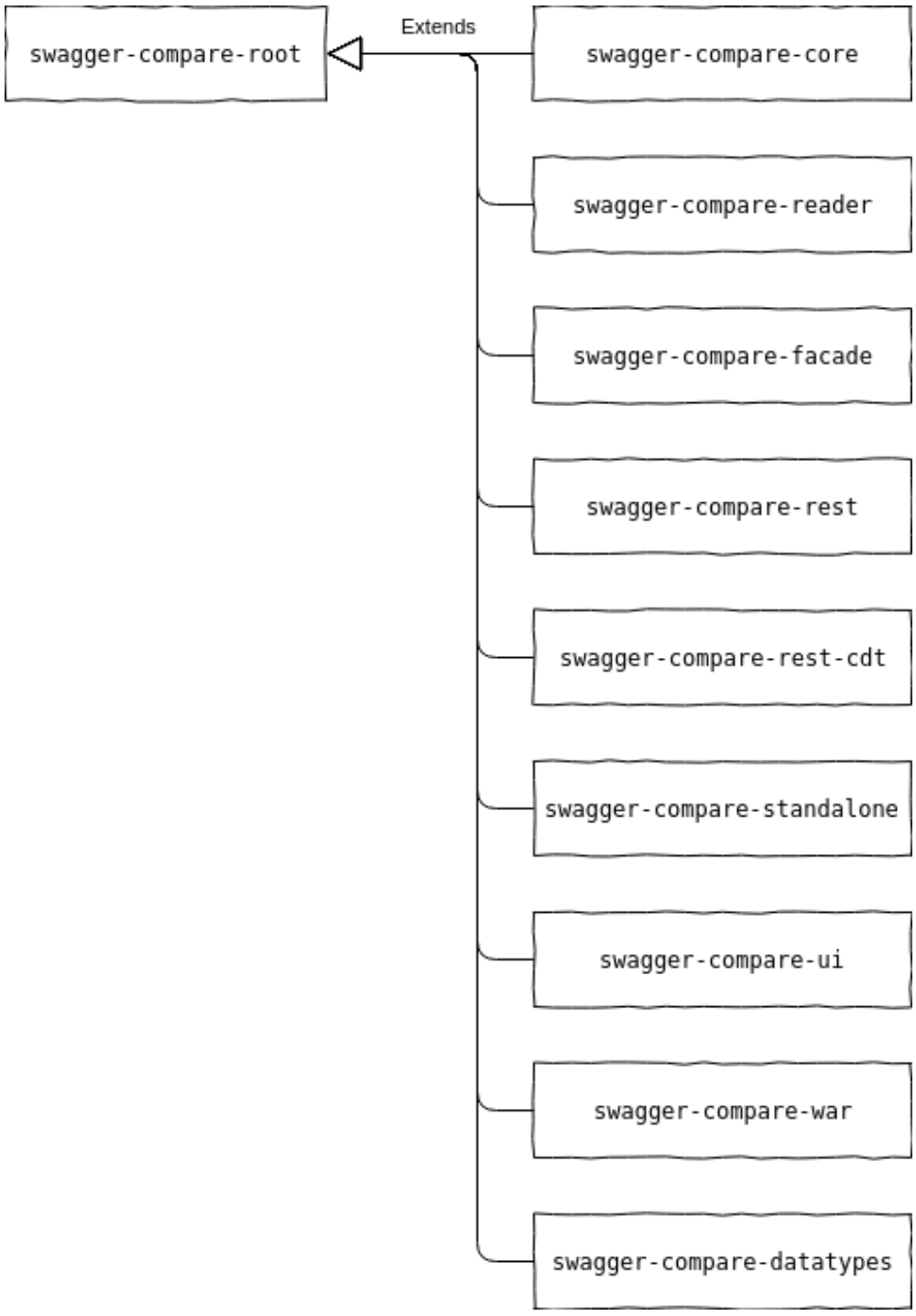

Abbildung 4.6: Maven Multi Module Struktur

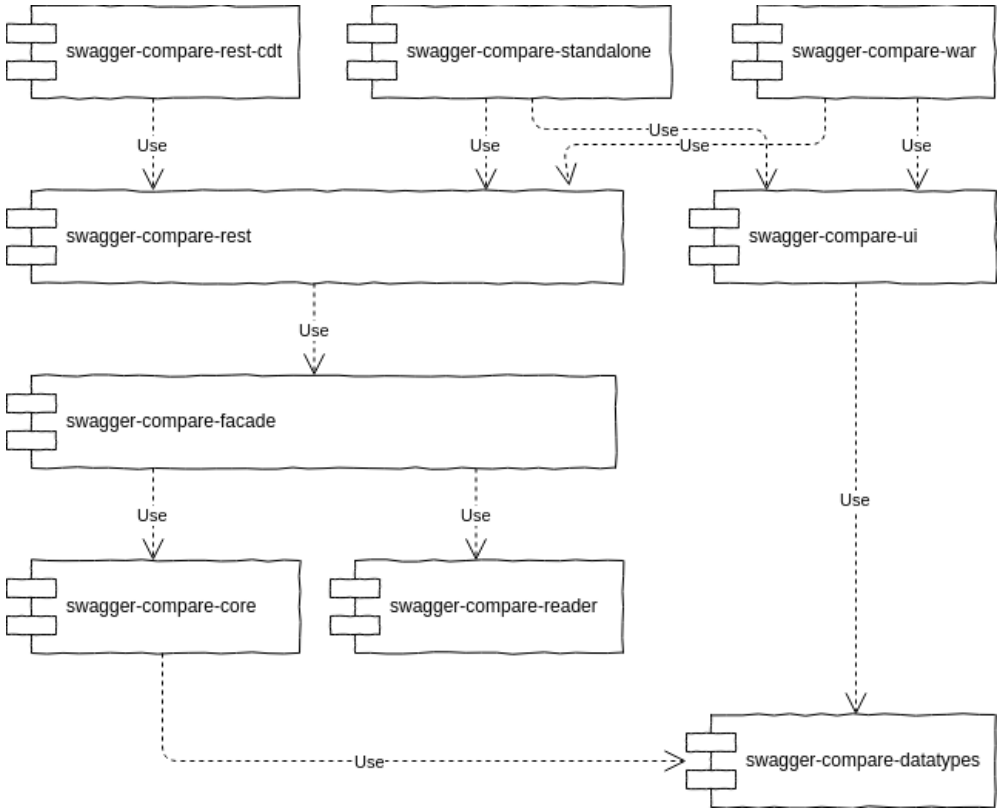

Abbildung 4.7: Maven Modul Abhängigkeiten

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>de.haug-dev.swagger-compare</groupId>
    <artifactId>swagger-compare-root</artifactId>
    <version>0.1.1</version>
  </parent>
  <artifactId>swagger-compare-ui</artifactId>
  <packaging>jar</packaging>
  [...]
  <build>
    <plugins>
```

```
      [...]
      <plugin>
        <groupId>com.github.eirslett</groupId>
        <artifactId>frontend-maven-plugin</artifactId>
        <executions>
          <execution>
            <id>install node and npm</id>
            <goals>
              <goal>install-node-and-npm</goal>
            </goals>
            <configuration>
              <nodeVersion>${node.version}</nodeVersion>
              <npmVersion>${npm.version}</npmVersion>
            </configuration>
          </execution>
          <execution>
            <id>npm install</id>
            <goals>
              <goal>npm</goal>
            </goals>
            <configuration>
              <arguments>install</arguments>
            </configuration>
          </execution>
          <execution>
            <id>npm run build</id>
            <goals>
              <goal>npm</goal>
            </goals>
            <configuration>
              <arguments>run build</arguments>
            </configuration>
          </execution>
        </executions>
      </plugin>
      [...]
  </build>
</project>
```

Listing 4.7: swagger-compare-ui/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>de.haug-dev.swagger-compare</groupId>
    <artifactId>swagger-compare-root</artifactId>
    <version>0.1.1</version>
  </parent>
  <artifactId>swagger-compare-standalone</artifactId>
  <packaging>jar</packaging>
  <profiles>
    <profile>
      <id>dockerBuild</id>
      <build>
        <plugins>
          <plugin>
            <groupId>com.spotify</groupId>
            <artifactId>docker-maven-plugin</artifactId>
            <executions>
              <execution>
                <id>build-image</id>
                <phase>package</phase>
                <goals>
                  <goal>build</goal>
                </goals>
              </execution>
            </executions>
            <configuration>
              <imageName>torbenhaug/swagger-compare</imageName>
              <baseImage>openjdk:8-jre</baseImage>
              <entryPoint>
                ["java", "-jar", "/${project.build.finalName}.jar"]
              </entryPoint>
              <resources>
                <resource>
                  <targetPath>/</targetPath>
```

```
                    <directory>${project.build.directory}</directory>
                    <include>${project.build.finalName}.jar</include>
                  </resource>
                </resources>
                <forceTags>true</forceTags>
                <imageTags>
                  <imageTag>${project.version}</imageTag>
                  <imageTag>latest</imageTag>
                </imageTags>
                <exposes>
                  <expose>8080</expose>
                </exposes>
              </configuration>
            </plugin>
          </plugins>
        </build>
      </profile>
    </profiles>
    [...]
</project>
```

Listing 4.8: swagger-compare-standalone/pom.xml

4.5.3.2 Jenkins Build

Um die Anforderungen an den Jenkins Build umzusetzen, wurden zwei automatische Builds auf dem Buildserver eingerichtet, zum einen ein Continous Build und zum anderen ein Nightly Build.

4.5.3.2.1 Continous Build

Bei dem Continous Build handelt es sich um eine „Multibranch Pipeline“, welche mit dem Githubrepository verknüpft ist. So wird automatisch für jeden Branch und jeden Pullrequest ein Contious Build erzeugt. Wird etwas auf das Repository gepusht, wird der Build durch einen Webhook getriggert. Listing 4.9 zeigt, wie der Contious Build abläuft. Zunächst wird das Repository ausgecheckt, dann wird der Maven Build durchgeführt und wenn der Build fehlschlägt, wird eine Email an die hinterlegte Adresse versendet.

```
pipeline {
  tools {
```

```
3     maven "M3"
4     jdk "JDK8U152"
5   }
6
7   post {
8     always {
9       cleanWs();
10      step([$class: 'Mailer',
11        notifyEveryUnstableBuild: true,
12        recipients: "info@haug-dev.de",
13        sendToIndividuals: true])
14    }
15  }
16
17  agent any
18  stages {
19    stage('Checkout') {
20      steps {
21        checkout scm
22      }
23    }
24    stage('Build') {
25      steps {
26      withMaven(
27        maven: 'M3',
28        mavenLocalRepo: '.repository') {
29          sh "mvn clean install -f swagger-compare/pom.xml"
30        }
31      }
32    }
33  }
34}
```

Listing 4.9: Jenkinsfile

4.5.3.2.2 Nightly Build

Bei dem Nightly Build handelt es sich um ein normales Pipeline Projekt, welches täglich zwischen 1 und 5 Uhr ausgeführt wird. Da lediglich der Master gebaut werden muss, ist eine „Multibranch Pipeline“ unnötig, und da das Projekt zeitgesteuert ausgeführt wird, muss es auch

nicht getriggert werden. Listing 4.10 zeigt wie der Contious Build abläuft. Zunächst wird das Repository ausgecheckt, dann wird der Maven Build durchgeführt. Während des Maven Builds wird auch der Dockercontainer erzeugt.Im nächsten Schritt wird dieser Dockercontainer dann auf dem lokalen Server deployed und bei Docker Hub bereitgestellt. Wenn der Build fehlschlägt wird eine Email an die hinterlegte Adresse versendet.

```
pipeline {
  tools {
    maven "M3"
    jdk "JDK8U152"
  }

  post {
    always {
      cleanWs();
      step([$class: 'Mailer',
        notifyEveryUnstableBuild: true,
        recipients: "info@haug-dev.de",
        sendToIndividuals: true])
    }
  }

  agent any
  stages {
    stage('Checkout') {
      steps {
        checkout scm
      }
    }
    stage('BuildBranch') {
      steps {
        withMaven(
        maven: 'M3',
        mavenLocalRepo: '.repository') {
          sh "mvn clean install -f swagger-compare/pom.xml -P
dockerBuild"
        }
      }
    }
    stage('Deploy') {
```

```
35        steps {
36          withCredentials(
37           [usernamePassword(
38             credentialsId: 'DockerHub',
39             usernameVariable: 'USERNAME',
40             passwordVariable: 'PASSWORD'
41           )]) {
42            sh "docker stop swagger-compare-instance ||
43 ␣␣␣␣␣␣␣␣␣␣␣␣true && docker rm swagger-compare-instance || true"
44            sh "docker run --name swagger-compare-instance -d
45 ␣␣␣␣␣␣␣␣␣␣␣␣-p 7070:8080 torbenhaug/swagger-compare:latest"
46            sh "docker login -u $USERNAME -p $PASSWORD &&
47 ␣␣␣␣␣␣␣␣␣␣␣␣docker push torbenhaug/swagger-compare:${version()} &&
48 ␣␣␣␣␣␣␣␣␣␣␣␣docker push torbenhaug/swagger-compare:latest"
49          }
50        }
51      }
52    }
53 }
54
55 def version() {
56   def matcher = readFile('swagger-compare/pom.xml') =~
57                 '<version>(.+)</version>'
58   matcher ? matcher[0][1].trim() : null
59 }
```

Listing 4.10: JenkinsfileNB

4.5.3.3 Fazit

Es konnte gezeigt werden, dass alle in Unterabschnitt 4.5.2 an den Build und das Deployment gestellten Anforderungen erfüllt werden können.

4.5.4 Technischer Prototype des Clients

In Unterabschnitt 4.5.2 wurden folgende Kriterien an den Client gestellt:

1. Es können zwei Swagger-Definitionen (die sich auf einem HTTP-Server befinden) ausgewählt werden.

2. Die ausgewählten URLs zu den Definitionen werden per REST-Schnittstelle an den Server gesendet.

3. Die Antwort des Servers wird grafisch aufgearbeitet und dargestellt

Der Client befindet sich vollständig im Modul „swagger-compare-ui“. Im folgenden wird gezeigt, wie dieser umgesetzt wurde.

4.5.4.1 GUI

Abbildung 4.8 zeigt einen Screenshot des technische Prototypen. Es können zwei URLs in Textboxen eingeben werden. Mit dem Submitbutton wird die Verarbeitung angestoßen und anschließend wird das Ergebnis unter Result ausgegeben.

