FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Entwicklung einer nutzergestützten Matching-Software zum Datenaustausch zwischen APIs

B A C H E L O R A R B E I T

zur Erlangung des akademischen Grades

Bachelor of Science (B. Sc.)

im Studiengang Informatik

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA

Fakultät für Mathematik und Informatik

eingereicht von Max Möbius

geb. am 21.03.1999 in Cottbus

Universitäre Betreuerin: Prof. Dr. Birgitta König-Ries

Betriebliche Betreuerin: Diana Peters (DLR e.V.)

Jena, 24. März 2021

Kurzzusammenfassung

Datenaustausch zwischen Software ist ein immer wichtiger werdendes Thema. Jede Software hat jedoch eine gewisse eigene Struktur und so ist ein einfacher Austausch der Daten meist nicht möglich. Diese Arbeit beschäftigt sich daher mit diesem Thema, dem Datenaustausch zwischen Schnittstellen von Software. Genauer erklärt die Arbeit die Entwicklung einer nutzergestützten Software, die durch Matching die Struktur einer beliebigen Software in die einer anderen beliebigen übersetzen kann. Sie besteht dabei aus einer webbasierten UI und einer Matching-API. Die webbasierte UI arbeitet betriebssystemunabhängig und bietet der nutzenden Person eine einfache Oberfläche, in der diese die Objekte auswählen kann, die miteinander gematcht werden sollen und dann die Parameter dieser Objekte einfach zueinander zuordnen kann. Die Matching-API berechnet für die ausgewählten Objekte mithilfe von Änhlichkeitsmaßalgorithmen die Änhlichkeiten der Parameter zueinander. Diese simplen Algorithmen sowie Datentypvergleiche, Ontologieverwendungen und mehr werden zu hybriden und kompositionalen Matching-Verfahren kombiniert und damit Matching-Vorschläge berechnet, die dem Nutzer zur Hilfe des Matchings angezeigt werden.

Inhaltsverzeichnis

1 Einleitung **6**
1.1 Motivation 6
1.2 Problemstellung und Lösung 6
1.3 Überblick über die Arbeit 7

2 Hintergrund und verwandte Arbeiten **8**
2.1 Matching 8
2.2 Schema Mapping vs. Matching 8
2.3 Lexikalische vs. semantische Matching-Verfahren 10
2.4 Labelbasierte Matching-Verfahren 10
2.4.1 Jaro-Winkler Distanz 10
2.4.2 Jaccard Index 11
2.4.3 Normalisierte Levenshtein Distanz 11
2.5 Ontologiebasiertes Matching 12
2.6 Matchingtool COMA++ 13

3 Konzept **16**

4 Implementierung **17**
4.1 Umsetzung des Konzepts 17
4.2 Webbasierte Software (UI) 18
4.2.1 Aufbau und Funktionsweise 19
4.3 Matching-API 24
4.3.1 Aufbau und Funktionsweise 25
4.3.2 Bestimmung der Matching-Vorschläge 28
4.3.3 Algorithmen zur Bestimmung der Ähnlichkeit 29
4.4 Ergebnisse des Matchings im Test 31

5 Zusammenfassung und Ausblick **36**

6 Literaturverzeichnis **37**

7 Anlagen **39**

Abbildungsverzeichnis

1 Mapping und Matching 8
2 Mischformen des Matchings 9
3 Levenshtein Distanz Matrix 12
4 Matching durch den Nutzer 14
5 Prozess des Matchings in COMA 14
6 Kombination der Matchingergebnisse 15
7 Umsetzung des Konzepts der Matching-Software 17
8 Startansicht 19
9 Aktivitätsdiagramm Startansicht 19
10 Beispielansicht der Objektauswahl 20
11 Aktivitätsdiagramm Objektauswahl 21
12 Beispielansicht der Matching-Ansicht 22
13 Aktivitätsdiagramm Matching-Ansicht 23
14 Beispielansicht des Vorschläge-Pop-Ups 24
15 Klassen für Einstellungen und Start des Servers 25
16 Klassen für das Matching 25
17 Klassen zur Datenkonvertierung 26
18 Klassen zur Verarbeitung von Strings und Erstellen von Paaren 26
19 Klassen zur Verarbeitung von Ontologien 27
20 Sequenzdiagramm Matching API 28
21 Startansicht 39
22 Beispielansicht der Objektauswahl 40
23 Beispielansicht der Matching-Ansicht 41
24 Beispielansicht des Vorschläge-Pop-Ups 42
25 Umsetzung des Konzepts der Matching-Software 43
26 Klassendiagramm Matching API 44
27 Sequenzdiagramm Matching API 45

Tabellenverzeichnis

1 Parameter Virtual Satellite und Part Database 32
2 Parameter-Matching der Nutzer 33
3 Die besten Matchingvorschläge der Ähnlichkeitsmaßalgorithmen 33
4 Die besten Matchingvorschläge der kompositionalen Matching-Verfahren 34

Vorwort

Ich möchte mich an dieser Stelle bei den Begleitern meiner Arbeit bedanken. Meinen Betreuerinnen Prof. Dr. Birgitta König-Ries und Diana Peters möchte ich für ihre gute Anleitung und Unterstützung während des Prozesses danken. Dr. Ing. Alsayed Algergawy hat mich stark bei der Entwicklung des eingebauten schematischen Matchings unterstützt und auf hilfreiche Tipps meines Kommilitonen Maximillian Enderling konnte ich immer zählen. Außerdem möchte ich den Mitarbeitern des DLRs danken, die bei den UI und Matching Tests mit ihrem Feedback mitgewirkt haben.

Aufgrund der besseren Lesbarkeit wird in dieser Arbeit nur ein Geschlecht, das Männliche, explizit genannt, jedoch sollen alle anderen mit einbezogen sein.