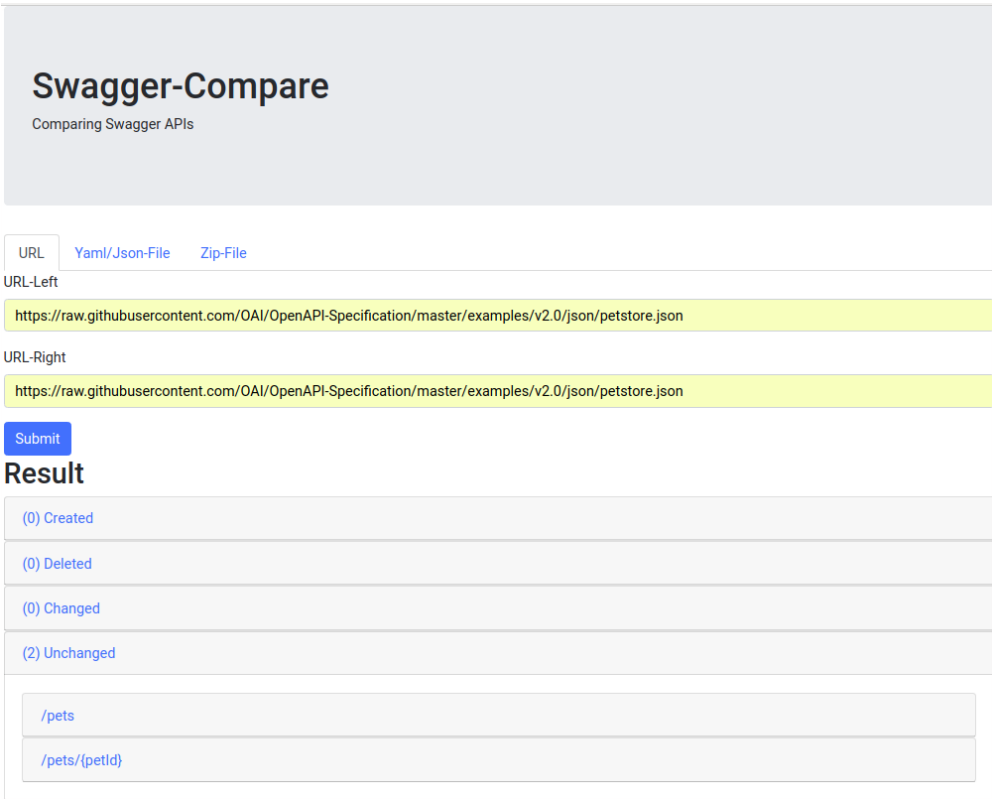

Abbildung 4.8: Screenshot des Prototypen

4.5.4.2 Kommunikation mit dem Server

Der Client kommuniziert mit dem Server via REST. Das eingesetzte Framework Angular unterstützt REST. Listing 4.11 zeigt wie dies genutzt wurde.

```
import {Component, isDevMode, OnInit} from '@angular/core';
import {HttpClient, HttpErrorResponse, HttpResponse} from
       "@angular/common/http";
import {FlashMessagesService} from "ngx-flash-messages";
import {TraceBoxDataService} from
       "../trace-box/trace-box.data.service";
import {CompareResultDataService} from
       "../compare-result/compare-result.data.service";

@Component({
  selector: 'load-url-form',
  templateUrl: './load-url-form.component.html',
  styleUrls: ['./load-url-form.component.scss']
})
export class LoadUrlFormComponent implements OnInit {

  constructor(
    private http: HttpClient,
    private flashMessagesService: FlashMessagesService,
    private traceBoxDataService: TraceBoxDataService,
    private compareResultDataService: CompareResultDataService) {

  }

  ngOnInit() {
  }

  onSubmit(urlLeft: string, urlRight: string){
    var body = {
      urlLeft: urlLeft,
      urlRight: urlRight
    }
    this.http.post("/api/compare",body).subscribe(
      (data) => {
        this.compareResultDataService.showResult(data);
        if(isDevMode()){
```

```
37            this.traceBoxDataService.showTrace(JSON.stringify(data));
38          }else {
39            this.traceBoxDataService.showTrace("");
40          }
41        },
42        (error: HttpErrorResponse) => {
43          this.compareResultDataService.showResult("");
44          this.traceBoxDataService.showTrace(error);
45        }
46      );
47    }
48  }
```

Listing 4.11: Rest Calls

4.5.4.3 Fazit

Es konnte gezeigt werden, dass alle in Unterabschnitt 4.5.2 an den Client gestellten Anforderungen erfüllt werden können.

4.5.5 Technischer Prototype des Backends

In Unterabschnitt 4.5.2 wurden folgende Kriterien an das Backend gestellt:

1. Es können zwei URLs Swagger-Definitionen per Rest-Schnittstelle empfangen werden.
2. Die URLs werden in OpenAPI-Dokumente umgewandelt.
3. Die beiden OpenAPI-Dokumente werden verglichen und es wird ermittelt, welche Pfade gelöscht, hinzugefügt oder verändert wurden, bzw. gleich geblieben sind.
4. Das Ergebnis wird per REST-Schnittstelle an den Client übertragen.

Das Backend ist über die Module swagger-compare-core, swagger-compare-datatypes, swagger-compare-reader, swagger-compare-facade, swagger-compare-rest, swagger-compare-rest-cdt, swagger-compare-standalone und swagger-compare-war realisiert.
Im Folgenden wird beschrieben, welche Funktion die einzelnen Module haben und und wie sie zusammenspielen, um die Anforderungen zu erfüllen.

4.5.5.1 Modulübersicht

Im folgenden Abschnitt wird auf die Module eingegangen. Alle Klassen der Module sind durch Unittests getestet.

4.5.5.1.1 swagger-compare-core

Das Modul „swagger-compare-core“ bildet die Kernkomponente des Projektes. Hier werden die Swagger-Definitionen verglichen und ein Vergleichsergebnis ermittelt.Abbildung 4.9 zeigt ein Klassendiagramm dieses Moduls. Die Einstiegsklasse ist „SwaggerCompareCore“. Diese dient zur Orchestrierung und Kapselung des Moduls. Der „Normailzer“ normalisiert die Pfade, damit Änderungen an der Benennung von Pfadvariablen nicht als solche von einem „SwaggerCompareProcessor“ erkannt werden.
Der „UnchangedPathFinder“ findet aus der Eingabemenge alle nicht geänderten Pfade, der „ChangedPathFinder“ findet aus der Eingabemenge alle geänderten Pfade, der „DeletedPathFinder“ findet aus der Eingabemenge alle gelöschten Pfade und der „CreatedPathFinder“ findet aus der Eingabemenge alle erstellten Pfade.
Abbildung 4.10 zeigt den Verarbeitungsablauf innerhalb des Moduls.

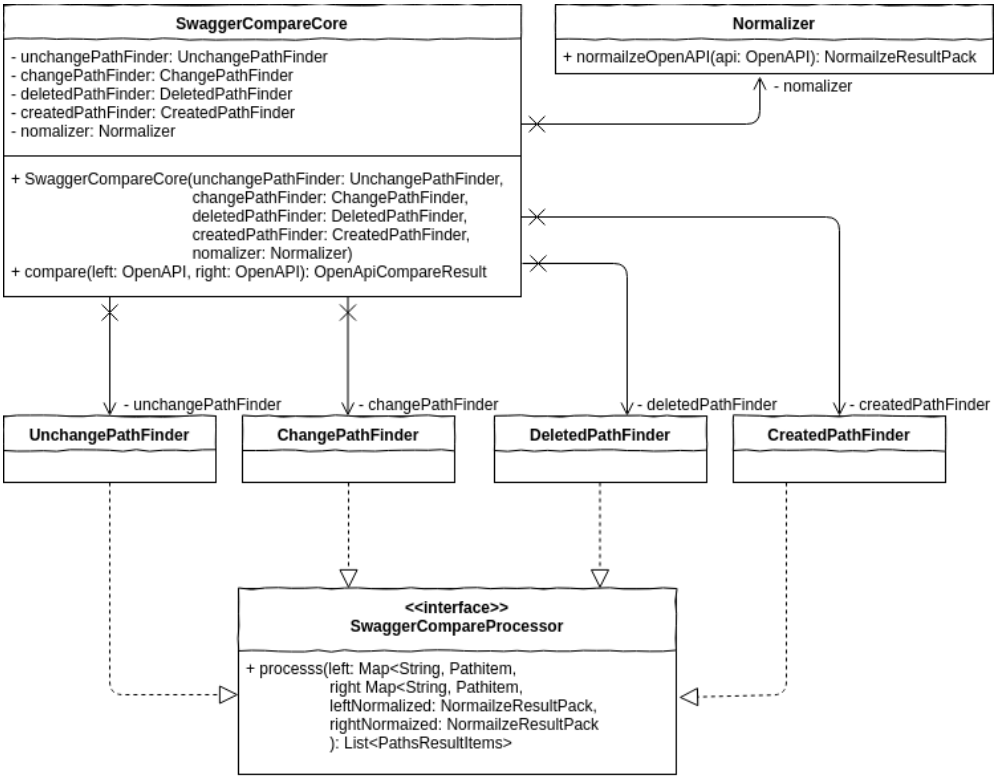

Abbildung 4.9: Klassendiagramm technischer Prototype: swagger-compare-core

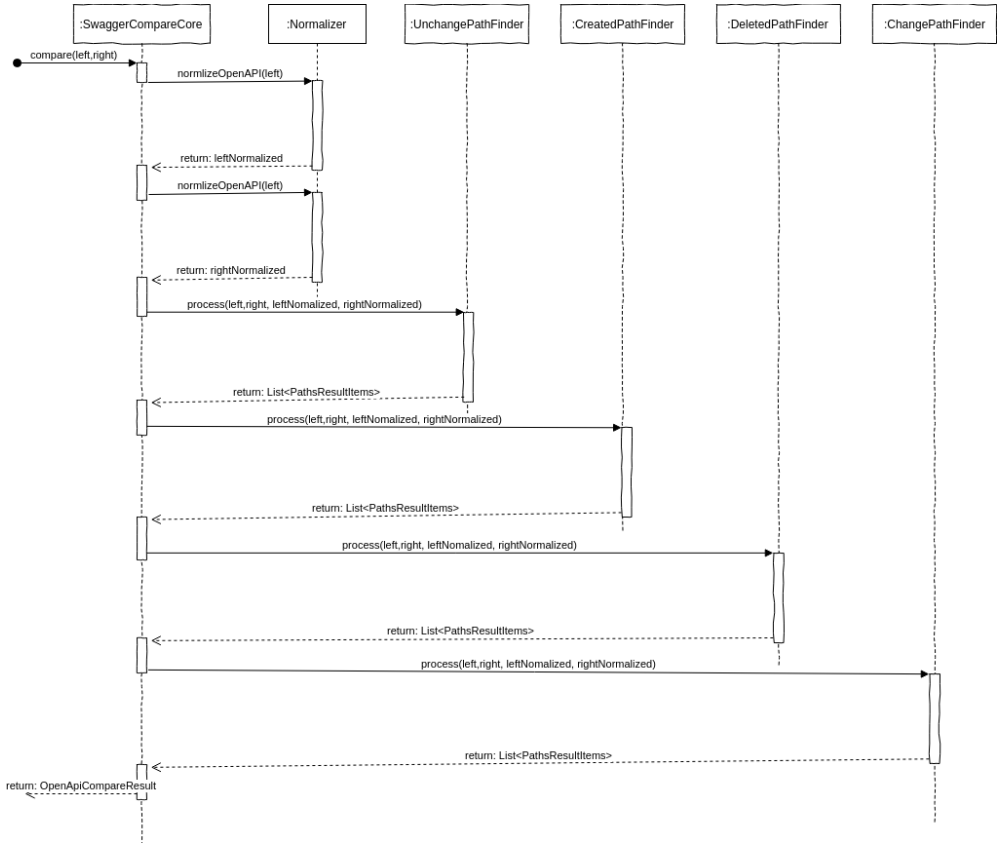

Abbildung 4.10: Sequenzdiagramm technischer Prototype: swagger-compare-core

4.5.5.1.2 swagger-compare-datatypes

Das Module „swagger-compare-datatypes“ enthält die Datentypen, die das Vergleichsergebnis repräsentieren. Abbildung 4.11 zeigt ein Klassendiagramm dieses Moduls.

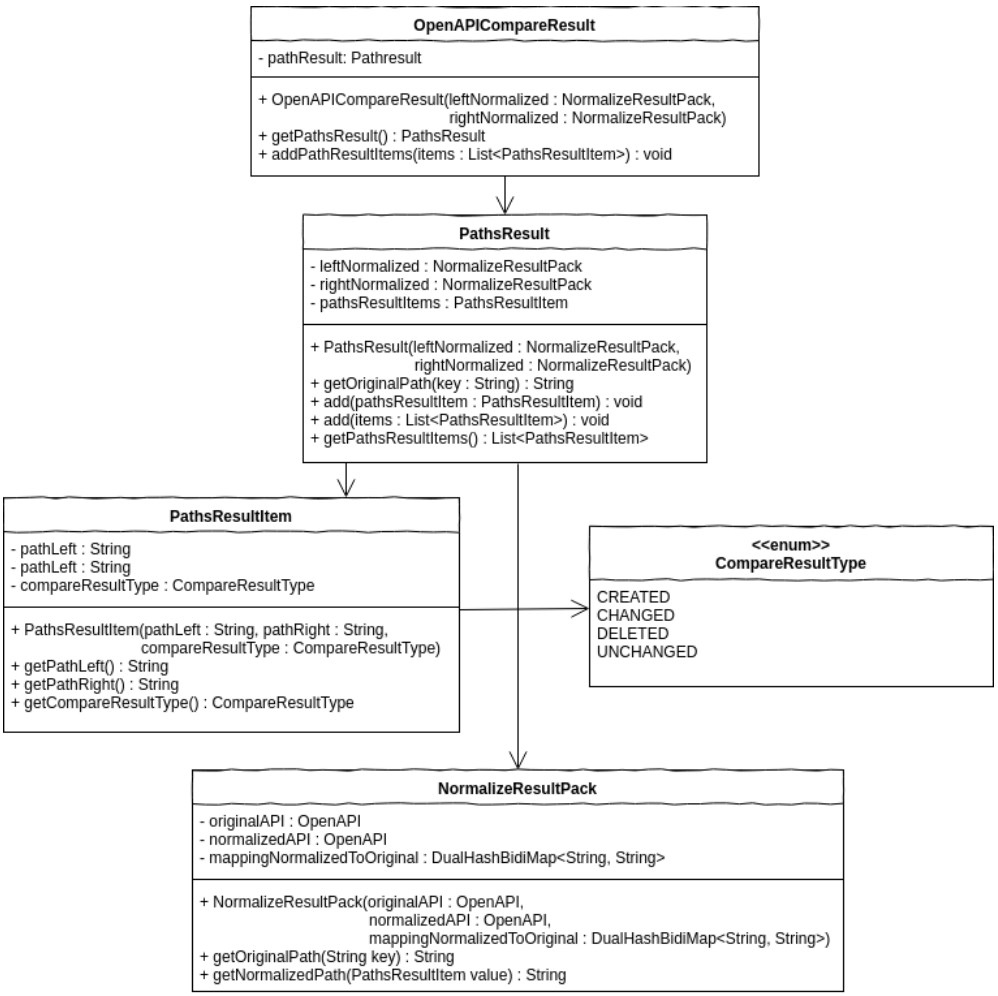

Abbildung 4.11: Klassendiagramm technischer Prototype: swagger-compare-datatypes

4.5.5.1.3 swagger-compare-reader

Das Modul „swagger-compare-reader“ dient dem Einlesen der Swagger-Definitionen. Abbildung 4.12 zeigt die interne Struktur des Moduls. Die Hauptklasse des Moduls ist „SwaggerCompareReader“. Die Klasse „SwaggerCompareReader“ wird genutzt, um die Dependency-Injection des OpenAPIV3Parsers mittels Spring zu gewährleisten.

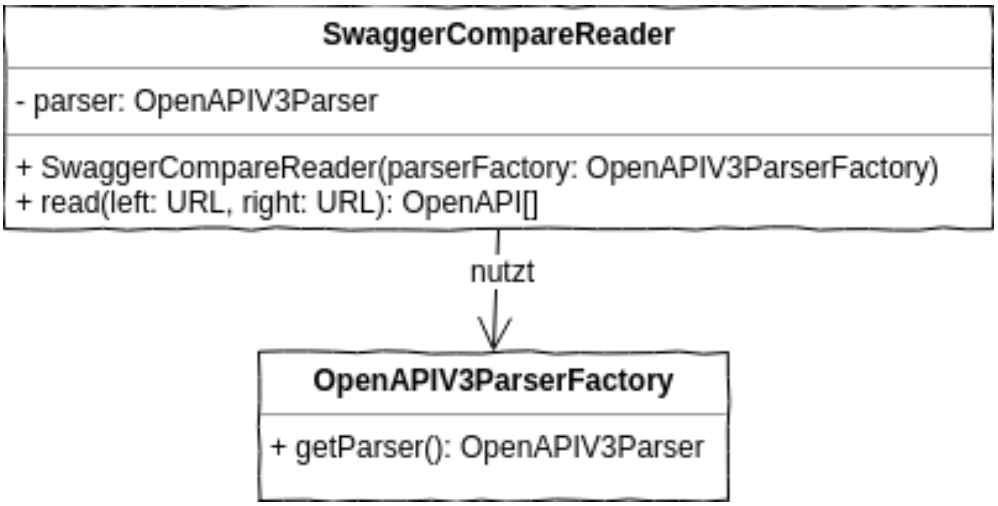

Abbildung 4.12: Klassendiagramm technischer Prototype: swagger-compare-reader

4.5.5.1.4 swagger-compare-facade

Das Modul „swagger-compare-facade“ kapselt die Businesslogik, die sich in den Modulen „swagger-compare-reader“ und „swagger-compare-core“ befindet. Zudem orchestriert dieses Modul die anderen beiden. Abbildung 4.13 zeigt die interne Struktur des Moduls.