1 Einleitung

1.1 Motivation

Heutzutage gibt es viele Programme und Software, die miteinander vernetzt sind. Diese Vernetzung ermöglicht die Kommunikation und den Datenaustausch zwischen diesen Softwares und damit zwischen ihnen eine Automatisierung der Prozesse. Der Datenaustausch ist meist nicht einfach umzusetzen, da jede Software eine andere Struktur und Namensgebung besitzt. Deshalb können Daten meist nicht automatisch übertragen werden, da diese gar nicht in die Struktur der anderen Software passen. Die Nutzer der Software müssen nun „per Hand” selbst die Daten übertragen. Dies ist jedoch mit Aufwand verbunden und sollte einfacher umsetzbar sein. Diese grundsätzliche Struktur, die eine Software besitzt, soll im Nachfolgenden wird weitläufig als Schema bezeichnet. Wird ein Schema in ein anderes übersetzt (dieser Prozess wird Mapping genannt), kann man resultierend auch Daten der Software in eine andere übersetzen.
Wenn nun die Datenstruktur in einer Software dynamisch ist oder Daten zwischen wechselnder Software ausgetauscht werden soll, muss der Nutzer zuerst ein Mapping erstellen, bevor die Daten automatisch übertragen werden können. Muss das nun vor jeder Datenübertragung geschehen, verliert die Software die Automatisiertheit. Die Lösung ist das Matching, bei dem anhand von Hintergrundinformation die Schemata ineinander übersetzt werden, sodass die Daten automatisch übertragen werden können. Matching ist also die Erstellung eines Mappings.
Diese Arbeit geht genau auf dieses Problem ein und erklärt die Entwicklung einer Matching-Software, die Datenschemata zwischen unterschiedlichen APIs nutzergestützt matchen kann.

1.2 Problemstellung und Lösung

Aufgabe: Datenaustausch zwischen **1. Software** und **2. Software**. Zuerst besteht die Annahme, 1. Software enthält verschiedene Objekte (auch Eingabeobjekte genannt) mit Parametern, die jeweils mit Werten gefüllt sind und 2. Software enthält auch verschiedene Objekte (Ausgabeobjekte) mit Parametern, wobei hier einige Objekte keine Parameterwerte besitzen. Datenaustausch bedeutet nun, dass zuerst ein Objekt aus 1. Software und eins aus 2. Software gewählt wird und dann automatisch die Parameterwerte vom Eingabeobjekt zum Ausgabeobjekt übertragen werden (ein Objekt besitzt immer Parameter und dient hier als allgemeine Bezeichnung für die Daten, die ausgetauscht werden sollen).
Beispiel: 1. Software ist eine Datenbank mit verschiedenen Produkten (eine Art von Objekt). Die 2. Software ist ein Programm, welches auch verschiedene Produkte speichern kann, jedoch enthalten Produkte bei der Erstellung keine Werte. Die Parameterwerte eines ausgewählten Produktes aus der Datenbank sollen in das neu erstellten Produkt eingetragen werden.

Problem: Die Parameter von zwei Objekten haben meist unterschiedliche Namen trotz gleicher Bedeutung (werden Synonyme genannt). So ist eine automatische Zuordnung der Parameter ohne weiteres Wissen nicht einfach möglich. Wenn zusätzlich die Parameternamen einer Software nicht festgesetzt sind, liegt zu wenig Wissen vor, um die

Parameter miteinander zu matchen.

Lösung: Schematisches Matching. Die Objekte einer Software folgen einem Schema. Mit Kenntnis der Schemata könnte man diese einfach ineinander, wie bei Sprachen, übersetzen. Jedoch können sich die Parameternamen und -eigenschaften (Datentypen, ...) verändern und damit auch das Schema. Das schematische Matching hilft dabei, zu den aktuellen Schemata die Übersetzung zu bestimmen. Das Matching berechnet dazu anhand von Zusammenhängen zwischen den Parametern Ähnlichkeiten, die für das Matchen der Parameter genutzt werden können. In diesem Fall sollen damit nur Matching-Vorschläge bestimmt werden, da nicht die Gewissheit besteht, dass genug Wissen vorliegt um Parameter sicher miteinander zu matchen. Die Software soll nutzergestützt arbeiten. Der Nutzer wählt die Objekte, die gematcht werden sollen. Beim eigentlichen Matching erhält der Nutzer Vorschläge, wie die Parameter miteinander gematcht werden können. Die Matching-Vorschläge sollten möglichst passend berechnet werden, dies ist jedoch nur mit Hintergrundwissen über die Parameter möglich. Zusätzliches Hintergrundwissen soll durch die Nutzung von Ontologien geschaffen werden. Diese können beispielsweise Synonyme der Parameternamen oder Einheiten zur Verfügung stellen. Außerdem sollen vom Nutzer gematchte Paare (nur deren Parameternamen) gespeichert werden um das Schema-Matching stetig zu verbessern.

1.3 Überblick über die Arbeit

In dieser Bachelorarbeit geht es grundsätzlich um die Entwicklung einer nutzergestützten Software die durch Matching einen Datenaustausch zwischen APIs ermöglichen soll. In Kapitel 2 wird dazu der Hintergrund, also die Aspekte, die eine wichtige Rolle spielen, gezeigt und erklärt werden. Außerdem werden verwandte Arbeiten gezeigt, an welche die Software angelehnt ist. Im dritten Kapitel werden nochmals alle benötigten Konzeptideen aufgelistet, die den Leitfaden zur Implementierung geboten haben. Im vierten Kapitel folgt zeigt dann die eigentliche Implementierung oder auch Umsetzung der Konzeptideen und abschließend wird zusammengefasst, wie gut die Software an das Ziel heran kommt, welches zur Lösung des Problems angestrebt wurde.

2 Hintergrund und verwandte Arbeiten

2.1 Matching

Ein **Match** bezeichnet man als ein Spiel oder eine Begegnung, aber auch als ein Paar oder eine Kombination. Ein **Matching** dahingegen bedeutet, etwas passendes oder übereinstimmendes zu finden. In Prinzip heißt das, es gibt mehrere Objekte oder Lebewesen, die gewisse Eigenschaften besitzen, welche möglichst so miteinander **gematcht** (miteinander verbunden) werden, dass deren Eigenschaften möglichst perfekt miteinander übereinstimmen bzw. sehr ähnlich sind.
Das Matching findet heute Einsatz in den verschiedensten Bereichen – beispielsweise in der Partnervermittlung, der Arbeitsvermittlung, der Statistik oder auch der Graphentheorie. Aber auch im weiteren Sinne findet es seinen Nutzen, so wird es auch im **Image Matching** in der Fotogrammetrie oder der **Attribut-Korrespondenz** zwischen Schemata eingesetzt. So spielt in diesem Projekt genau diese **Attribut-Korrespondenz** eine wichtige Rolle. Dort werden ein oder mehrere Schemata zu einem neuen Schema kombiniert, weshalb man das Prinzip auch als **Schematransformation und -integration** bezeichnet. Verfahren die dafür eingesetzt werden, sind das **Schema Mapping** und **Schema Matching** (vergleiche [5]).