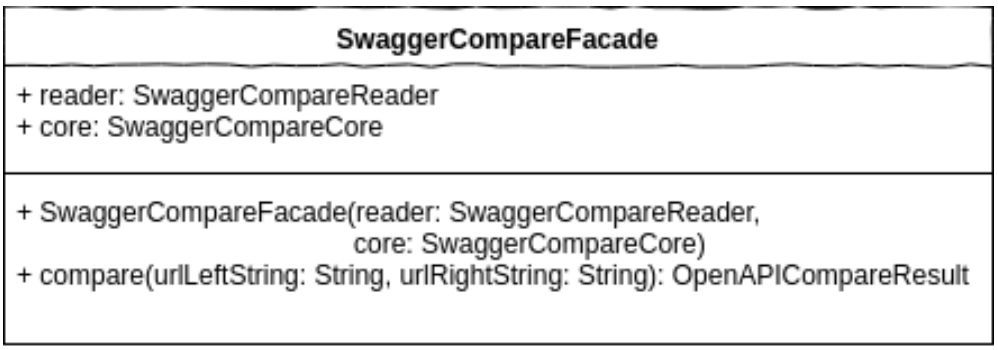

Abbildung 4.13: Klassendiagramm technischer Prototype: swagger-compare-facade

4.5.5.1.5 swagger-compare-rest

Das Modul „swagger-compare-rest“ kapselt die REST-Funktionalitäten. Abbildung 4.14 zeigt die interne Struktur des Moduls. Der „CompareController“ stellt die Rest-Schnittstelle bereit. Die Klasse „CompareRequest“ ist ein DTO für die Restanfrage. Der „RestResponseEntityExep-

tionHandler“ ist für das Fehlermanagement zuständig. Die Klasse „ErrorMessage“ ist ein DTO und symbolisiert einen Error-Response.

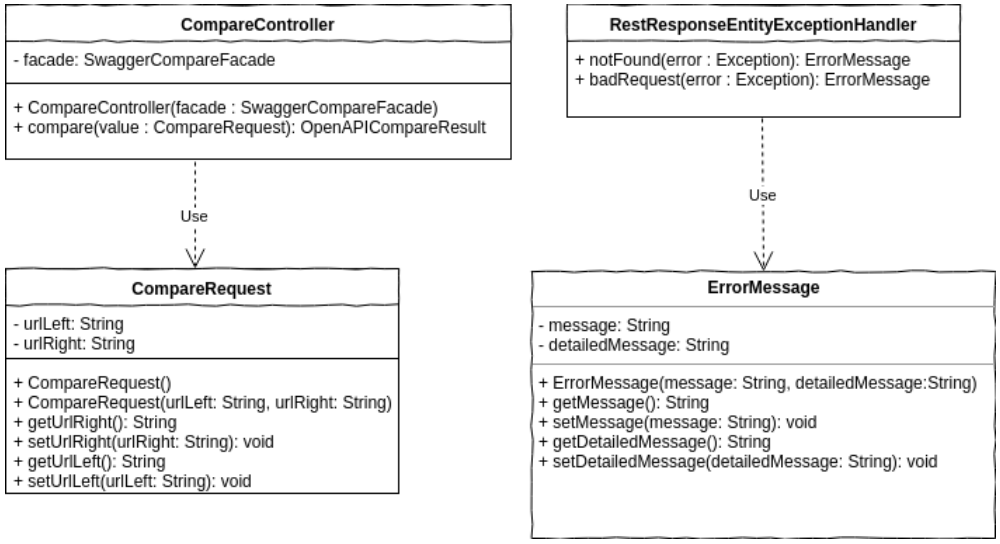

Abbildung 4.14: Klassendiagramm technischer Prototype: swagger-compare-rest

4.5.5.1.6 swagger-compare-rest-cdt

Das Modul „swagger-compare-rest-cdt“ ist für die Consumer-Driven-Tests zuständig. Diese werden auf Basis von „Spring Cloud Contract“ erstellt (siehe: Absatz 4.3.1.2.5). Hierbei handelt es sich um fast vollständige Integrationstests, lediglich der „Swagger-Parser“ wird gemockt, um auf bestimmte Anfragen gezielt reagieren zu können.

4.5.5.1.7 swagger-compare-standalone

Das Modul „swagger-compare-standalone“ stellt den Rest-Controller (siehe: Absatz 4.5.5.1.5) und die GUI (siehe: Unterunterabschnitt 4.5.4.1) als Standalone-Application zur Verfügung. Zudem wird die Applikation hier als Docker-Container bereitgestellt (siehe: Unterunterabschnitt 4.5.3.1).

4.5.5.1.8 swagger-compare-war

Das Modul „swagger-compare-war“ stellt den Rest-Controller (siehe: Absatz 4.5.5.1.5) und die GUI (siehe: Unterunterabschnitt 4.5.4.1) als WAR-File zur Verfügung. Dieses kann dann in einem beliebigen Application-Server bereitgestellt werden. Es ist vorrangig für die Entwicklung

gedacht, da der Redeploy eines WAR-Files in einem laufenden Application-Server erheblich schneller ist, als das ausführen der Standalonevariante (siehe: Absatz 4.5.5.1.7).

4.5.5.2 Verarbeitungsablauf einer Anfrage

In Unterunterabschnitt 4.5.5.1 wurde gezeigt, wie die Verarbeitung innerhalb eines jeden Moduls abläuft. In Abbildung 4.15 wird veranschaulicht, wie die Interaktion zwischen den Modulen abläuft.

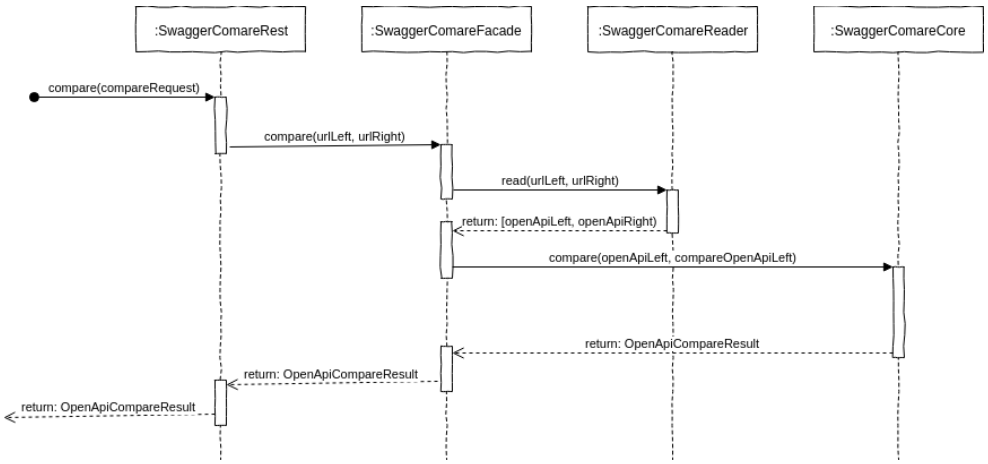

Abbildung 4.15: Sequenzdiagramm technischer Prototype: Backend

4.5.5.3 Fazit

Es konnte gezeigt werden, dass alle in Unterabschnitt 4.5.2 an das Backend gestellten Anforderungen erfüllt werden konnten.

4.5.6 Ergebnis

Alle in Unterabschnitt 4.5.2 aufgestellten Kriterien konnten erfüllt werden. Es wurde ein lauffähiger technischer Prototyp erstellt, der einen minimalen Vergleich durchführt. Zudem stellt dieser das Ergebnis in einer GUI dar. Des Weiteren wurde eine belastbare Grundarchitektur geschaffen und alle rein technischen Hürden, wie der Build und das Deployment konnten gelöst werden.

4.6 Architektur und Umsetzung des fachlichen Prototypen

In diesem Abschnitt wird auf die Ziele, Architektur und Umsetzung des fachlichen Prototypen eingegangen. Hierfür werden zunächst in Unterabschnitt 4.6.1 die Ziele, die mit dem fachlichen Prototypen verfolgt werden, erläutert und in Unterabschnitt 4.6.2 Erfolgskriterien für diesen festgelegt. In Unterabschnitt 4.6.3 wird dann die Umsetzung REST-Schnittstelle und in Unterabschnitt 4.6.4 und Unterabschnitt 4.6.5 die Umsetzung des Clients und Backends gezeigt.

4.6.1 Ziele des fachlichen Prototypen

Im fachlichen Prototypen soll gezeigt werden, dass mit der gewählten Architektur (siehe: Abschnitt 4.1) das Projektziel, zwei verschiedene Versionen einer Swagger-Definition zu vergleichen, fachlich umsetzbar ist. Dafür ist zu zeigen, dass die in Abschnitt 3.1 ermittelten Maßnahmen zur Erkennung und Bewertung von Unterschieden in zwei Swagger-Definitionen umgesetzt werden können. Zudem sollte eine stabile Schnittstelle zwischen Back- und Frontend ermöglicht werden und eine UI auf Basis dieser Schnittstelle bereitgestellt werden, die eine verständliche Ausgabe der ermittelten Unterschiede ermöglicht.

4.6.2 Erfolgskriterien für den fachlichen Prototypen

1. Client:
    a) Es können zwei Swagger-Definitionen (die sich auf einem HTTP-Server befinden) ausgewählt werden.
    b) Es können zwei Swagger-Definitionen (die sich lokal auf dem Client befinden) ausgewählt werden.
    c) Die ausgewählten URLs zu den Definitionen oder Dateien werden per REST-Schnittstelle an den Server gesendet.
    d) Der Client empfängt die Antwort des Servers.
    e) Die Antwort des Servers wird grafisch aufgearbeitet und dargestellt.
2. Backend:
    a) Es können zwei URLs zu Swagger-Definitionen oder Dateien per Rest-Schnittstelle empfangen werden
    b) Die URLs oder Dateien werden in POJOs umgewandelt.

c) Die beiden OpenAPI-Dokumente werden, gemäß der in Unterabschnitt 3.1.6 festgelegten Kriterien, verglichen.

d) Das Ergebnis wird per REST-Schnittstelle an den Client übertragen.

e) Die REST-Schnittstelle im wesentlich Änderungsunanfällig.

4.6.3 REST-Schnittstelle

An die REST-Schnittstelle werden in Unterabschnitt 4.6.2 die folgenden Kriterien gestellt:

1. Client:
   a) Die ausgewählten URLs zu den Definitionen oder die Dateien werden per REST-Schnittstelle an den Server gesendet.
   b) Der Client empfängt die Antwort des Servers.
2. Backend:
   a) Es können zwei URLs zu Swagger-Definitionen oder Dateien per Rest-Schnittstelle empfangen werden.
   b) Das Ergebnis wird per REST-Schnittstelle an den Client übertragen.
   c) Die REST-Schnittstelle ist im wesentlichen änderungsunanfällig.

in diesem Abschnitt wird darauf eingegangen, wie diese Ziele erreicht werden können.

4.6.3.1 Ergebnismenge

Um eine Schnittstelle zwischen Client und Server definieren zu können, wird zunächst ermittelt, wie das Ergebnis eines Vergleichs zweier Swagger-Definitionen aussieht. In Abschnitt 3.1 wurde ermittelt, dass eine Swagger-Definition baumartig aufgebaut ist und die Schnittstelle soll stabil und im wesentlichen änderungsunanfällig sein. Daher bietet sich ein Baum an, der über standardisierte Knoten und Blätter verfügt. Dieser kann rekursiv durchsucht werden und wenn ein Blatt oder ein Knoten zu der Ergebnismenge hinzukommt, muss die Schnittstelle nicht geändert werden. Abbildung 4.16 zeigt beispielhaft, wie solch ein Baum aussehen kann. An die Blätter und Knoten dieses Baumes können dann als Attribute die Zustände nach einer Operation (siehe: Unterabschnitt 3.1.3), sowie das Kritikalitätslevel (siehe: Unterabschnitt 3.1.4) bereitgestellt werden. Bei einem Blatt kann zusätzlich der Wert des Blattes vor und nach der Operation als Attribut hinzugefügt werden.

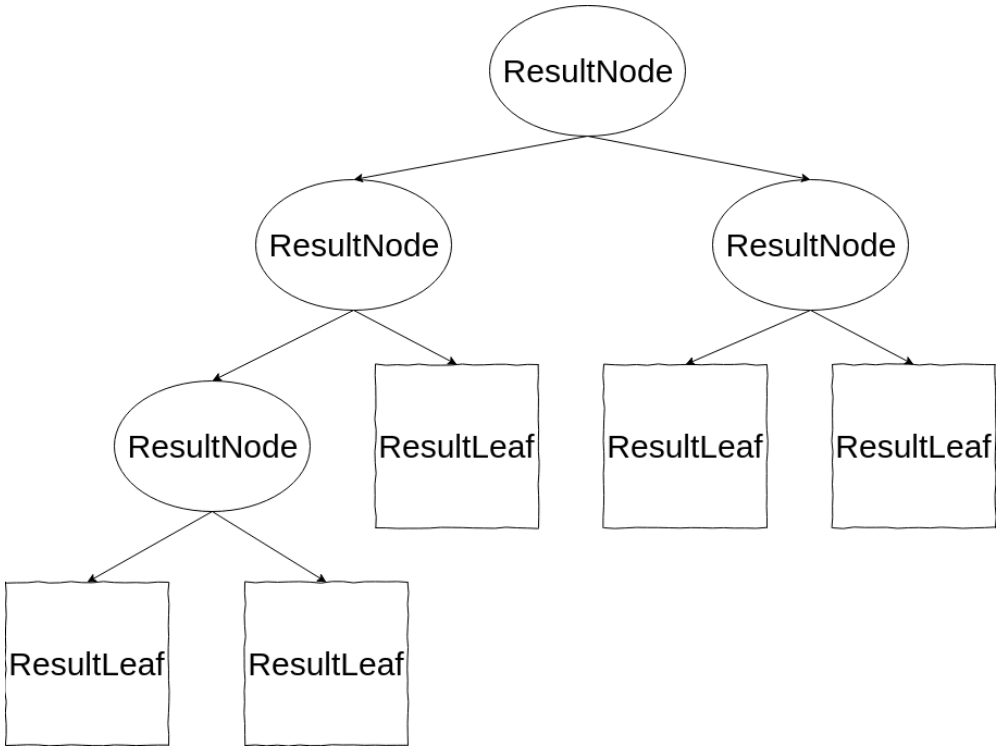

Abbildung 4.16: Ergebnisbaumbeispiel

4.6.3.2 Swagger-Definition der API

Listing 4.12 zeigt die Schnittstelle als Swagger-Definition, die gemäß der Definiton aus Unterunterabschnitt 4.6.3.1 erstellt wurde. Des Weiteren sind die Ergebnistypen noch um das Attribut „compareType“ angereichert, welches anzeigt, ob es sich um das Ergebnis eines Vergleichs zweier Blätter oder Knoten handelt.

```
openapi: 3.0.0
servers:
  - url: https://swagger-compare.haug-dev.de/
info:
  description: Api Documentation of Swagger-Compare
  version: '1.0'
  title: Api Documentation of Swagger-Compare
  license:
    name: MIT
    url: 'https://raw.githubusercontent.com/TorbenHaug/swagger-
       compare/master/LICENSE'
```

```
paths:
  /api/compare:
    post:
      summary: Comparision of two Swagger-Definitions provided by an
          URL
      description: Comparision of two Swagger-Definitions provided
         by an URL
      operationId: compareUsingPOST
      requestBody:
        content:
          application/json:
            schema:
              type: object
              properties:
                urlLeft:
                  description:  Url to the left Swagger-Defintion
                  type: string
                urlRight:
                  description: Url to the right Swagger-Defintion
                  type: string
              required:
                - urlLeft
                - urlRight
        required: true
      responses:
        '200':
          description: OK
          content:
            '*/*':
              schema:
                $ref: '#/components/schemas/NodeCompareResult'
        '400':
          description: Bad Request
          content:
            '*/*':
              schema:
                $ref: '#/components/schemas/ErrorMessage'
        '404':
          description: Not Found
          content:
```

```
49            '*/*':
50              schema:
51                $ref: '#/components/schemas/ErrorMessage'
52  /api/compareFiles:
53    post:
54      summary: Comparision of two Swagger-Definitions provided as
            files
55      description: Comparision of two Swagger-Definitions provided
            as files
56      operationId: compareFilesUsingPOST
57      requestBody:
58        content:
59          multipart/form-data:
60            schema:
61              type: object
62              properties:
63                fileLeft:
64                  description: fileLeft
65                  type: string
66                  format: binary
67                fileRight:
68                  description: fileRight
69                  type: string
70                  format: binary
71              required:
72                - fileLeft
73                - fileRight
74        required: true
75      responses:
76        '200':
77          description: OK
78          content:
79            '*/*':
80              schema:
81                $ref: '#/components/schemas/NodeCompareResult'
82        '400':
83          description: Bad Request
84          content:
85            '*/*':
86              schema:
```

```
87                $ref: '#/components/schemas/ErrorMessage'
88        '404':
89          description: Not Found
90          content:
91            '*/*':
92              schema:
93                $ref: '#/components/schemas/ErrorMessage'
94 components:
95   schemas:
96     NodeCompareResult:
97       type: object
98       properties:
99         values:
100          type: object
101          additionalProperties:
102            oneOf:
103              - $ref: '#/components/schemas/NodeCompareResult'
104              - $ref: '#/components/schemas/LeafCompareResult'
105        compareCriticalType:
106          type: string
107          enum:
108            - CRITICAL
109            - WARNING
110            - INFO
111            - NONE
112        compareResultType:
113          type: string
114          enum:
115            - CREATED
116            - CHANGED
117            - DELETED
118            - UNCHANGED
119        compareType:
120          type: string
121          enum:
122            - NODE
123    LeafCompareResult:
124      type: object
125      properties:
126        valueLeft:
```

```
127           type: string
128         valueRight:
129           type: string
130         compareCriticalType:
131           type: string
132           enum:
133             - CRITICAL
134             - WARNING
135             - INFO
136             - NONE
137         compareResultType:
138           type: string
139           enum:
140             - CREATED
141             - CHANGED
142             - DELETED
143             - UNCHANGED
144         compareType:
145           type: string
146           enum:
147             - LEAF
148     ErrorMessage:
149       type: object
150       properties:
151         message:
152           type: string
153         detailedMessage:
154           type: string
```

Listing 4.12: Swagger-Compare API als Swagger-Definition

4.6.3.3 Umsetzung der API im Backend

Die Umsetzung der in Unterunterabschnitt 4.6.3.2 beschriebenen Schnittstelle erfolgt im Backend über die Module „swagger-compare-datatypes“ (siehe: Absatz 4.6.5.1.1) und „swagger-compare-rest“ (siehe: Absatz 4.5.5.1.5). In Listing 4.13 wird der Rest-Controller des Backends gezeigt, welcher sich im Modul „swagger-compare-rest“ befindet. Dieser stellt die beiden Zugriffspunkte „compare“ und „compareFiles“ bereit. Diese geben jeweils ein Objekt vom Typ „ICompareResult“ zurück. Die möglichen Exceptions werden von dem in Listing 4.14 gezeigten „RestResponseEntityExceptionHandler“ verarbeitet.

```java
package de.haug_dev.swagger_compare_rest;

import de.haug_dev.swagger_compare.swagger_compare_datatypes.ICompareResult;
import de.haug_dev.swagger_compare.swagger_compare_facade.SwaggerCompareFacade;
import de.haug_dev.swagger_compare.swagger_compare_reader.InvalidOpenAPIFileException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.net.MalformedURLException;

@RestController()
@RequestMapping("/api")
public class CompareController {


    private Logger logger = LoggerFactory.getLogger(CompareController.class);

    private SwaggerCompareFacade facade;

    @Autowired
    public CompareController(SwaggerCompareFacade facade) {
        this.facade = facade;
    }

    @PostMapping(value = "/compare")
    public ICompareResult compare(@RequestBody CompareRequest value) throws MalformedURLException, InvalidOpenAPIFileException {
        ICompareResult compareResult = facade.compare(value.urlLeft, value.urlRigth);
        return compareResult;
    }

```

```
35      @PostMapping(value = "/compareFiles")
36      public ICompareResult compareFiles(@RequestParam("fileLeft")
            MultipartFile fileLeft, @RequestParam("fileRight")
            MultipartFile fileRight) throws IOException,
            InvalidOpenAPIFileException {
37          ICompareResult compareResult = facade.compareFiles(new
                String(fileLeft.getBytes()), new String(fileRight.
                getBytes()));
38          return compareResult;
39      }
40
41  }
```

Listing 4.13: Rest-Controller Backend

```
1  package de.haug_dev.swagger_compare_rest;
2
3  import de.haug_dev.swagger_compare.swagger_compare_reader.
       InvalidOpenAPIFileException;
4  import org.apache.commons.lang3.exception.ExceptionUtils;
5  import org.slf4j.Logger;
6  import org.slf4j.LoggerFactory;
7  import org.springframework.http.HttpStatus;
8  import org.springframework.web.bind.annotation.ControllerAdvice;
9  import org.springframework.web.bind.annotation.ExceptionHandler;
10 import org.springframework.web.bind.annotation.ResponseBody;
11 import org.springframework.web.bind.annotation.ResponseStatus;
12 import org.springframework.web.servlet.mvc.method.annotation.
       ResponseEntityExceptionHandler;
13
14 import java.net.MalformedURLException;
15
16 @ControllerAdvice
17 public class RestResponseEntityExceptionHandler extends
       ResponseEntityExceptionHandler {
18
19      Logger logger = LoggerFactory.getLogger(
            RestResponseEntityExceptionHandler.class);
20
21      @ExceptionHandler({
22              InvalidOpenAPIFileException.class
```

```
23      })
24      @ResponseBody
25      @ResponseStatus(HttpStatus.NOT_FOUND)
26      public ErrorMessage notFound(final Exception error) {
27          Throwable rootError = getRootCause(error);
28          logger.debug("Generating error: " + rootError.getMessage());
29          return new ErrorMessage(rootError.getMessage(),
                ExceptionUtils.getStackTrace(rootError));
30      }
31
32      @ExceptionHandler({
33              MalformedURLException.class
34      })
35      @ResponseBody
36      @ResponseStatus(HttpStatus.BAD_REQUEST)
37      public ErrorMessage badRequest(final Exception error) {
38          Throwable rootError = getRootCause(error);
39          logger.debug("Generating error: " + rootError.getMessage());
40          return new ErrorMessage(rootError.getMessage(),
                ExceptionUtils.getStackTrace(rootError));
41      }
42
43      private Throwable getRootCause(Throwable throwable){
44          if(throwable.getCause() == null) {
45              return throwable;
46          }
47          return getRootCause(throwable.getCause());
48      }
49  }
```

Listing 4.14: RestResponseEntityExceptionHandler Backend

Listing 4.15 zeigt das Interface „ICompareResult“ welches die Gemeinsamkeiten zwischen einem Vergleichsergebnis eines Knotens und eines Blattes darstellt. Listing 4.16 zeigt das Interface „INodeCompareResult“, welches das Vergleichsergebnis eines Knoten darstellt und Listing 4.17 zeigt das Interface „ILeafCompareResult“, welches das Vergleichsergebnis eines Blattes darstellt. Diese befinden sich im Modul „swagger-compare-datatypes“.

```
1  package de.haug_dev.swagger_compare.swagger_compare_datatypes;
2
3  public interface ICompareResult {
```

```
4     CompareCriticalType getCompareCriticalType();
5
6     CompareResultType getCompareResultType();
7
8     CompareType getCompareType();
9 }
```

Listing 4.15: ICompareResult Backend

```
1 package de.haug_dev.swagger_compare.swagger_compare_datatypes;
2
3 import java.util.Map;
4
5 public interface INodeCompareResult extends ICompareResult {
6     Map<String, ICompareResult> getValues();
7 }
```

Listing 4.16: INodeCompareResult Backend

```
1 package de.haug_dev.swagger_compare.swagger_compare_datatypes;
2
3 public interface ILeafCompareResult extends ICompareResult {
4     Object getValueLeft();
5
6     Object getValueRight();
7 }
```

Listing 4.17: ILeafCompareResult Backend

4.6.3.4 Zugriff auf die API im Frontend

Listing 4.18 zeigt, wie die Datentypen im Frontend umgesetzt sind. Listing 4.19 und Listing 4.20 zeigen wie vom Frontend aus auf die Schnittstelle zugegriffen wird.

```
1 export interface INodeCompareResult extends ICompareResult {
2   values: { [index: string]: ICompareResult }
3 }
4
5 export interface ILeafCompareResult {
6   valueLeft: any;
7   valueRight: any;
8 }
```

```
9
10 export interface ICompareResult {
11   compareCriticalType: CompareCriticalType;
12   compareResultType: CompareResultType;
13   getCompareType: CompareType;
14 }
15
16 export enum CompareType {
17   NODE="NODE",
18   LEAF="LEAF"
19 }
20
21 export enum CompareResultType {
22   CREATED="CREATED",
23   CHANGED="CHANGED",
24   DELETED="DELETED",
25   UNCHANGED="UNCHANGED"
26 }
27
28 export enum CompareCriticalType {
29   CRITICAL="CRITICAL",
30   WARNING="WARNING",
31   INFO="INFO",
32   NONE="NONE"
33 }
```

Listing 4.18: Datentypen für die Schnittstelle im Frontend

```
1 import {Component, isDevMode, OnInit} from '@angular/core';
2 import {TraceBoxDataService} from "../trace-box/trace-box.data.
    service";
3 import {FlashMessagesService} from "ngx-flash-messages";
4 import {HttpClient, HttpErrorResponse} from "@angular/common/http";
5 import {CompareResultDataService} from "../compare-result/compare-
    result.data.service";
6
7 @Component({
8   selector: 'load-from-file',
9   templateUrl: './load-from-file.component.html',
10  styleUrls: ['./load-from-file.component.scss']
11 })
```

```
12 export class LoadFromFileComponent implements OnInit {
13   private fileLeft: File;
14   private fileRight: File;
15
16   constructor(private http: HttpClient,
17               private flashMessagesService: FlashMessagesService,
18               private traceBoxDataService: TraceBoxDataService,
19               private compareResultDataService:
                     CompareResultDataService) { }
20
21   ngOnInit() {
22   }
23
24   handleFileLeft(files: FileList) {
25     this.fileLeft = files[0];
26   }
27
28   handleFileRight(files: FileList) {
29     this.fileRight = files[0];
30   }
31
32   frmValid() {
33     return this.fileLeft != null && this.fileRight != null;
34   }
35
36   onSubmit(){
37     let formdata: FormData = new FormData();
38     formdata.append('fileLeft', this.fileLeft);
39     formdata.append('fileRight', this.fileRight);
40     this.http.post("/api/compareFiles",formdata).subscribe(
41       (data) => {
42         this.compareResultDataService.showResult(data);
43         if(isDevMode()){
44           this.traceBoxDataService.showTrace(JSON.stringify(data));
45         }else {
46           this.traceBoxDataService.showTrace("");
47         }
48       },
49       (error: HttpErrorResponse) => {
50         this.compareResultDataService.showResult("");
```

```
51          this.traceBoxDataService.showTrace(error);
52
53        }
54      );
55    }
56  }
```

Listing 4.19: REST-Call „compareFiles“ im Frontend

```
1  import {Component, isDevMode, OnInit} from '@angular/core';
2  import {HttpClient, HttpErrorResponse, HttpResponse} from "@angular/
       common/http";
3  import {FlashMessagesService} from "ngx-flash-messages";
4  import {TraceBoxDataService} from "../trace-box/trace-box.data.
       service";
5  import {CompareResultDataService} from "../compare-result/compare-
       result.data.service";
6
7  @Component({
8    selector: 'load-url-form',
9    templateUrl: './load-url-form.component.html',
10   styleUrls: ['./load-url-form.component.scss']
11 })
12 export class LoadUrlFormComponent implements OnInit {
13
14   constructor(
15     private http: HttpClient,
16     private flashMessagesService: FlashMessagesService,
17     private traceBoxDataService: TraceBoxDataService,
18     private compareResultDataService: CompareResultDataService) {
19
20   }
21
22   ngOnInit() {
23   }
24
25   onSubmit(urlLeft: string, urlRight: string){
26     console.log("urlLeft: \"" + urlLeft + "\", " + "urlRight: \"" +
         urlRight)
27     var body = {
28       urlLeft: urlLeft,
```

```
29        urlRight: urlRight
30      }
31      this.http.post("/api/compare",body).subscribe(
32        (data) => {
33          this.compareResultDataService.showResult(data);
34          if(isDevMode()){
35            this.traceBoxDataService.showTrace(JSON.stringify(data));
36          }else {
37            this.traceBoxDataService.showTrace("");
38          }
39        },
40        (error: HttpErrorResponse) => {
41          this.compareResultDataService.showResult("");
42          this.traceBoxDataService.showTrace(error);
43
44        }
45      );
46    }
47  }
```

Listing 4.20: REST-Call „compare“ im Frontend

4.6.3.5 Fazit

Der Client kann sowohl URLs als auch Dateien an den Server zur Auswertung übertragen und die Antworten zur Weiterverarbeitung empfangen. Der Server kann sowohl URLs als auch Dateien zur Verarbeitung empfangen und die Ergebnisse an den Client senden. Zudem ist die Schnittstelle, durch die gewählte Baumstruktur als Ergebnismenge, sehr änderungsunanfällig. Somit konnten alle an die Schnittstelle gestellten Anforderungen erfüllt werden.

4.6.4 Fachlicher Prototype des Clients

In Unterabschnitt 4.6.2 wurden folgende Kriterien an den Client gestellt:

1. Es können zwei Swagger-Definitionen (die sich auf einem HTTP-Server befinden) ausgewählt werden.
2. Es können zwei Swagger-Definitionen (die sich lokal auf dem Client befinden) ausgewählt werden.
3. Die Antwort des Servers wird grafisch aufgearbeitet und dargestellt.

Der Client befindet sich vollständig im Modul „swagger-compare-ui“. Im folgenden wird gezeigt, wie dieser umgesetzt wurde.

4.6.4.1 Client Übersicht

Der Client gliedert sich in folgende Komponenten:

- **home:** Diese Komponente ist der Einstiegspunkt in die UI. Sie wird in Unterunterabschnitt 4.6.4.2 weiter behandelt.
- **load-url-form:** Mit dieser Komponente wird das Laden von Swagger-Definitionen per URL ermöglicht. Sie wird in Unterunterabschnitt 4.6.4.3 weiter behandelt.
- **load-from-file:** Mit dieser Komponente wird das Laden von Swagger-Definitionen aus lokalen Dateien ermöglicht. Sie wird in Unterunterabschnitt 4.6.4.4 weiter behandelt.
- **compare-result:** Diese Komponente stellt den Einstiegspunkt für die Darstellung des Vergleichsergebnisses dar. Sie wird in Unterunterabschnitt 4.6.4.5 weiter behandelt.
- **node-compare-result:** Diese Komponente ist speziell für die Darstellung von Vergleichsergebnissen zweier Knoten zuständig. Sie wird in Unterunterabschnitt 4.6.4.6 weiter behandelt.
- **leaf-compare-result:** Diese Komponente ist speziell für die Darstellung von Vergleichsergebnissen zweier Blätter zuständig. Sie wird in Unterunterabschnitt 4.6.4.7 weiter behandelt.

Abbildung 4.17 zeigt einen Screenshot des Clients. Über die Tabs „URL“ und „Yaml/Json-File“ kann ausgewählt werden, ob die Swagger-Definitionen per URL (siehe: Unterunterabschnitt 4.6.4.3) oder per Dateiupload (siehe: Unterunterabschnitt 4.6.4.4) an den Server übertragen werden. Mittels des Submit-Buttons wird die Übertragung dann angestoßen (siehe: Unterunterabschnitt 4.6.3.4). Unter „Result“ wird das Ergebnis des Vergleichs dargestellt (siehe: Unterunterabschnitt 4.6.4.5). Dabei werden die Knoten hierarchisch gemäß ihrer Position in der Ergebnismenge (siehe: Unterunterabschnitt 4.6.3.1 in Boxen, die auf- und zugeklappt werden können, dargestellt. Ist das Kritikalitätslevel eines Knotens „None“, wird er nicht farblich markiert. Bei „Info“ wird er blau, bei „Warning“ gelb und bei „Critical“ rot eingefärbt (siehe: Unterunterabschnitt 4.6.4.6). Die Blätter werden ebenfalls gemäß ihrer Position in der Ergebnismenge (siehe: Unterunterabschnitt 4.6.3.1 in Boxen, die auf- und zugeklappt werden können, dargestellt und gemäß ihres Kritikalitätslevels eingefärbt. Im Gegensatz zu den Knoten, werden,

wenn diese aufgeklappt werden, keine weiteren Unterknoten angezeigt, sondern der Wert des Blattes auf der linken und rechten Seite (siehe: Unterunterabschnitt 4.6.4.7).