2.2 Schema Mapping vs. Matching

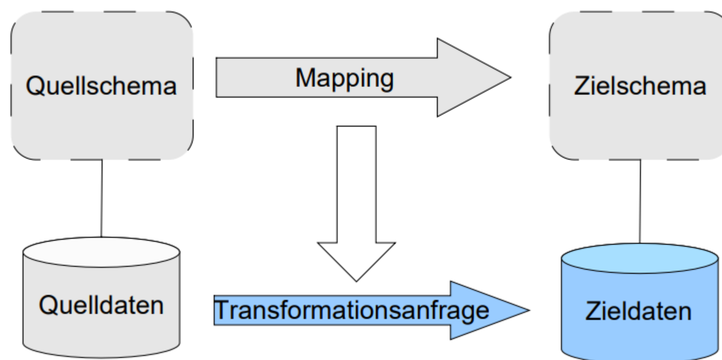

Abbildung 1: Mapping und Matching

Quelle: [5]

Bei beiden Verfahren besteht immer das gleiche Ausgangsproblem: es gibt Quelldaten, die einem Schema entsprechen und Zieldaten, die einem anderen Schema entsprechen. Nun ist das Ziel, das Quellschema möglichst passend bzw. sinnvoll in das Zielschema zu übersetzen (siehe Abbildung 1) – im Prinzip wie das Übersetzen einer Sprache in eine andere. Sobald die Schemata ineinander übersetzbar sind, können auch die Daten von Quelle nach Ziel übersetzt werden.
Das **Mapping** wird eingesetzt, wenn die Schemata sich nicht verändern. Es sind also alle Begriffe und Eigenschaften der verschiedenen Schemata bekannt, weshalb im Vorhinein die richtigen Algorithmen entworfen werden können, die diese Schemata dann übersetzen. Das Verfahren muss dazu erkennen, um welches Schema es sich bei Quelle und Ziel

handelt und soll dann dementsprechend übersetzen (vergleiche [5]).
Beim **Matching** hingegen sind die Schemata von Quelle und Ziel nicht fest gesetzt. Sowohl die Namen als auch die Eigenschaften der Elemente können sich ändern und so ist eine festgelegte Übersetzung nicht möglich. Hierzu sind Korrespondenzen (Zusammenhänge) zwischen den Elementen der Quell- und Zieldatei notwendig. Diese sind grundsätzlich nicht trivial und müssen anhand der Eigenschaften der Werte der Elemente festgestellt werden (Eigenschaften → durchschnittliche Wortlänge, Datentyp, Buchstabenhäufigkeit, . . .). Dazu werden alle Elemente der Quell- und Zieldatei miteinander zu Paaren kombiniert und für die Paare wird nach verschiedenen Eigenschaften die Ähnlichkeit bestimmt. Die höchsten Ähnlichkeiten geben dann an, welche Paare miteinander gematcht werden sollten (vergleiche [5]).
Dafür gibt es grundsätzlich drei verbreitete Verfahren. Beim labelbasierten Verfahren wird nach Algorithmen wie Jaro Winkler Distanz, Jaccard Index oder Levenshtein Distanz die Ähnlichkeit der Namen der Paare bestimmt. Weiterhin gibt es das instanzbasierte Verfahren, bei welchem die Eigenschaften der Werte der Elemente nach Ähnlichkeit überprüft werden. Das dritte ist das strukturbasierte Matching-Verfahren. Dieses versucht Relationen und Nachbarschaftsbeziehungen der einzelnen Elemente beizubehalten, da Attribute einer Relation meist auch in anderen Schemata in denselben Relationen auftauchen. Jedes der Verfahren weißt aber gewisse Nachteile auf. So kann das labelbasierte nur schwer genutzt werden, wenn viele Elementnamen gleich oder ähnlich sind, da die Ähnlichkeit bei Paaren hoch ist, obwohl die Daten gar nicht zueinander passen. Beim instanzbasierten Matching kann es passieren, dass Eigenschaften schlecht gewählt sind oder keine aussagekräftigen Ähnlichkeiten entstehen. Wenn nun auch keine klaren Relationen oder Nachbarschaftsbeziehungen erkennbar sind, kann nicht bzw. nur schlecht nach der Struktur gematcht werden (vergleiche [14]).

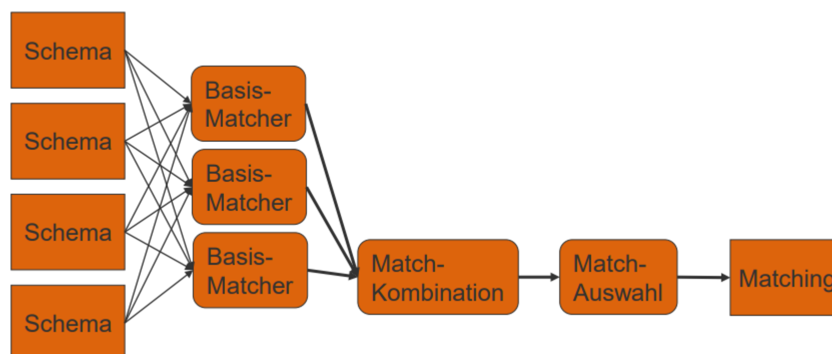

Abbildung 2: Mischformen des Matchings

Quelle: [5]

Oft werden Verfahren einfach kombiniert, da diese in unterschiedlichen Situationen Nachteile aufweisen. Diese Mischformen nennt man dann **Hybrid** oder **Komposit**. Beim hybriden Verfahren werden mehrere Verfahren bzw. auch Algorithmen kombiniert und genutzt um ein besseres Ergebnis zu berechnen. Beim kompositionalen Verfahren werden ebenso verschiedenen Verfahren genutzt, diese werden aber unabhängig voneinander aus-

geführt und am Ende durch Gewichtung zu einem einzelnen Ergebnis kombiniert (siehe Abbildung 2). Beide haben dabei den Vorteil, dass durch die Kombination der Verfahren Nachteile der einzelnen Verfahren entfallen. Beim kompositionalen ist ein weiterer Vorteil, dass durch die unterschiedliche Gewichtung das Endergebnis angepasst werden kann, damit dieses optimal aus den einzelnen berechnet wird (vergleiche [5]).
Das schematische Matching ist somit die beste Wahl für dieses Projekt, da es keine festen APIs gibt, zwischen denen Daten ausgetauscht werden und somit auch keine festen Schemata. Außerdem wird dabei auf kompositionale Matching-Verfahren gesetzt, da so die Verfahren und die Gewichtung der Verfahren variieren und optimal an verschiedene Situationen angepasst werden können. Dort spielen vor allem sowohl labelbasierte Verfahren wie die schon genannten als auch instanzbasierte Verfahren eine große Rolle. Die Verwendung von strukturbasierten Verfahren wird vernachlässigt, da keine Gewissheit besteht, ob Relationen und Nachbarschaftsbeziehung bestehen.