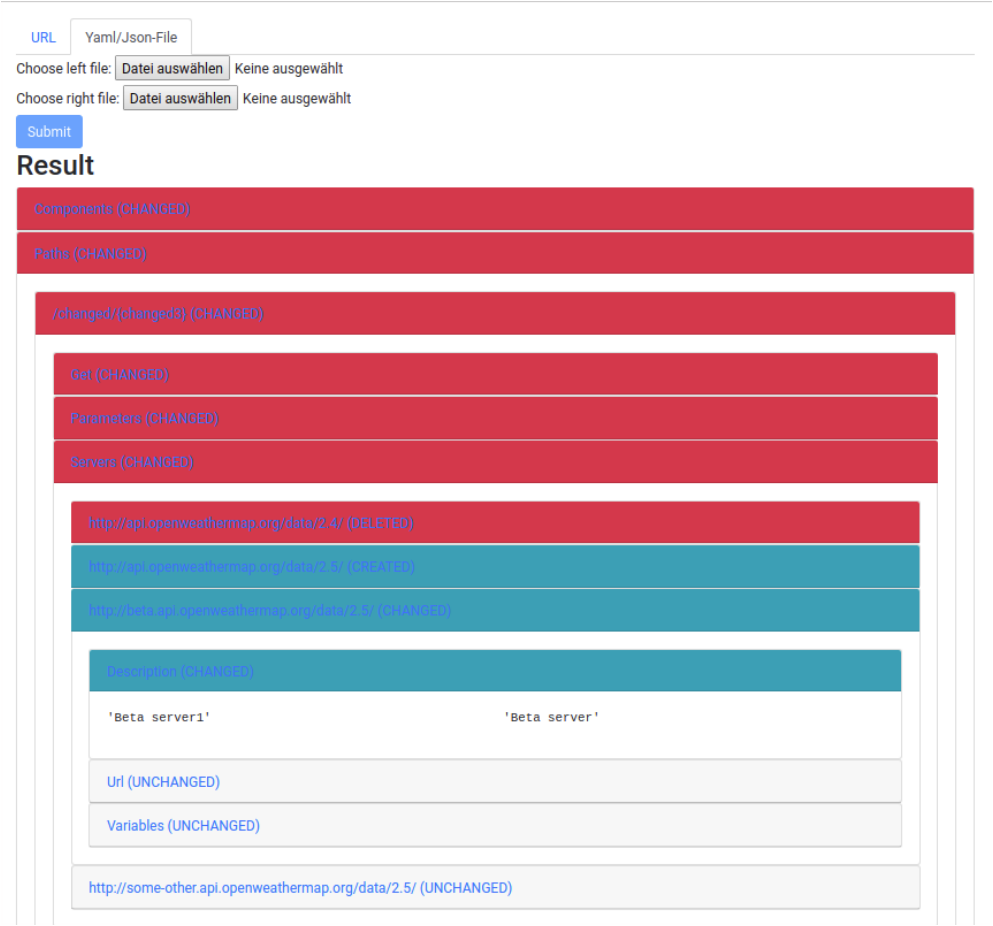

Abbildung 4.17: Screenshot des Clients des fachlichen Prototypen

4.6.4.2 „home“-Komponente

Listing 4.21 zeigt den HTML-Teil der „home“-Komponente. Über ein „ngb-tabset“ werden die Tabs „URL“ und „Yaml/Json-File“ realisiert. In dem Content der Tabs werden die entsprechenden Komponenten „load-url-form“ (siehe: Unterunterabschnitt 4.6.4.3) und „load-from-file“ (siehe: Unterunterabschnitt 4.6.4.4) geladen. Zuletzt wird die Komponente „compare-result“ geladen.

```
1 <div>
2   <ngb-tabset>
3     <ngb-tab title="URL">
```

```
      <ng-template ngbTabContent>
        <load-url-form></load-url-form>
      </ng-template>
    </ngb-tab>
    <ngb-tab title="Yaml/Json-File">
      <ng-template ngbTabContent>
        <load-from-file></load-from-file>
      </ng-template>
    </ngb-tab>
  </ngb-tabset>
</div>
<compare-result></compare-result>
```

Listing 4.21: HTML-Teil der „home“-Komponente

4.6.4.3 „load-url-form“-Komponente

Listing 4.22 zeigt den HTML-Teil der „load-url-form“-Komponente. Hierbei handelt es sich um ein Formular zur Eingabe von zwei Strings. Die eingegebenen Strings werden darauf überprüft, ob es sich um gültige URLs handelt. Wenn zwei gültige URLs eingegeben wurden, wird der Submit-Button aktiviert.

```

<form name="loadUrlForm" id="loadUrlForm" #loadUrlForm="ngForm" (
   ngSubmit)="onSubmit(urlLeft.value,urlRight.value)">
  <div class="form-group">
    <label for="urlLeft">URL-Left</label>
    <input
      required
      pattern="^((http[s]?|ftp):\/)?\/?([^:\/\s]+)((\/\w+)*\/)([\w
          \-\.]+[^#?\s]+)(.*)?(#[\w\-]+)?$"
      ngModel
      name= "urlLeft"
      #urlLeft="ngModel"
      id="urlLeft"
      type="text"
      class="form-control">
    <div
      class="alert alert-danger"
      *ngIf="urlLeft.touched && !urlLeft.valid">
      <div
```

```
18          *ngIf="urlLeft.errors.required">
19          URL-Left is required.
20        </div>
21        <div
22          *ngIf="urlLeft.errors.pattern">
23          Please enter a valid url.
24        </div>
25      </div>
26    </div>
27    <div class="form-group">
28      <label for="urlRight">URL-Right</label>
29      <input
30        required
31        pattern="^((http[s]?|ftp):\/)?\/?([^:\/\s]+)((\/\w+)*\/)([\w
              \-\.]+[^#?\s]+)(.*)?(#[\w\-]+)?$"
32        ngModel
33        name= "urlRight"
34        #urlRight="ngModel"
35        id="urlRight"
36        type="text"
37        class="form-control">
38      <div
39        class="alert alert-danger"
40        *ngIf="urlRight.touched && !urlRight.valid">
41        <div
42          *ngIf="urlRight.errors.required">
43          URL-Right is required.
44        </div>
45        <div
46          *ngIf="urlRight.errors.pattern">
47          Please enter a valid url.
48        </div>
49      </div>
50    </div>
51    <button class="btn btn-primary" [disabled]="!loadUrlForm.form.
        valid">Submit</button>
52  </form>
```

Listing 4.22: HTML-Teil der „load-url-form“-Komponente

4.6.4.4 „load-from-file“-Komponente

Listing 4.23 zeigt den HTML-Teil der „load-from-file“-Komponente. Hierbei handelt es sich um ein Formular zur Eingabe von zwei Dateien. Wenn zwei Dateien eingegeben wurden, wird der Submit-Button aktiviert.

```
<form name="loadFileForm" id="loadFileForm" #loadFileForm="ngForm" (
    ngSubmit)="onSubmit()">
  <div class="form-group-left">
    <label for="fileLeft">Choose left file: </label>
    <input required type="file" id="fileLeft" (change)="
        handleFileLeft($event.target.files)" accept=".json,␣.yaml">
  </div>
  <div class="form-group-right">
    <label for="fileRight">Choose right file: </label>
    <input required type="file" id="fileRight" (change)="
        handleFileRight($event.target.files)" accept=".json,␣.yaml">
  </div>
  <button class="btn␣btn-primary" [disabled]="!loadFileForm.form.
      valid␣||␣!frmValid()">Submit</button>
</form>
```

Listing 4.23: HTML-Teil der „load-from-file“-Komponente

4.6.4.5 „compare-result“-Komponente

Listing 4.24 zeigt den HTML-Teil der „compare-result“-Komponente. Wenn die eingehenden Daten ein Vergleichsergebnis sind, wird dieses an die „node-compare-result“-Komponente (siehe: Unterunterabschnitt 4.6.4.6) weitergegeben, da bei einem Vergleich der Wurzelknoten der Ergebnismenge immer das Ergebnis des Vergleichs der Wurzelknoten der zu vergleichenden Swagger-Definitionen ist. Handelt es sich nicht um ein Vergleichsergebnis, werden die empfangenen Daten zu einem String zusammengefasst und ausgegeben.

```
<h2>Result</h2>


<div *ngIf="isCompareResult">
  <node-compare-result [nodeValue]="result"></node-compare-result>
</div>

```

```
8 <div [innerHTML]="resultText␣|␣stringify" *ngIf="!isCompareResult"
    ></div>
```

Listing 4.24: HTML-Teil der „compare-result“-Komponente

4.6.4.6 „node-compare-result“-Komponente

Listing 4.25 zeigt den HTML-Teil der „node-compare-result“-Komponente. Über ein „ngb-accordion“ werden die Boxen dargestellt. Für jeden Unterknoten und jedes Blatt des Knotens wird ein „ngb-panel“ erstellt, das als Titel den Namen des Knotens trägt und mittels der „type“-funktion entsprechend des Kritikalitätslevels eingefärbt wird. Als Inhalt eines Panels wird entweder erneut die „node-compare-result“-Komponente für den entsprechenden Unterknoten, oder die „leaf-compare-result“-Komponente für das entsprechende Blatt gerendert.

```
1  <div>
2    <ngb-accordion #acc="ngbAccordion">
3      <ngb-panel [title]="key␣+␣'␣('␣+␣nodeValue.values[key].
         compareResultType␣+␣')'" [type]="nodeValue.values[key].
         compareCriticalType␣|␣compareCriticalTypeToBootstrapType" *
         ngFor="let␣key␣of␣nodeValue.values␣|␣keys">
4        <ng-template ngbPanelContent>
5          <div *ngIf="nodeValue.values[key].compareType␣==␣compareType
             .NODE">
6            <node-compare-result [nodeValue]="nodeValue.values[key]"
               ></node-compare-result>
7          </div>
8          <div *ngIf="nodeValue.values[key].compareType␣==␣compareType
             .LEAF">
9            <leaf-compare-result [leaf]="nodeValue.values[key]"></leaf
               -compare-result>
10         </div>
11       </ng-template>
12     </ngb-panel>
13   </ngb-accordion>
14 </div>
```

Listing 4.25: HTML-Teil der „node-compare-result“-Komponente

4.6.4.7 „leaf-compare-result“-Komponente

Listing 4.26 zeigt den HTML-Teil der „leaf-compare-result“-Komponente. Diese Komponente zeigt den linken und rechten Wert des zu vergleichenden Blattes an.

```
1 <div class="leafLeft"><pre>{{leaf.valueLeft | stringify}}</pre></div
     >
2 <div class="leafRight"><pre>{{leaf.valueRight | stringify}}</pre></
     div>
```

Listing 4.26: HTML-Teil der „leaf-compare-result“-Komponente

4.6.4.8 Fazit

Es können zwei Swagger-Definitionen per URL oder Datei zu Vergleich ausgewählt werden und das Vergleichsergebnis kann angezeigt werden.

4.6.5 Fachlicher Prototype des Backends

In Unterabschnitt 4.6.2 wurden folgende Kriterien an das Backend gestellt:

1. Es können zwei URLs zu Swagger-Definitionen oder Dateien per Rest-Schnittstelle empfangen werden.
2. Die URLs oder Dateien werden in POJOs umgewandelt.
3. Die beiden OpenAPI-Dokumente werden, gemäß der in Unterabschnitt 3.1.6 festgelegten Kriterien, verglichen.
4. Das Ergebnis wird per REST-Schnittstelle an den Client übertragen.
5. Die REST-Schnittstelle im wesentlich änderungsunanfällig.

4.6.5.1 Modulübersicht

In Unterunterabschnitt 4.5.5.1 wurde die Modulübersicht für den technischen Prototypen gezeigt. Dieser Abschnitt befasst sich nun mit der Modulübersicht des fachlichen Prototypen. Die eigentlichen Aufgaben der einzelnen Module haben sich nicht geändert, daher wird in diesem Abschnitt lediglich auf die Module eingegangen, bei denen sich die Funktionsweise gravierend geändert hat.

4.6.5.1.1 swagger-compare-datatypes

Abbildung 4.18 zeigt das Klassendiagramm für dieses Modul. Bei den Klassen „LeafCompareResult“ und „NodeCompareResult“ handelt es sich um die Implementierung der für die Ergebnismenge (siehe: Unterabschnitt 4.6.3) benötigten Datentypen. Beim „LeafCompareResult“ werden die Attribute über den Konstruktor übergeben. Listing 4.27 zeigt die Implementierung der Klasse „NodeCompareResult“. Über den Konstruktor werden die Kritikalitätslevel für die Zustände „Gelöscht“ und „Erstellt“ übergeben. Mit der Methode „put“ können dem Knoten weitere Ergebnisse hinzugefügt werde. Wird ein Ergebnis hinzugefügt, wird die Kritikalität des Knotens berechnet.

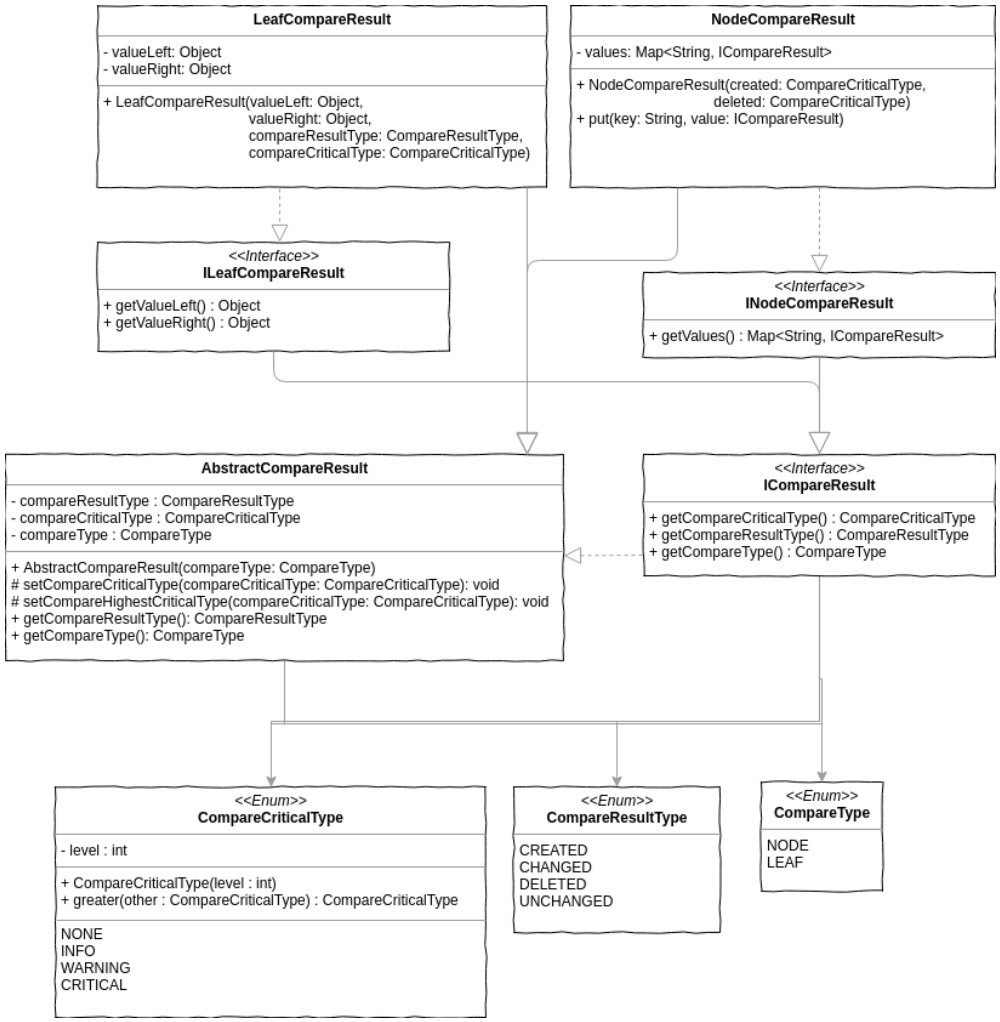

Abbildung 4.18: Klassendiagramm fachlicher Prototype: swagger-compare-datatypes

```
1  package de.haug_dev.swagger_compare.swagger_compare_datatypes;
2
3  import java.util.Map;
4  import java.util.Objects;
5  import java.util.TreeMap;
6
7  public class NodeCompareResult extends AbstractCompareResult
       implements INodeCompareResult {
8
9      private final Map<String, ICompareResult> values;
10     private final CompareCriticalType created;
11     private final CompareCriticalType deleted;
12
13     public NodeCompareResult(CompareCriticalType created,
           CompareCriticalType deleted){
14         super(CompareType.NODE);
15         this.created = created;
16         this.deleted = deleted;
17         this.values = new TreeMap<>();
18     }
19
20     public void put(String key, ICompareResult value){
21         this.values.put(key, value);
22         boolean allResultTypesAreEqual = this.values.
23                 values().
24                 stream().
25                 allMatch((v) ->
26                         v.getCompareResultType().equals(value.
                               getCompareResultType())
27                 );
28         if(allResultTypesAreEqual && (value.getCompareResultType().
               equals(CompareResultType.CREATED) || value.
               getCompareResultType().equals(CompareResultType.DELETED))
               ){
29             setCompareResultType(value.getCompareResultType());
30             if(value.getCompareResultType().equals(CompareResultType
                   .CREATED)){
31                 this.setCompareCriticalType(created);
32             }else if(value.getCompareResultType().equals(
                   CompareResultType.DELETED)){
```

```
33                  this.setCompareCriticalType(deleted);
34              }
35          }else if((allResultTypesAreEqual && value.
                getCompareResultType().equals(CompareResultType.CHANGED)
                || !allResultTypesAreEqual)) {
36              setCompareResultType(CompareResultType.CHANGED);
37              this.setCompareCriticalType(CompareCriticalType.NONE);
38              this.values.values().forEach(v -> {
39                  this.setHighestCompareCriticalType(v.
                        getCompareCriticalType());
40              });
41          }
42      }
43
44      @Override
45      public Map<String, ICompareResult> getValues() {
46          return new TreeMap<>(values);
47      }
48
49      @Override
50      public boolean equals(Object o) {
51          if (this == o) return true;
52          if (!(o instanceof NodeCompareResult)) return false;
53          if (!super.equals(o)) return false;
54          NodeCompareResult that = (NodeCompareResult) o;
55          return Objects.equals(getValues(), that.getValues());
56      }
57
58      @Override
59      public int hashCode() {
60          return Objects.hash(super.hashCode(), getValues());
61      }
62
63      @Override
64      public String toString() {
65          return "NodeCompareResult{" +
66                  "values=" + values +
67                  ",␣compareCriticalType=" + getCompareCriticalType()
                        +
68                  ",␣compareResultType=" + getCompareResultType() +
```

```
69                  ",␣compareType=" + getCompareType() +
70                  '}';
71      }
72  }
```

Listing 4.27: NodeCompareResult Backend

4.6.5.1.2 swagger-compare-core

Abbildung 4.19 zeigt ein vereinfachtes Klassendiagramm für das Modul „swagger-compare-core“. Die „CompareHolder“ dienen zum Vergleich zweier Knoten. Sie erweitern die Klasse „AbstractCompareHolder“, welche eine Default-Implemtierung der Methode „compare“ beinhaltet, sowie einige Hilfsmethoden, die die gängigsten Vergleichsarten abbilden. Die Default-Implementierung der Methode „compare“ geht davon aus, dass die eingehenden zu vergleichenden Objekte Blätter sind und wertet alle Operationen als kritisch. Zudem implementiert die Klasse „AbstractCompareHolder“ das Interface „ICompareHolder“. Die meisten Implementationen der „CompareHolder“ folgen alle dem gleichen gezeigtem Muster. Die Methode „compare“ wird implementiert, um den Vergleich dieses speziellen Knotens abzubilden. Um Unterknoten zu vergleichen, wird dabei der entsprechende CompareHolder für diesen Unterknoten genutzt. Die „CompareHolder“ für die Unterknoten werden in der Methode „compare“ über die „CompareHolderFactory“ aufgelöst. Die Dependency-Injection über die „CompareHolderFactory“ geschieht, um zirkuläre Abhängigkeiten aufzulösen und um sicherzustellen, dass bei jedem Vergleich ein neuer „CompareHolder“ verwendet wird, damit sich der „CompareHolder“ in einem definierten Zustand befindet. Dies ist nötig, da einige „CompareHolder“, wie zum Beispiel der „PathItemCompareHolder“, weitere Informationen vor dem Vergleich übergeben bekommen. Listing 4.28 zeigt, am Beispiel des „OperationCompareHolder“, wie ein „CompareHolder“ implementiert sein kann.

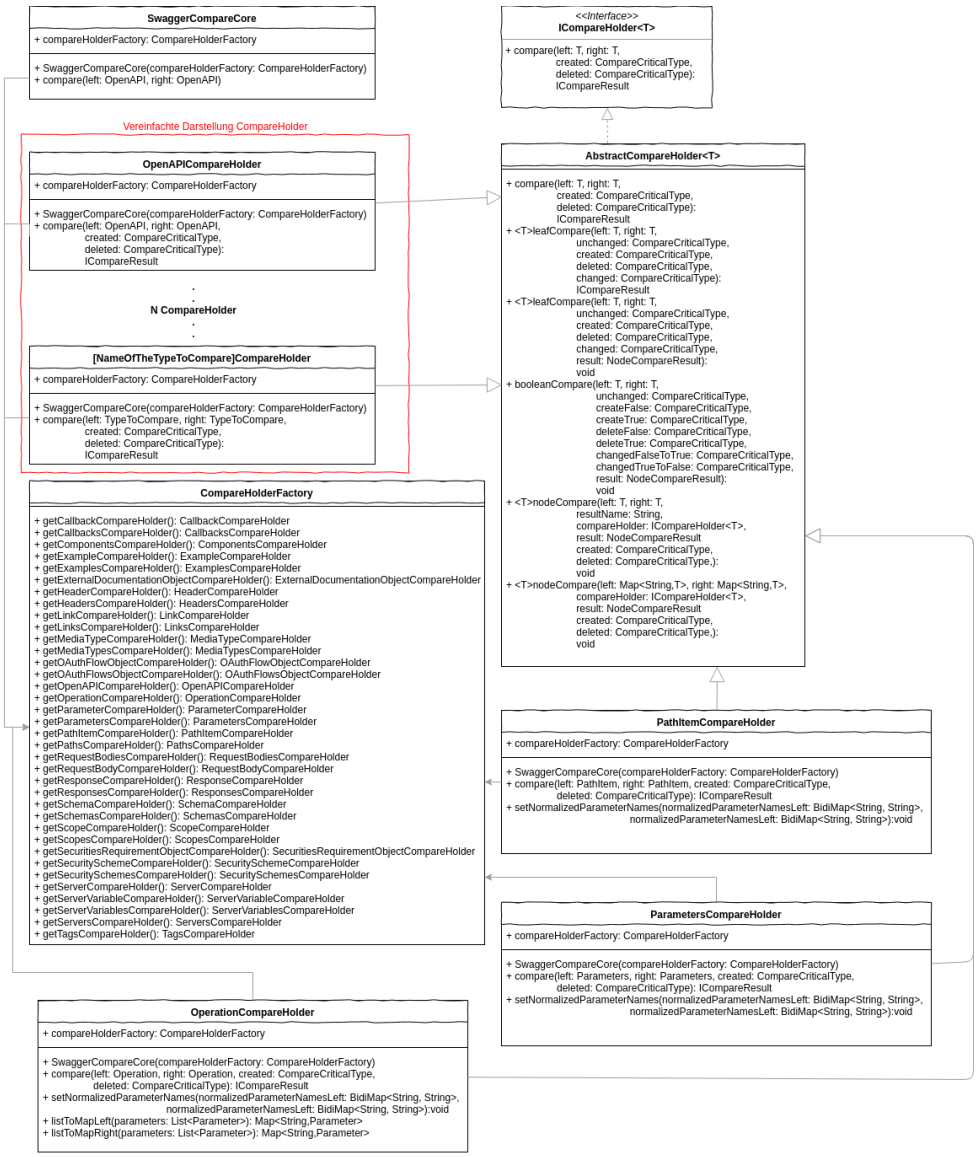

Abbildung 4.19: Klassendiagramm fachlicher Prototype: swagger-compare-core

```
1  package de.haug_dev.swagger_compare.swagger_compare_core.paths;
2
3  import de.haug_dev.swagger_compare.swagger_compare_core.
       AbstractCompareHolder;
4  import de.haug_dev.swagger_compare.swagger_compare_core.
       CompareHolderFactory;
5  import de.haug_dev.swagger_compare.swagger_compare_core.callbacks.
       CallbacksCompareHolder;
6  import de.haug_dev.swagger_compare.swagger_compare_core.misc.
       ExternalDocumentationObjectCompareHolder;
7  import de.haug_dev.swagger_compare.swagger_compare_core.misc.
       TagsCompareHolder;
8  import de.haug_dev.swagger_compare.swagger_compare_core.parameters.
       ParametersCompareHolder;
9  import de.haug_dev.swagger_compare.swagger_compare_core.
       request_bodies.RequestBodyCompareHolder;
10 import de.haug_dev.swagger_compare.swagger_compare_core.responses.
       ResponsesCompareHolder;
11 import de.haug_dev.swagger_compare.swagger_compare_core.
       security_schemes.SecuritiesRequirementObjectCompareHolder;
12 import de.haug_dev.swagger_compare.swagger_compare_core.servers.
       ServersCompareHolder;
13 import de.haug_dev.swagger_compare.swagger_compare_datatypes.*;
14 import io.swagger.v3.oas.models.Operation;
15 import io.swagger.v3.oas.models.parameters.Parameter;
16 import org.apache.commons.collections4.BidiMap;
17 import org.apache.commons.collections4.bidimap.DualHashBidiMap;
18 import org.slf4j.Logger;
19 import org.slf4j.LoggerFactory;
20
21 import java.util.Map;
22 import java.util.Objects;
23
24 import static de.haug_dev.swagger_compare.swagger_compare_datatypes.
       CompareCriticalType.INFO;
25 import static de.haug_dev.swagger_compare.swagger_compare_datatypes.
       CompareCriticalType.NONE;
26 import static de.haug_dev.swagger_compare.swagger_compare_datatypes.
       CompareCriticalType.WARNING;
27
```

```
28 public class OperationCompareHolder extends AbstractCompareHolder<
       Operation> {
29
30     private static Logger LOG = LoggerFactory.getLogger(
           OperationCompareHolder.class);
31
32     private BidiMap<String, String> normalizedParameterNamesLeft =
           new DualHashBidiMap<>();
33     private BidiMap<String, String> normalizedParameterNamesRight =
           new DualHashBidiMap<>();
34     private CompareHolderFactory compareHolderFactory;
35
36     public OperationCompareHolder(CompareHolderFactory
           compareHolderFactory) {
37         this.compareHolderFactory = compareHolderFactory;
38     }
39
40     public void setNormalizedParameterNames(BidiMap<String, String>
           normalizedParameterNamesLeft, BidiMap<String, String>
           normalizedParameterNamesRight) {
41         this.normalizedParameterNamesLeft =
               normalizedParameterNamesLeft == null ? new
               DualHashBidiMap<>() : normalizedParameterNamesLeft;
42         this.normalizedParameterNamesRight =
               normalizedParameterNamesRight == null ? new
               DualHashBidiMap<>() : normalizedParameterNamesRight;
43         LOG.debug("Set parameterNames:");
44         LOG.debug("Left: " + normalizedParameterNamesLeft.toString()
               );
45         LOG.debug("Right: " + normalizedParameterNamesRight.toString
               ());
46     }
47
48     @Override
49     public ICompareResult compare(Operation left, Operation right,
           CompareCriticalType created, CompareCriticalType deleted) {
50         Operation leftValue = left == null ? new Operation() : left;
51         Operation rightValue = right == null ? new Operation() :
               right;
52
```

```
53          TagsCompareHolder tagsCompareHolder = compareHolderFactory.
                getTagsCompareHolder();
54          ExternalDocumentationObjectCompareHolder
                externalDocumentationObjectCompareHolder =
                compareHolderFactory.
                getExternalDocumentationObjectCompareHolder();
55          ParametersCompareHolder parametersCompareHolder =
                compareHolderFactory.getParametersCompareHolder();
56          RequestBodyCompareHolder requestBodyCompareHolder =
                compareHolderFactory.getRequestBodyCompareHolder();
57          ResponsesCompareHolder responsesCompareHolder =
                compareHolderFactory.getResponsesCompareHolder();
58          CallbacksCompareHolder callbacksCompareHolder =
                compareHolderFactory.getCallbacksCompareHolder();
59          ServersCompareHolder serversCompareHolder =
                compareHolderFactory.getServersCompareHolder();
60          SecuritiesRequirementObjectCompareHolder
                securitiesRequirementObjectCompareHolder =
                compareHolderFactory.
                getSecuritiesRequirementObjectCompareHolder();
61
62          NodeCompareResult result = new NodeCompareResult(created,
                deleted);
63          this.nodeCompare(leftValue.getTags(), rightValue.getTags(),
                "Tags", tagsCompareHolder, result, CompareCriticalType.
                INFO, CompareCriticalType.INFO);
64          this.leafCompare(leftValue.getSummary(), rightValue.
                getSummary(), "Summary", CompareCriticalType.NONE,
                CompareCriticalType.INFO, CompareCriticalType.INFO,
                CompareCriticalType.INFO, result);
65          this.leafCompare(leftValue.getDescription(), rightValue.
                getDescription(), "Description", CompareCriticalType.NONE
                , CompareCriticalType.INFO, CompareCriticalType.INFO,
                CompareCriticalType.INFO, result);
66          this.nodeCompare(leftValue.getExternalDocs(), rightValue.
                getExternalDocs(), "ExternalDocs",
                externalDocumentationObjectCompareHolder, result,
                CompareCriticalType.INFO, CompareCriticalType.INFO);
67          this.leafCompare(leftValue.getOperationId(), rightValue.
                getOperationId(), "OperationId", CompareCriticalType.NONE
```

```
            , CompareCriticalType.WARNING, CompareCriticalType.
            WARNING, CompareCriticalType.WARNING, result);
68          parametersCompareHolder.setNormalizedParameterNames(
            normalizedParameterNamesLeft,
            normalizedParameterNamesRight);
69          Map<String, Parameter> parametersLeft =
            parametersCompareHolder.listToMapLeft(leftValue.
            getParameters());
70          Map<String, Parameter> parametersRight =
            parametersCompareHolder.listToMapRight(rightValue.
            getParameters());
71          this.nodeCompare(parametersLeft, parametersRight, "
            Parameters", parametersCompareHolder, result,
            CompareCriticalType.CRITICAL, CompareCriticalType.
            CRITICAL);
72          this.nodeCompare(leftValue.getRequestBody(), rightValue.
            getRequestBody(), "RequestBody", requestBodyCompareHolder
            , result, CompareCriticalType.CRITICAL,
            CompareCriticalType.CRITICAL);
73          this.nodeCompare(leftValue.getResponses(), rightValue.
            getResponses(), "Responses", responsesCompareHolder,
            result, CompareCriticalType.CRITICAL, CompareCriticalType
            .INFO);
74          this.nodeCompare(leftValue.getCallbacks(), rightValue.
            getCallbacks(), "Callbacks", callbacksCompareHolder,
            result, CompareCriticalType.CRITICAL, CompareCriticalType
            .CRITICAL);
75          this.booleanCompare(
76                  leftValue.getDeprecated(),
77                  rightValue.getDeprecated(),
78                  "Deprecated",
79                  NONE,
80                  NONE,
81                  WARNING,
82                  NONE,
83                  INFO,
84                  INFO,
85                  WARNING,
86                  result);
```

```
87          this.nodeCompare(leftValue.getSecurity(), rightValue.
                getSecurity(), "Security",
                securitiesRequirementObjectCompareHolder, result,
                CompareCriticalType.CRITICAL, CompareCriticalType.
                CRITICAL);
88          this.nodeCompare(leftValue.getServers(), rightValue.
                getServers(), "Servers", serversCompareHolder, result,
                CompareCriticalType.INFO, CompareCriticalType.CRITICAL);
89          return result;
90      }
91  }
```