2.3 Lexikalische vs. semantische Matching-Verfahren

Sobald bei den Matching-Verfahren nicht nur die Namen oder Werte der Elemente an sich eine Rolle spielen, sondern auch die Bedeutung dieser, ist es wichtig, zwischen lexikalischen und semantischen Verfahren zu unterscheiden. Bei lexikalischen Verfahren wird rein die Bedeutung der Worte ohne Kontext beachtet. Das kann in der Ähnlichkeitsberechnung oft zu ungewünschten Werten führen. Bei semantischen Verfahren hingegen wird die Hintergrundinformation (Kontext) eines Schemas mit einbezogen. So verändert sich lexikalisch nicht die Bedeutung eines Wortes mit dem Schema, in der diese verwendet wird, semantisch jedoch ist ein Wort in der Bedeutung immer stark von dem Schema abhängig (vergleiche [3]).
In diesem Projekt findet deshalb auch nur die Verwendung von semantischen Verfahren statt, um anhand des Schemas optimal die Ähnlichkeiten zu bestimmen bzw. Matches zu finden.

2.4 Labelbasierte Matching-Verfahren

Diese Verfahren berechnen die Ähnlichkeit zwischen zwei Strings s_1 und s_2. Meist werden dafür normalisierte Verfahren verwendet, welche einen Wert zwischen 0 (keine Ähnlichkeit) bis 1 (Identisch) ausgeben. Nachfolgend werden die drei bekanntesten kurz erläutert.

2.4.1 Jaro-Winkler Distanz

$$\mathrm{sim}_j = \begin{cases} 0 & \text{wenn } m = 0, \\ \frac{1}{3} \cdot \left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m} \right) & \text{sonst.} \end{cases} \tag{1}$$

$$m = \left\lfloor \frac{\max(|s_1|,\ |s_2|)}{2} \right\rfloor - 1 \tag{2}$$

$$\mathrm{sim}_w = \mathrm{sim}_j + l \cdot p \cdot (1 - \mathrm{sim}_j) \tag{3}$$

Die Jaro-Winkler Distanz ist eine Erweiterung der Jaro Distanz. Die Jaro Distanz wird berechnet mit der Gleichung (1). In der Gleichung ist $|s_i|$ die Länge der Strings. Die Anzahl der identischen Zeichen m wird mit der Matching Range bestimmt. Die Matching Range wird mit der Gleichung (2) berechnet und gibt an, wie viele Indizes vor und hinter dem Zeichen auf dasselbe Zeichen im anderen String überprüft werden. Die Anzahl der so gleichen Zeichen ist m. Um t zu berechnen, werden die identischen Zeichen jeweils in der Reihenfolge der beiden Strings miteinander verglichen. Dazu wird die Anzahl der dabei nicht gleichen Zeichen am selben Index (Anzahl der Transpositionen) geteilt durch 2 und das Ergebnis ist t.
Die Jaro-Winkler Distanz wurde entworfen, da die Annahme bestand, dass ein Präfix wichtig sei und bei einer Übereinstimmung in beiden Strings eine Aufwertung bedarf. Für diese Distanz wird im Gegensatz zur Jaro Distanz am Anfang ein gesetzter Präfix l und die Präfixskala p festgelegt. Der Präfix l ist die Länge des gemeinsamen Präfix (maximal 4 Zeichen) und die Präfixskala p ist maximal $\frac{1}{l}$ und wird üblicherweise auf $0,\ 1$ festgelegt. Das p korrigiert die Ähnlichkeit der Strings in Abhängigkeit vom gleichen Präfix l. Die Formel dazu ist (3) (vergleiche [19]).
Die berechnete Distanz ist normalisiert, liegt also zwischen 0 (Identisch) und 1 (keine Ähnlichkeit). Diese ist invers zur Ähnlichkeit, muss also zuerst umgekehrt werden, um einen Ähnlichkeitswert zu liefern. Damit gilt dann für die Ähnlichkeit 0 (keine Ähnlichkeit) bis 1 (Identisch).

2.4.2 Jaccard Index

Der Jaccard Index wird berechnet mit der Formel:

$$J(s_1, s_2) = \frac{|s_1 \cap s_2|}{|s_1 \cup s_2|} \tag{4}$$

s_1 und s_2 werden als Mengen aufgefasst, mehrfach vorkommende Zeichen und die Reihenfolge spielen deshalb keine Rolle. Dabei teilt man die Anzahl der gemeinsamen Zeichen durch die gesamte Anzahl an Zeichen beider Strings. Der dabei entstehende Index liegt zwischen 0 und 1, ist also normalisiert. Der Index muss somit nicht invertiert werden und gibt sofort die Ähnlichkeit zwischen zwei Strings an (vergleiche [12]).

2.4.3 Normalisierte Levenshtein Distanz

Die Levenshtein-Distanz (auch Editierdistanz) zwischen zwei Zeichenketten ist die minimale Anzahl von Einfüge-, Lösch- und Ersetz-Operationen, um die erste Zeichenkette in die zweite umzuwandeln (vergleiche [20]).

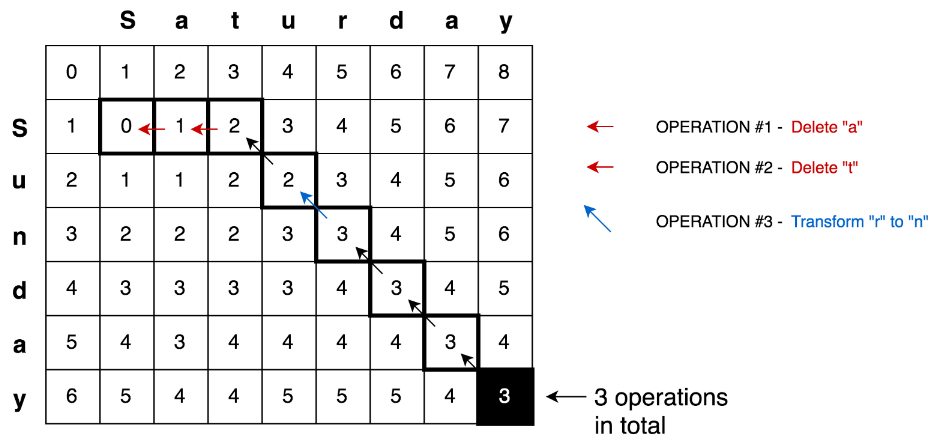

Abbildung 3: Levenshtein Distanz Matrix

Quelle: [16]