Listing 4.28: OperationCompareHolder Backend

4.6.5.1.3 swagger-compare-rest-cdt

Im technischen Prototypen wurde über den CDT noch ein fast vollständiger Integrationstest durchgeführt (siehe: Absatz 4.5.5.1.6). Aufgrund der in Unterunterabschnitt 4.6.3.1 festgelegten Ergebnismenge ist dies nicht mehr sinnvoll, da es sich bei dem Ergebnis um eine dynamische Struktur handelt. Zudem sind alle Klassen intern durch Unittests getestet. Daher wird durch den CDT lediglich die Struktur des zurückgegebenen Ergebnisses getestet. Listing 4.29 zeigt, wie der Test aufgebaut wird und Listing 4.29 zeigt, wie darauf basierend die Struktur überprüft wird.

```
1  package de.haug_dev.swagger_compare_rest_cdt;
2
3  import de.haug_dev.swagger_compare.swagger_compare_core.
       SwaggerCompareCore;
4  import de.haug_dev.swagger_compare.swagger_compare_datatypes.
       CompareCriticalType;
5  import de.haug_dev.swagger_compare.swagger_compare_datatypes.
       CompareResultType;
6  import de.haug_dev.swagger_compare.swagger_compare_datatypes.
       LeafCompareResult;
7  import de.haug_dev.swagger_compare.swagger_compare_datatypes.
       NodeCompareResult;
8  import de.haug_dev.swagger_compare.swagger_compare_facade.
       SwaggerCompareFacade;
9  import de.haug_dev.swagger_compare.swagger_compare_reader.
       OpenAPIV3ParserFactory;
```

```
10 import de.haug_dev.swagger_compare.swagger_compare_reader.
       SwaggerCompareReader;
11 import de.haug_dev.swagger_compare_rest.CompareController;
12 import io.restassured.module.mockmvc.RestAssuredMockMvc;
13 import io.swagger.v3.oas.models.OpenAPI;
14 import io.swagger.v3.parser.OpenAPIV3Parser;
15 import org.junit.Before;
16 import org.mockito.Mockito;
17
18 import java.net.MalformedURLException;
19 import java.net.URISyntaxException;
20
21 import static org.mockito.Mockito.*;
22
23 public class MvcTest {
24
25     @Before
26     public void setup() {
27
28         NodeCompareResult goodValue0_0 = new NodeCompareResult(
               CompareCriticalType.INFO, CompareCriticalType.CRITICAL);
29         NodeCompareResult goodValue1_0 = new NodeCompareResult(
               CompareCriticalType.INFO, CompareCriticalType.CRITICAL);
30         LeafCompareResult goodValue1_1 = new LeafCompareResult("
               goodValue1_1_left", "goodValue1_1_right",
               CompareResultType.CHANGED, CompareCriticalType.INFO);
31         LeafCompareResult goodValue2_0 = new LeafCompareResult("
               goodValue2_0_left", "goodValue2_0_right",
               CompareResultType.CHANGED, CompareCriticalType.INFO);
32         goodValue0_0.put("goodValue1_0", goodValue1_0);
33         goodValue0_0.put("goodValue1_1", goodValue1_1);
34         goodValue1_0.put("goodValue2_0", goodValue2_0);
35         OpenAPI goodValueLeft = Mockito.mock(OpenAPI.class);
36         OpenAPI goodValueRight = Mockito.mock(OpenAPI.class);
37
38         OpenAPIV3Parser openAPIV3Parser = Mockito.mock(
               OpenAPIV3Parser.class);
39         Mockito.when(openAPIV3Parser.read("http://goodValue/left.
               yaml")).thenReturn(goodValueLeft);
```

```
40          Mockito.when(openAPIV3Parser.read("http://goodValue/right.
                yaml")).thenReturn(goodValueRight);
41
42          OpenAPIV3ParserFactory openAPIV3ParserFactory = mock(
                OpenAPIV3ParserFactory.class);
43          Mockito.when(openAPIV3ParserFactory.getParser()).thenReturn(
                openAPIV3Parser);
44
45          SwaggerCompareCore swaggerCompareCore = Mockito.mock(
                SwaggerCompareCore.class);
46          Mockito.when(swaggerCompareCore.compare(goodValueLeft,
                goodValueRight)).thenReturn(goodValue0_0);
47          RestAssuredMockMvc.standaloneSetup(
48                  new CompareController(
49                          new SwaggerCompareFacade(
50                                  new SwaggerCompareReader(
                                        openAPIV3ParserFactory),
51                                  swaggerCompareCore
52                          )
53                  )
54          );
55      }
56
57  }
```

Listing 4.29: MvcTest Backend

```
1  package contracts
2
3  org.springframework.cloud.contract.spec.Contract.make {
4      request {
5          method 'POST'
6          url '/api/compare'
7          body("""
8      {
9        "urlLeft":"http://goodValue/left.yaml",
10       "urlRight":"http://goodValue/right.yaml"
11     }
12     """)
13         headers {
14             header('Content-Type', 'application/json')
```

```
        }
    }
    response {
        status 200
        body(
            """{
                "compareResultType":"CHANGED",
                "compareCriticalType":"INFO",
                "compareType":"NODE",
                "values":{
                    "goodValue1_0":{
                        "compareResultType":"CHANGED",
                        "compareCriticalType":"INFO",
                        "compareType":"NODE",
                        "values":{
                            "goodValue2_0":{
                                "compareResultType":"CHANGED",
                                "compareCriticalType":"INFO",
                                "compareType":"LEAF",
                                "valueLeft":"goodValue2_0_left",
                                "valueRight":"goodValue2_0_right"
                            }
                        }
                    },
                    "goodValue1_1":{
                        "compareResultType":"CHANGED",
                        "compareCriticalType":"INFO",
                        "compareType":"LEAF",
                        "valueLeft":"goodValue1_1_left",
                        "valueRight":"goodValue1_1_right"
                    }
                }
            }"""
        )
        headers {
            contentType(applicationJson())
        }
    }
```

```
53 }
```

Listing 4.30: CDT Backend

4.6.5.2 Fazit

Über die „CompareHolder“ konnten alle in Unterabschnitt 3.1.6 zum Vergleich ausgewählten Elemente der OpenAPI-Spezifikation verglichen werden und auf ihre Abwärtskompatibilität hin geprüft werden. Eine Ausnahme hiervon bildet das Hinzufügen eines „Parameter“-Knoten, da seine Kritikalität beim Hinzufügen von dem Attribut „required“ abhängig ist (siehe: Unterunterabschnitt 3.1.6.6 und Unterunterabschnitt 3.1.6.9). Die gewählte Architektur deckt diesen speziellen Fall zur Zeit nicht ab. Zudem konnte über die gewählte Ergebnismenge (siehe: Unterunterabschnitt 4.6.3.1) eine im wesentlich änderungsunanfällige REST-Schnittstelle, die sowohl zwei URLs als auch zwei Dateien als Parameter akzeptiert, diese zum Vergleich weiterleitet und das Ergebnis an den Client zurücksendet, erstellt werden.

5 Fazit

5.1 Evaluation des fachlichen Prototypen

In Unterabschnitt 3.2.1 wurden die Kriterien aufgestellt, nach denen bestehende Difftools auf ihre Verwendbarkeit hin geprüft wurde. In diesem Abschnitt wird nun der fachliche Prototyp gegen die gleichen Kriterien geprüft.

5.1.1 Unterstützung von Definitionen gemäß OpenAPI-Spezifikation Version 3.0.0

Der fachliche Prototyp unterstützt Swagger-Definitionen gemäß der OpenAPI-Spezifikation Version 3.0.0.

5.1.2 Erkennung von nicht geänderten Spezifikationen bei einer Reihenfolgeveränderungen

Abbildung 5.1 zeigt einen Screenshot von dem fachlichen Prototypen nach dem Vergleich der Swagger-Definitionen Listing 3.10 und Listing 3.11. Daraus wird ersichtlich, dass die beiden Definitionen korrekt als gleich erkannt wurden.

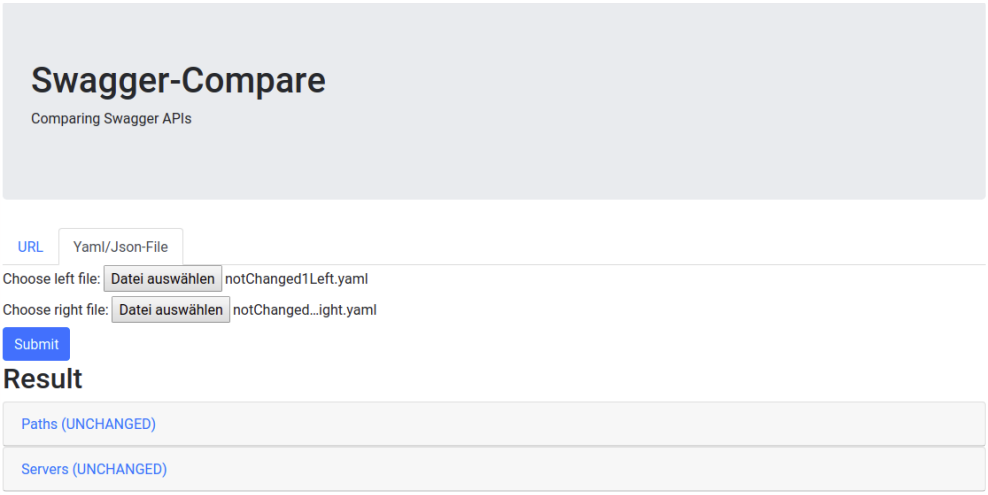

Abbildung 5.1: Screenshot des fachlichen Prototypen nach der Auswertung der Swagger-Definitionen Listing 3.10 und Listing 3.11

5.1.3 Erkennung von nicht geänderten Spezifikationen beim Hinzufügen eines Attributes mit Default-Wert

Abbildung 5.2 zeigt einen Screenshot von dem fachlichen Prototypen nach dem Vergleich der Swagger-Definitionen Listing 3.12 und Listing 3.13. Daraus wird ersichtlich, dass die beiden Definitionen korrekt als gleich erkannt wurden.

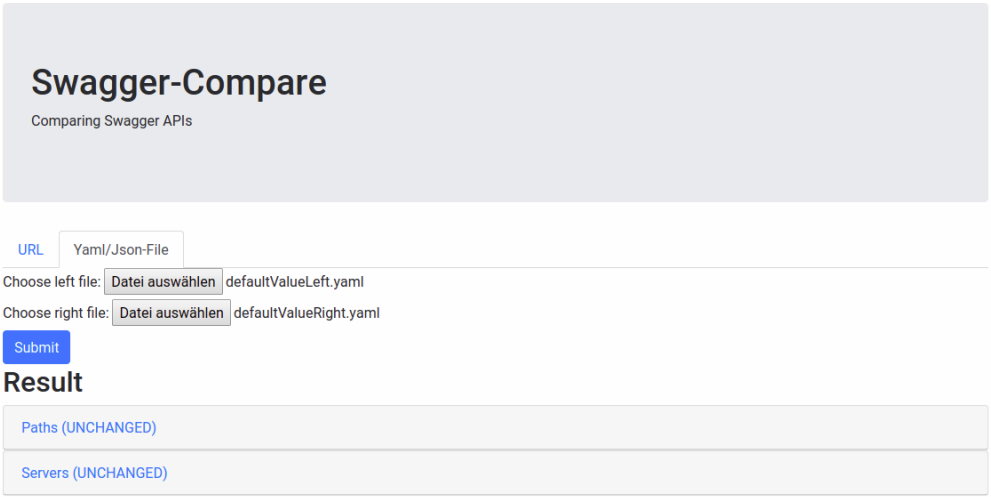

Abbildung 5.2: Screenshot des fachlichen Prototypen nach der Auswertung der Swagger-Definitionen Listing 3.12 und Listing 3.13

5.1.4 Erkennung von geänderten Spezifikationen beim Hinzufügen eines Attributes, wodurch eine kritische API-Änderung eintritt

Abbildung 5.3 zeigt einen Screenshot von dem fachlichen Prototypen nach dem Vergleich der Swagger-Definitionen Listing 3.14 und Listing 3.15. Daraus wird ersichtlich, dass die beiden Definitionen korrekt als geändert erkannt wurden.

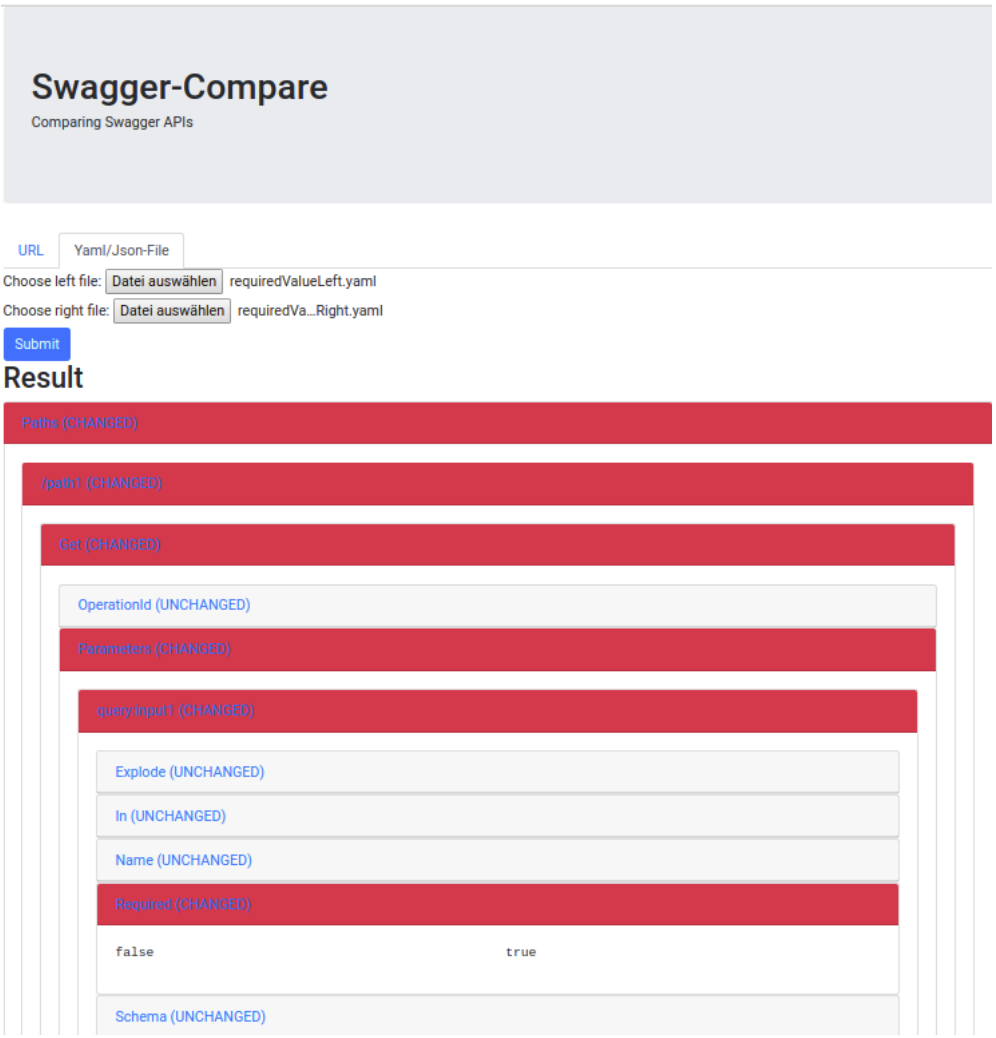

Abbildung 5.3: Screenshot des fachlichen Prototypen nach der Auswertung der Swagger-Definitionen Listing 3.14 und Listing 3.15

5.2 Andere Arbeiten

Im Verlauf dieser Arbeit ist aufgefallen, dass kurz vor dem Beginn dieser Arbeit ein weiteres Vergleichstool veröffentlicht wurde, das ebenfalls den Vergleich zweier Swagger-Definitionen gemäß der OpenAPI Spezifikation Version 3.0.0 unterstützt (vgl.: Desramé (2018)). In Unterabschnitt 5.2.1 wird gezeigt, dass dieses Tool ebenfalls alle in Unterabschnitt 3.2.1 aufgestellten Kriterien erfüllt und in Unterabschnitt 5.2.2 wird auf die Unterschiede zu dieser Arbeit eingegangen.

5.2.1 Evaluation des Programms „OpenAPI-diff“ von Quentin Desramé

In Unterabschnitt 3.2.1 wurden die Kriterien aufgestellt, nach denen bestehende Difftools auf ihre Verwendbarkeit hin geprüft wurden. In diesem Abschnitt wird nun „OpenAPI-diff“ gegen die gleichen Kriterien geprüft.

5.2.1.1 Unterstützung von Definitionen gemäß OpenAPI-Spezifikation Version 3.0.0

Der fachliche Prototyp unterstützt Swagger-Definitionen gemäß der OpenAPI-Spezifikation Version 3.0.0.

5.2.1.2 Erkennung von nicht geänderten Spezifikationen bei einer Reihenfolgeveränderungen

Listing 5.1 zeigt die Ausgabe von „OpenAPI-diff“ nach dem Vergleich der Swagger-Definitionen Listing 3.10 und Listing 3.11. Daraus wird ersichtlich, dass die beiden Definitionen korrekt als gleich erkannt wurden.

```
1 No differences. Specifications are equivalents
```

Listing 5.1: Output von „OpenAPI-diff“ nach der Auswertung der Swagger-Definitionen Listing 3.10 und Listing 3.11

5.2.1.3 Erkennung von nicht geänderten Spezifikationen beim Hinzufügen eines Attributes mit Default-Wert

Listing 5.2 zeigt die Ausgabe von „OpenAPI-diff“ nach dem Vergleich der Swagger-Definitionen Listing 3.12 und Listing 3.13. Daraus wird ersichtlich, dass die beiden Definitionen korrekt als gleich erkannt wurden.

```
No differences. Specifications are equivalents
```

Listing 5.2: Output von „OpenAPI-diff“ nach der Auswertung der Swagger-Definitionen Listing 3.12 und Listing 3.13

5.2.1.4 Erkennung von geänderten Spezifikationen beim Hinzufügen eines Attributes, wodurch eine kritische API-Änderung eintritt

Listing 5.3 zeigt die Ausgabe von „OpenAPI-diff“ nach dem Vergleich der Swagger-Definitionen Listing 3.14 und Listing 3.15. Daraus wird ersichtlich, dass die beiden Definitionen korrekt als geändert erkannt wurden.

```
============================================================================
==                           API CHANGE LOG                               ==
============================================================================
                                 TestAPI
----------------------------------------------------------------------------
--                           What's Changed                               --
----------------------------------------------------------------------------
- GET    /path1
  Parameter:
    - Changed input1 in query
----------------------------------------------------------------------------
--                              Result                                    --
----------------------------------------------------------------------------
               API changes broke backward compatibility
----------------------------------------------------------------------------
```

Listing 5.3: Output von „OpenAPI-diff“ nach der Auswertung der Swagger-Definitionen Listing 3.14 und Listing 3.15

5.2.2 Unterschiede zu dieser Arbeit

Beide Programme erfüllen die gleiche Funktion, weisen aber in ihrer Implementierung grundlegende Unterschiede auf. Die zwei wahrscheinlich gravierendsten Unterschiede sind zum einen die Ergebnismenge und zum anderen die Verarbeitung der „Components Objects“. In dieser Arbeit wurde versucht, eine möglichst abstrakte Ergebnismenge in Form einer Baumstruktur zu erreichen (siehe: Unterunterabschnitt 4.6.3.1), während in „OpenAPI-diff“ für jeden zu vergleichenden Knoten ein eigener Ergebnistyp erstellt wurde (vgl.: Desramé (2018)). In „OpenAPI-diff“ werden Referenzen zu „Components Objects“ aufgelöst und an der Stelle des

Baums verarbeitet, wo sie sich befinden (vgl.: Desramé (2018)), während in dieser Arbeit die „Components Objects“ als Teil des OpenAPI-Baums und die Referenzen hierauf als Blätter betrachtet werden (siehe Abschnitt 3.1).

5.3 Ergebnis der Arbeit

Durch diese Arbeit als auch durch das Programm „OpenAPI-diff“ (siehe: Abschnitt 5.2) wird gezeigt, dass das maschinelle Vergleichen und die Prüfung auf die Abwärtskompatibilität zweier Swagger-Definitionen, die gemäß der OpenAPI Spezifikation Version 3.0.0 erstellt wurden, möglich ist.

In Kapitel 3 wurde ermittelt, das eine Swagger-Definition gemäß OpenAPI-Spezifikation Version 3.0.0 baumartig aufgebaut ist und jeder Knoten und jedes Blatt dieses Baums spezifiziert ist. Außerdem wurden die Operationen untersucht, die auf solch einen Baum durchgeführt werden können. Diese sind das Löschen, Erstellen und Ändern von Kindern und Blättern eines Knotens. Dabei wurde auch untersucht, welche Auswirkungen diese Operationen haben können, wobei von dem Blickwinkel eines bestehenden Consumers und eines geänderten Producers ausgegangen wurde. Dabei wurde ermittelt, dass die Kritikalität, die eine Operation auf ein Blatt hat, davon abhängig ist, welche Auswirkungen diese Operation auf den Elternknoten des Blattes hat. Zudem wurde gezeigt, dass die Kritikalität eines Knotens beim Hinzufügen oder Löschen dieses Knotens abhängig von den Auswirkungen auf seinen Elternknoten ist und dass, wenn ein Knoten verändert wird, die Elternknoten die Kritikalität von ihren Kindern übernehmen.

In Kapitel 4 wurde zunächst ein technischer Prototyp erstellt, mit dem gezeigt wurde, dass ein maschinelles Vergleichen von zwei Swagger-Definitionen prinzipiell möglich ist. Danach wurde ein fachlicher Prototyp erstellt, mit dem gezeigt wurde, dass die in Kapitel 3 ermittelten vorgaben für den Vergleich zweier Swagger-Definitionen und die anschließende strukturierte Visualisierung möglich ist. Die dabei gewählte Architektur kann dabei alle in Unterabschnitt 3.1.6 ermittelten Kriterien abbilden, bis auf einen Sonderfall. Beim Hinzufügen von „Parameter Object“ ist die Auswirkung auf den Knoten abhängig von dem Blatt „required“ von dem „Parameter Object“.

5.4 Ausblick

In Unterabschnitt 3.1.6 wurde die Korrektheit der Zuordnung der Kritikalitätslevel angenommen, aber nicht überprüft. Eine Überprüfung dieser auf Basis verschiedener REST-Frameworks wäre sinnvoll. Zudem wurde in dieser Arbeit davon ausgegangen, dass es sich bei den Attributen „$ref“ um Blätter handelt, während in „OpenAPI-diff“ diese auf Basis der „components“ aufgelößt werden. Es sollte eine Betrachtung durchgeführt werden, welche von beiden Lösungen sinnvoller ist, oder ob es noch weitere Möglichkeiten gibt. Außerdem wurden in dieser Arbeit alle Blätter und Knoten einzeln betrachtet, allerdings können zum Beispiel die Blätter und Knoten „style“, „explode“, „allowReserved“, „schema“, „example“ und „examples“ von einem „Parameter Object“ möglicherweise zu einen „Media Type Object“ zusammengeführt werden. Daraus ergibt sich, dass gegebenenfalls einige der zurzeit erkannten Änderungen False-Positives sind. Eine Überprüfung, ob dies zutrifft und ob es noch weitere Elemente dieser Art gibt, erscheint sinnvoll.

Literaturverzeichnis

[Amaral u. a. 2015] Amaral, M. ; Polo, J. ; Carrera, D. ; Mohomed, I. ; Unuvar, M. ; Steinder, M.: Performance Evaluation of Microservices Architectures Using Containers. In: *2015 IEEE 14th International Symposium on Network Computing and Applications*, 2015, S. 27–34

[Apache Software Foundation 2018] Apache Software Foundation: *What is Maven*. https://maven.apache.org/what-is-maven.html. Version: 2018, Abruf: 2018-02-18

[Barosan und Shah 2018] Barosan, Vlad ; Shah, Vishrut: *README*. https://github.com/Azure/openapi-diff/blob/1d465ba498a0604ff768b7f00ed060bd9665abdf/README.md. Version: 2018, Abruf: 2018-06-15

[Ben-Kiki u. a. 2009] Ben-Kiki, Oren ; Evans, Clark ; Net, Ingy döt: *YAML Ain't Markup Language (YAML™) Version 1.2*. http://yaml.org/spec/1.2/spec.pdf. Version: 2009, Abruf: 2018-07-21

[Cousens 2018] Cousens, Jeff: *README*. https://github.com/civisanalytics/swagger-diff/blob/b0be8416357a019d5ca0fcafe415b57cbca0ff73/README.md. Version: 2018, Abruf: 2018-06-15

[Culbertson u. a. 2018] Culbertson, Ryan ; Xia, David ; Brown, Matt ; Norberg, Daniel ; Flemström, David ; Christensen, Mads M. ; Froeder, Marvin ; lrolaz ; gidzzz ; Georgeson, Justin ; Thiebault, Cedric ; Gimåker, Staffan ; Patterson, Andrew ; Schulze, Marc C. ; Gelev, Emil ; Harringa, Justin ; Malinowski, Dawid ; Bedemann, R. ; craykg ; Stettler, Christian ; Poznachowski, Grzegorz ; Weis, Simon ; Wojdyla, Rafal ; Lundberg, Olle ; Gärtner, Ronny ; Green, James ; Wirooks, Andreas ; Blomqvist, Christer ; Hersevoort, Christian ; ront ; Nicoletti, Umberto: *docker-maven-plugin*. https://github.com/spotify/docker-maven-plugin. Version: 2018, Abruf: 2018-06-15

[Desramé 2018] Desramé, Quentin: *openapi-diff*. https://github.com/quen2404/openapi-diff. Version: 2018, Abruf: 2018-06-14

[Docker Inc. 2018] Docker Inc.: *What Docker*. https://www.docker.com/what-docker. Version: 2018, Abruf: 2018-02-18

[Edo 2018a] Edo, Mauri: *README*. https://bitbucket.org/atlassian/openapi-diff/src/6be86b5bbed73a8f7945c81f96561ff4e7ea3207/README.md. Version: 2018, Abruf: 2018-06-15

[Edo 2018b] Edo, Mauri: *Swagger 2 / OpenAPI 3 feature support*. https://bitbucket.org/atlassian/openapi-diff/src/6be86b5bbed73a8f7945c81f96561ff4e7ea3207/SPEC_SUPPORT.md. Version: 2018, Abruf: 2018-06-15

[Faber 2018] Faber, Szczepan: *Mockito*. http://site.mockito.org/. Version: 2018, Abruf: 2018-01-14

[Fielding u. a. 1999] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Masinter, L. ; Leach, P. ; Berners-Lee, T.: *RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*. https://tools.ietf.org/html/rfc2616. Version: 1999

[Fielding 2000] Fielding, Roy T.: *REST: Architectural Styles and the Design of Network-based Software Architectures*, University of California, Irvine, Doctoral dissertation, 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[Fortin und Sevilla 2018] Fortin, Nicolas ; Sevilla, Devin: *README*. https://github.com/zallek/swagger-diff/blob/b6d62d47330d022c22ab83a92385974293096e47/README.md. Version: 2018, Abruf: 2018-06-15

[Jenkins 2018] Jenkins: *Jenkins User Documentation*. https://jenkins.io/doc/. Version: 2018, Abruf: 2018-02-18

[Jensen und Krochmalski 2018] Jensen, Allan H. ; Krochmalski, Jarek: *README*. https://github.com/AllanHoejgaardJensen/open-api-diff/blob/9613b967a63f991dc5fb8c1d6b1bef7359315859/README.md. Version: 2018, Abruf: 2018-06-15

[JSon.org 2018] JSon.org: *Einführung in JSON*. https://www.json.org/json-de.html. Version: 2018, Abruf: 2018-07-21

[Messinger 2018a] Messinger, James: *Swagger-Parser*. https://github.com/BigstickCarpet/swagger-parser. Version: 2018, Abruf: 2018-06-14

[Messinger 2018b] Messinger, James: *Swagger-Parser History*. https://github.com/BigstickCarpet/swagger-parser/commits/master. Version: 2018, Abruf: 2018-06-14

[Miller und Ron 2017] Miller, Darrel ; Ron, Web: *OpenAPI Specification – Version 3.0.0*. https://github.com/OAI/OpenAPI-Specification/blob/3.0.0/versions/3.0.0.md. Version: 2017, Abruf: 2018-01-14

[Moser und Menke 2018] Moser, Derrick ; Menke, Henri: *Diffuse*. http://diffuse.sourceforge.net/. Version: 2018, Abruf: 2018-02-18

[Newman und Lorenzen 2015] Newman, Sam ; Lorenzen, Knut: *Microservices: Konzeption und Design*. Wachtendonk : MITP, 2015. – ISBN 978–3–95845–081–3

[OpenAPI Initiative 2018] OpenAPI Initiative: *About*. https://www.openapis.org/about. Version: 2018, Abruf: 2018-06-15

[Pivotal Software, Inc. 2018] Pivotal Software, Inc.: *Spring Cloud Contract*. https://cloud.spring.io/spring-cloud-contract/. Version: 2018, Abruf: 2018-01-14

[Sayi 2018] Sayi: *README*. https://github.com/Sayi/swagger-diff/blob/456846e041415e2705da91fa91a4f2ac581baa24/README.md. Version: 2018, Abruf: 2018-06-15

[Sletteberg 2018] Sletteberg, Eirik: *frontend-maven-plugin*. https://github.com/eirslett/frontend-maven-plugin. Version: 2018, Abruf: 2018-02-18

[SmartBear Software 2018a] SmartBear Software: *About Swagger*. https://swagger.io/about/. Version: 2018, Abruf: 2018-06-14

[SmartBear Software 2018b] SmartBear Software: *About SwaggerHub*. https://app.swaggerhub.com/help/index. Version: 2018, Abruf: 2018-06-15

[SmartBear Software 2018c] SmartBear Software: *Compare and Merge API Specs*. https://app.swaggerhub.com/help/apis/compare-and-merge. Version: 2018, Abruf: 2018-06-15

[SmartBear Software 2018d] SmartBear Software: *OpenAPI Specification and Swagger*. https://swagger.io/solutions/getting-started-with-oas/. Version: 2018, Abruf: 2018-06-15

[SmartBear Software 2018e] SmartBear Software: *Swagger Open Source*. https://swagger.io/tools/open-source/. Version: 2018, Abruf: 2018-06-14

[Tam u. a. 2018] Tam, Tony ; webron, Ron ; Tumanischvili, Francesco ; Diaz, Grace Karina G. ; Richards, Andre ; Cheng, William ; Ducin, Tomasz ; Rylander, Stephen ; Chazalet, Boris ; Lensmar, Ole ; Brandon, Lorinda: *Swagger-Parser Readme*. https://github.com/swagger-api/swagger-parser/blob/0c8b3ae4b9a41fe4c0d8e55417a63180add0450b/README.md. Version: 2018, Abruf: 2018-06-14

[Tilkov 2015] Tilkov, Stefan: *REST und HTTP - Entwicklung und Integration nach dem Architekturstil des Web*. Heidelberg : dpunkt, 2015. – ISBN 978–3–864–90120–1

[Ullenboom 2014] Ullenboom, Christian: *Java ist auch eine Insel : Einführung, Ausbildung, Praxis ; [Programmieren mit der Java Plattform, Standard Edition 8 ; Java von A bis Z: Einführung, Praxis, Referenz ; von Klassen und Objekten zu Datenstrukturen und Algorithmen ; aktuell zu Java 8*. Bonn : Galileo Press, 2014. – ISBN 978–3–8362–2873–2

[Walls 2016] Walls, Craig: *Spring Boot in Action*. Shelter Island : Manning Publications, 2016. – ISBN 1617292540

[Woiwode u. a. 2017] Woiwode, Gregor ; Malcher, Ferdinand ; Koppenhagen, Danny ; Hoppe, Johannes: *Angular Grundlagen, fortgeschrittene Techniken und Best Practices mit TypeScript: ab Angular 4, inklusive NativeScript und Redux*. Heidelberg : dpunkt.verlag, 2017. – ISBN 3864903572

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 27. Juli 2018 Claus Torben Haug