Wie in der Abbildung 3 zu erkennen baut der Algorithmus auf einer Matrix D auf, die zuerst berechnet werden muss. Aufsteigend werden den Zeichen jedes Strings Zahlen zugeordnet und die Zahlen werden in die Matrix nach folgenden Regeln eingetragen:

$$
\begin{aligned}
&D_{0,0} = 0 \\
&D_{i,0} = i, (1 \leq i \leq |s_1|) \\
&D_{0,j} = j, (1 \leq j \leq |s_2|) \\
&D_{i,j} = \min \begin{cases} D_{i-1,j-1} + 0, & \text{falls } s_{1i} = s_{2j} \\ D_{i-1,j-1} + 1 & \text{Ersetzung} \\ D_{i,j-1} + 1 & \text{Einfügung} \\ D_{i-1,j} + 1 & \text{Löschung} \end{cases}
\end{aligned} \tag{5}
$$

Unten rechts in Abbildung 3 steht dann die Anzahl der Operationen und somit die Distanz. Diese wird mit dem längeren der beiden Strings normalisiert und da die Distanz invers zur Ähnlichkeit ist, wird sie invertiert. So liegt die Distanz zwischen 0 und 1 (vergleiche [20]).

2.5 Ontologiebasiertes Matching

Ontologien sind Mengen von Begriffen, die in einer Beziehung zueinander stehen und ein gewisses Themengebiet umfassen. Für eine Matching-Software, sind es Dateien, die zusätzlichen Inhalt für die Quell- und/oder Zieldaten liefen. Sie können beispielsweise für das labelbasierte Verfahren Synonyme von Elementnamen speichern. Diese liefern Kontext, mit der eine bessere Ähnlichkeit zwischen den Elementen bestimmt werden kann. Zusätzlich können bereits gematchte Objekte genutzt werden, um deren Inhalt in den Ontologien zu speichern. Wie im Beispiel der Synonyme können so die Ontologien durch die gemachten Elementnamen erweitert werden. Der Vorteil ist hierbei, dass die Ontologien mehr Information liefern, um das Matching optimal durchzuführen. Außerdem ist es sinnvoll, diese Information zu speichern, da, wenn keine Ontologien vorhanden sind,

es keine Hintergrundinformation für das Matching gibt und somit auch keine optimale Ähnlichkeitsbestimmung. (vergleiche [5])

2.6 Matchingtool COMA++

COMA++ ist ein kompositionales Schema Matching Tool, welches mit den semantischen Eigenschaften der zu matchenden Daten der Objekte arbeitet. „Der Rückgriff erfolgt dabei auf Ontologien, Beschreibungen und globale Zusammenhänge zwischen Daten, welche nicht unmittelbar den reinen Daten zu entnehmen sind.“ [6] Das Tool hat große Bekanntheit im schematischen Matching und zeichnet sich durch die unterschiedlichen Matcher, Ansätze des Matchings und Wiederverwendbarkeit von Matching-Ergebnissen aus.
Dieses Tool besitzt die benötigten Funktionen, um die erklärte Problemstellung (Kapitel 1.2) zu lösen. Das Problem liegt jedoch bei der Eingabe der Daten – COMA++ arbeitet nur mit SQL, XML Schema oder OWL. APIs tauschen ihre Daten meist jedoch als JSON aus, der Datentyp wurde für den Datenaustausch entwickelt, ist dementsprechend weniger komplex als XML und schneller sowie einfacher umzusetzen. In diesem Projekt soll auch explizit der Austausch von Daten zwischen APIs ermöglicht werden, was die Verwendung von JSON nahelegt (vergleiche [11] und [6]).

Einige Aspekte dieses Projektes stammen jedoch aus COMA++:

C.1 Verwendung kompositionalen Schema Matchings mit semantischen Matching-Verfahren für die Problemstellung

C.2 Nutzung von Ontologien, um Unabhängigkeit von den vorliegenden Daten, speziell für das semantische Matching, zu erhalten

C.3 nutzergestützte Matchings und die Idee der visuellen Umsetzung für den Nutzer

C.4 Berechnung des Matchings mithilfe des **Similarity Cubes**

C.5 Ideen für einfache und hybride Matching-Algorithmen

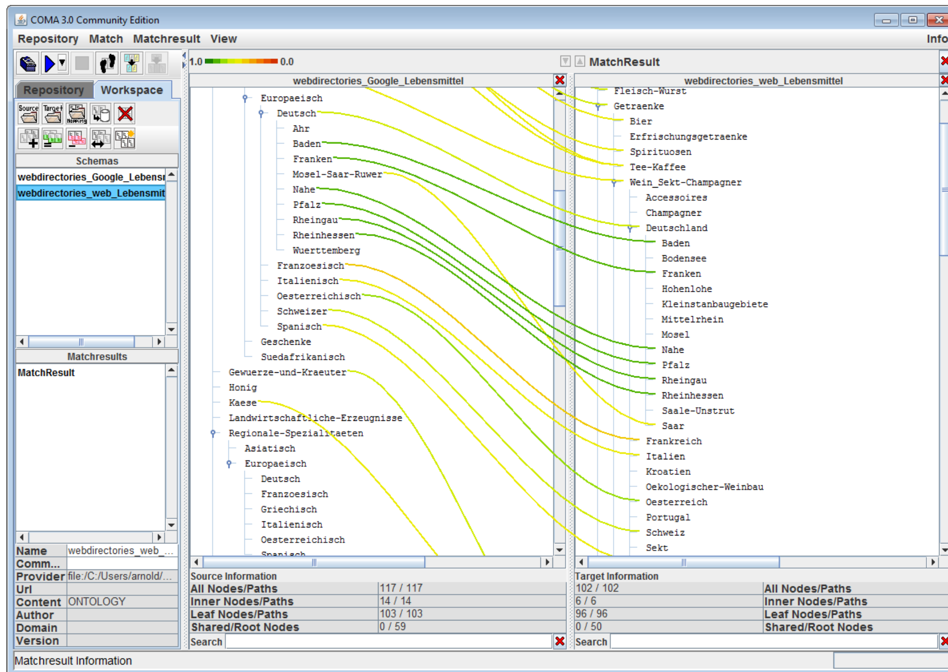

Abbildung 4: Matching durch den Nutzer

Quelle: [17]

Der Aspekt C3 ist in Abbildung 4 dargestellt. Diese Abbildung zeigt im Tool COMA++ das nutzergestützte Matching mit Linien. Dabei symbolisieren die durch den Nutzer erstellten Linien die Matchings zwischen den einzelnen Elementen.

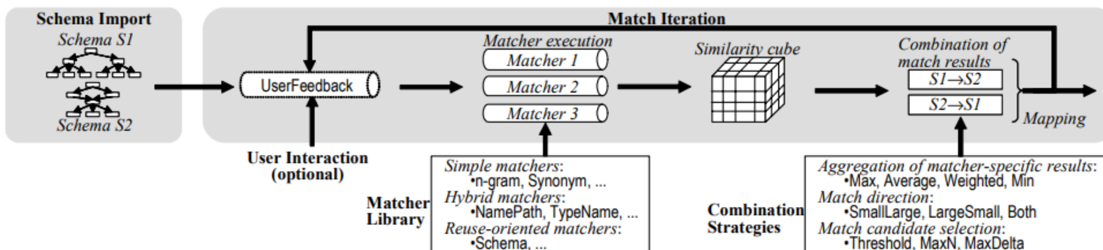

Abbildung 5: Prozess des Matchings in COMA

Quelle: [2]

Die Abbildung 5 zeigt das Matching von den beiden Schemata zum Zielschema. Zuerst werden die Quell- und Zieldaten mit den zugehörigen Schemata importiert. Dann wird iterativ der Matching Prozess durchgeführt, wobei unterschiedliche Matching-Verfahren eingesetzt werden, um das Ergebnis zu optimieren. Der Nutzer hat vor jeder Iteration die Möglichkeit der Interaktion, um manuell zu korrigieren. Danach werden je nach Iterationsschritt bestimmte Matching-Verfahren ausgeführt. Grundsätzlich werden im gesamten Prozess alle Verfahren sukzessiv angewandt. Dadurch entsteht mit k Matching-Verfahren, m Elementen der Quelldatei (S1) und n Elementen der Zieldatei (S2) ein **Similarity Cube** (Größe: $k \cdot m \cdot n$) (vergleiche [6]).

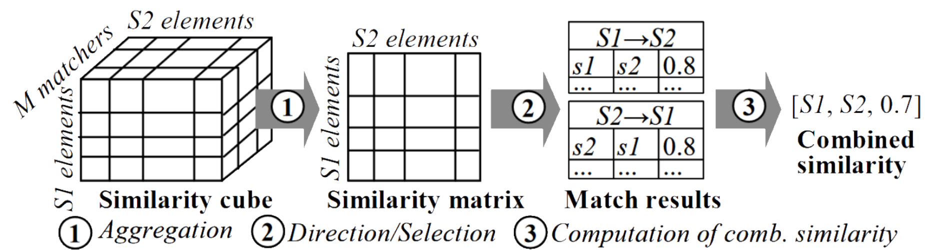

Abbildung 6: Kombination der Matchingergebnisse

Quelle: [2]

In der Abbildung 6 wird die Kombination der Ähnlichkeitswerte der einzelnen Matching-Verfahren gezeigt. Dabei enthält der schon erwähnte **Similarity Cube** auf jeder seiner Seiten die Ähnlichkeitswerte eines Matching-Verfahrens für jedes Element aus S1 und S2. Anschließend werden durch **Aggregation** die Ähnlichkeitswerte der einzelnen Verfahren von den Elementen aus S1 mit den Elementen aus S2 kombiniert. Dadurch entsteht für jedes Verfahren eine **Similarity Matrix**, welche nach den optimalen Matchingkandidaten für jedes Schema gefiltert werden. Abschließend werden die Ergebnisse der beiden Schemata S1 und S2 aggregiert zu kombinierten Ähnlichkeitswerten für ein Schema (vergleiche [2]).
Nachdem die Ergebnisse berechnet wurden, können nach Abbildung 4 nun die Schemata miteinander zu einem gemappt oder durch weitere Iterationen die Ergebnisse noch verfeinert werden.

3 Konzept

K.1 Ermöglichen des Datenaustauschs zwischen verschiedener Software (ändern der Software soll einfach möglich sein)

K.2 Dem Nutzer soll das Durchführen des Matchings einfacher gemacht werden (z.B. durch ein Overlay)

K.3 Die Software soll betriebssystemunabhängig arbeiten

K.4 Der Nutzer bestimmt, wann dieser das Matching starten möchte

K.5 Es soll eine freie Auswahl bei den Objekten geben, die gematcht werden können

K.6 Für die Auswahl der Objekte für das Matching soll der Nutzer filtern können, um einfacher die passenden Objekte zu finden

K.7 Der Nutzer soll einfach die Parameter der Objekte zueinander zuteilen können

K.8 Dem Nutzer sollen für das Matching Vorschläge geboten werden, die das Matching vereinfachen

K.9 Das Matching soll nutzergestützt arbeiten (Nutzer bestimmt selbst, was gematcht werden soll)

K.10 Existenz einer externen Software, die das Berechnen der Matching-Vorschläge durchführt

K.11 Die Matching-Vorschläge sollen unabhängig von den Parameternamen berechnet werden (Parameternamen sollen sich ändern können)

K.12 Zur Ähnlichkeitsbestimmung von Parametern sollen mehrere Matching-Verfahren nutzbar sein bzw. deren Ergebnisse sollten zu einem zusammengefasst werden können

K.13 Für die Berechnung der Matching-Vorschläge soll es eine Möglichkeit geben Daten zu übergeben, die z.B. Synonyme zur Verfügung stellen, um das Matching zu verbessern

K.14 Die Objekte sollten automatisch zu einem gematcht werden und das fertige Objekt soll direkt in der Zielsoftware gespeichert werden

K.15 Gematchte Begriffe nach einem Matching speichern und verwenden für das Berechnen der nächsten Matching-Vorschläge

4 Implementierung

4.1 Umsetzung des Konzepts

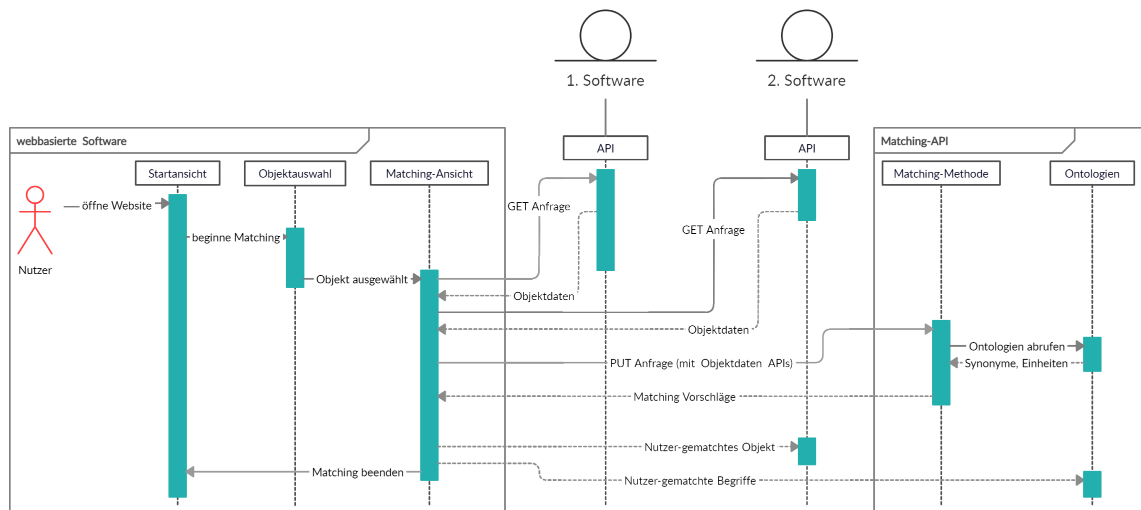

Abbildung 7: Umsetzung des Konzepts der Matching-Software

Die Abbildung 7 zeigt den Plan zur Umsetzung des Konzepts der Matching-Software (Großansicht in Abbildung 25).
Grundsätzlich arbeitet die Software nur mit den APIs, also Schnittstellen der Softwares, zwischen denen die Daten ausgetauscht werden sollen. APIs bieten die Möglichkeit der Kommunikation der Matching-Software mit anderer Software. Daten werden so über Anfragen durch die API der jeweiligen Software an die Matching-Software gesendet. Wenn man die APIs wechseln möchte, müssen die Anfragen und die Verarbeitung der Daten angepasst werden, da die ankommenden Daten ein anderes Format haben können. Die Matching-Software kann so unabhängig von den Softwares arbeiten, die miteinander gematcht werden soll (nach Konzeptpunkt 1).
Dem Nutzer wird zum Durchführen des Matchings eine webbasierte Software (wird hier auch „UI“ genannt) geboten. Diese wird ein einfaches Overlay zur Benutzung haben und viele Aufgaben vom Nutzer übernehmen. Durch die Webbasiertheit kann sie betriebssystemunabhängig arbeiten (nach Konzeptpunkten 2 und 3).
Wenn der Nutzer die UI aufruft, erscheint für diesen die **Startansicht**, in der er ein Matching beliebig starten kann (nach Konzeptpunkt 4).
Nach dem Starten eines Matchings gelangt der Nutzer auf die **Objektauswahl**, wo er von den gekoppelten APIs alle Daten angezeigt bekommt und aus diesen die Objekte für das Matching einfach auswählen kann (nach Konzeptpunkt 5). Die **1. Software** liefert hier die Eingabeobjekte – diese enthalten die Parameter mit den Werten für das Endprodukt – und die **2. Software** liefert die Ausgabeobjekte, welche die Objekte ohne bzw. mit nur wenigen Parameterwerten sind und als Ergebnisse an die Zielsoftware (2. Software) zurück gehen. Dem Nutzer wird die Auswahl der Objekte erleichtert, indem die Objekte hervorgehoben werden, welche zum Matching verfügbar sind. Außerdem gibt es die Funktion des Filterns, die das Ausschließen nach Parametern möglich macht und so die Objektauswahl einschränkt (nach Konzeptpunkt 6).

Nach der Auswahl der Objekte gelangt der Nutzer auf die Matching-Ansicht. Dort bekommt er die Objekte (die von den APIs geladen werden) mit den dazugehörigen Parametern angezeigt und kann einfach Parameter miteinander verknüpfen, die letztendlich auch miteinander gematcht werden sollen (nach Konzeptpunkt 7).
Dem Nutzer soll die Möglichkeit geboten werden, sich beim Matching durch die Software unterstützen zu lassen. Hierbei ist jedoch wichtig, dass die Matching-Software nur nutzergestützt arbeiten soll, sie dem Nutzer für das Matching also lediglich Vorschläge anbietet. Dort gibt es die Möglichkeit, alle Matching-Vorschläge (nur die besten) direkt zuzuordnen und sich für einzelne Parameter mehrere Vorschläge anzeigen zu lassen (nach Konzeptpunkten 8 und 9).
Zur Berechnung der passenden Matching-Vorschläge gibt es die Schnittstelle **Matching API**. Diese ist eine externe Software, die durch Anfragen zum Matching die mitgelieferten Objekte bzw. Parameter der Objekte matcht und die gematchten Parameter als Vorschläge zurückgibt (Ein- und Ausgabe als JSON). Der Vorteil ist hierbei, dass die Software für die UI und die Software zum Berechnen der Matching-Vorschläge getrennt arbeiten und so unabhängig voneinander verändert oder auch ausgetauscht werden können. Da die Matching API als Schnittstelle arbeitet, besitzt sie auch keine Einschränkung in der Wahl der Programmiersprache, in der diese umgesetzt werden muss (nach Konzeptpunkt 10).
Bei der Berechnung der Matching-Vorschläge wird auf semantische Matching-Verfahren gesetzt, die zwischen Parametern die Ähnlichkeit, unabhängig von den Parameternamen, berechnen. Aufgrund einer Similarity Matrix kann mit mehreren Verfahren die Ähnlichkeit bestimmt und zu einem optimalen Ergebnis zusammengefasst werden. Dabei bietet die Matching-API die Möglichkeit, Ontologien zu verarbeiten. Je nach gekoppelten APIs an der webbasierten Software können entsprechende Ontologien dieser (mit z.B. Synoymen für Parameternamen) an die Matching-API weitergeben werden. Durch die Ontologien, werden einfacher und auch bessere Matching-Vorschläge berechnet (nach Konzeptpunkten 11, 12 und 13).
Wenn der Nutzer das Matching beenden bzw. speichern möchte, werden die Objekte automatisch von der UI zu einem Objekt gematcht und an die Ziel API (2. Software) weitergeben. Außerdem gibt die UI die gematchten Parameternamen an die Matching-API weiter, wo die Begriffe in einer Ontologie gespeichert werden, um diese für die Berechnung der nächsten Matching-Vorschläge zu nutzen (nach Konzeptpunkten 14 und 15).

4.2 Webbasierte Software (UI)

Programmiert wird die UI mit **Javascript**, jedoch unter der Verwendung des Frameworks **React** [10], da dieses viele Funktionen besitzt, die die Erstellung einer webbasierten Software erleichtern. Mit React lassen sich außerdem leichter komplexe und dynamische Webseiten erstellen. Es bietet weiterhin beispielsweise die Wiederverwendbarkeit von Komponenten und bessere Wartbarkeit (vergleiche [1]).

4.2.1 Aufbau und Funktionsweise

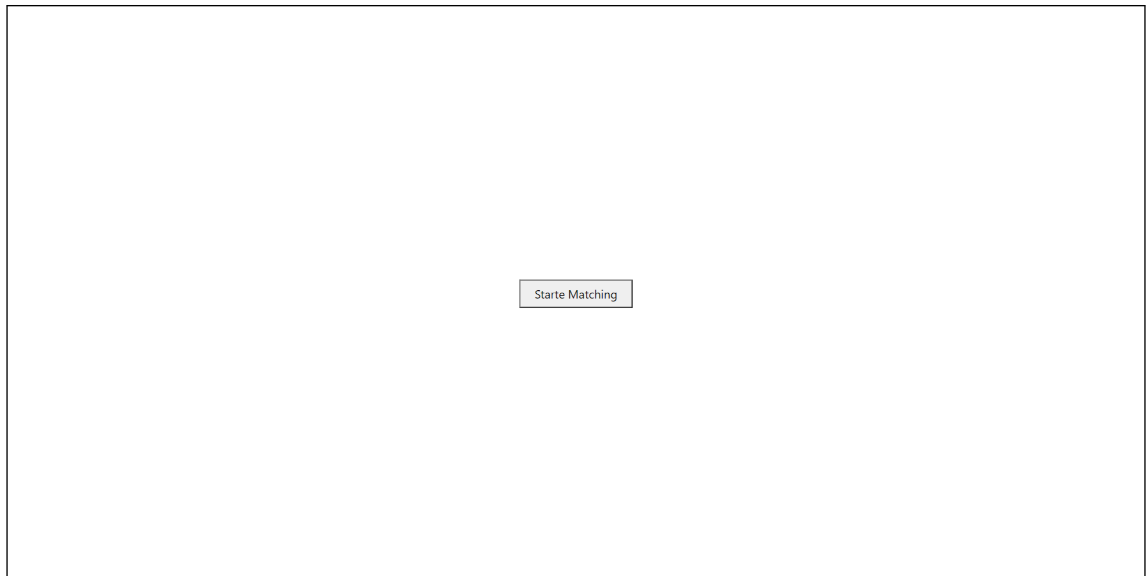

Abbildung 8: Startansicht

Abbildung 8 stellt die Seite dar, die dem Nutzer gezeigt wird, sobald dieser den spezifischen Web-Link öffnet (Großansicht in Abbildung 21).

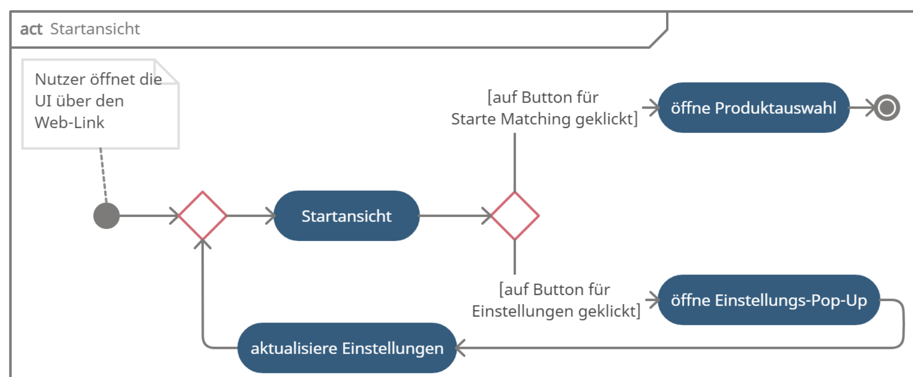

Abbildung 9: Aktivitätsdiagramm Startansicht

Zum Verständnis der Funktionweise dient das Aktivitätsdiagramm 9. Auf der **Startansicht** findet der Nutzer den Button **Starte Matching** und **Einstellungen**. Wenn der Button für die Einstellungen geklickt wird, öffnet sich ein Pop-Up, in welchem der Nutzer gewisse Einstellungen vornehmen kann. Beim Klick auf **Starte Matching** wird der Nutzer zur Webseite **Objektauswahl** weitergeleitet.

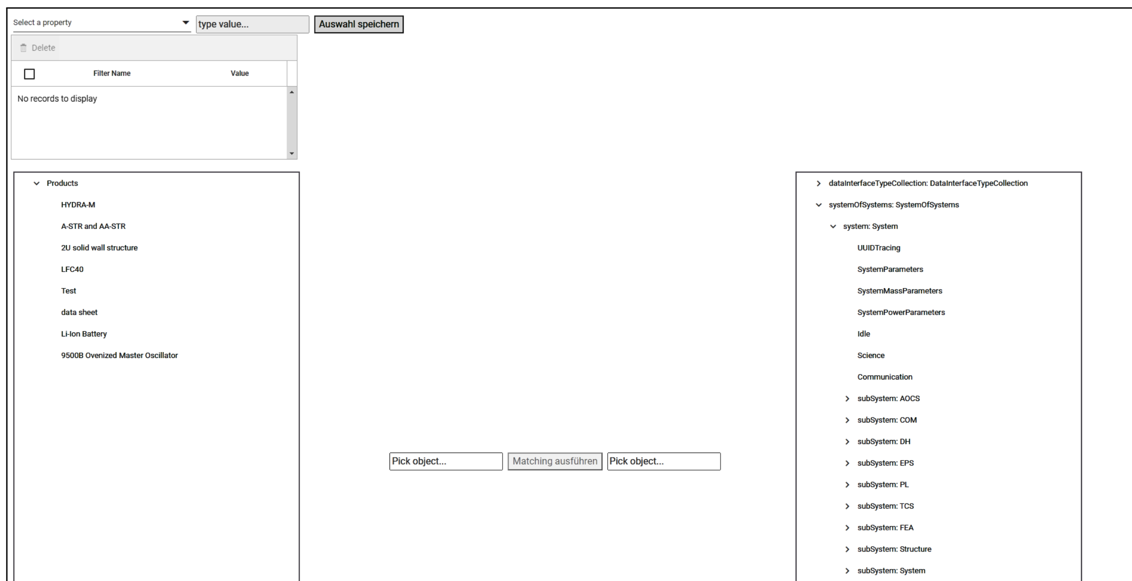

Abbildung 10: Beispielansicht der Objektauswahl

Diese Objektauswahl ist als Beispielansicht in Abbildung 10 zu sehen (Großansicht in Abbildung 22). Diese kann sich leicht verändern, wenn die APIs ihre Ausgaben verändern oder andere APIs gekoppelt sind.

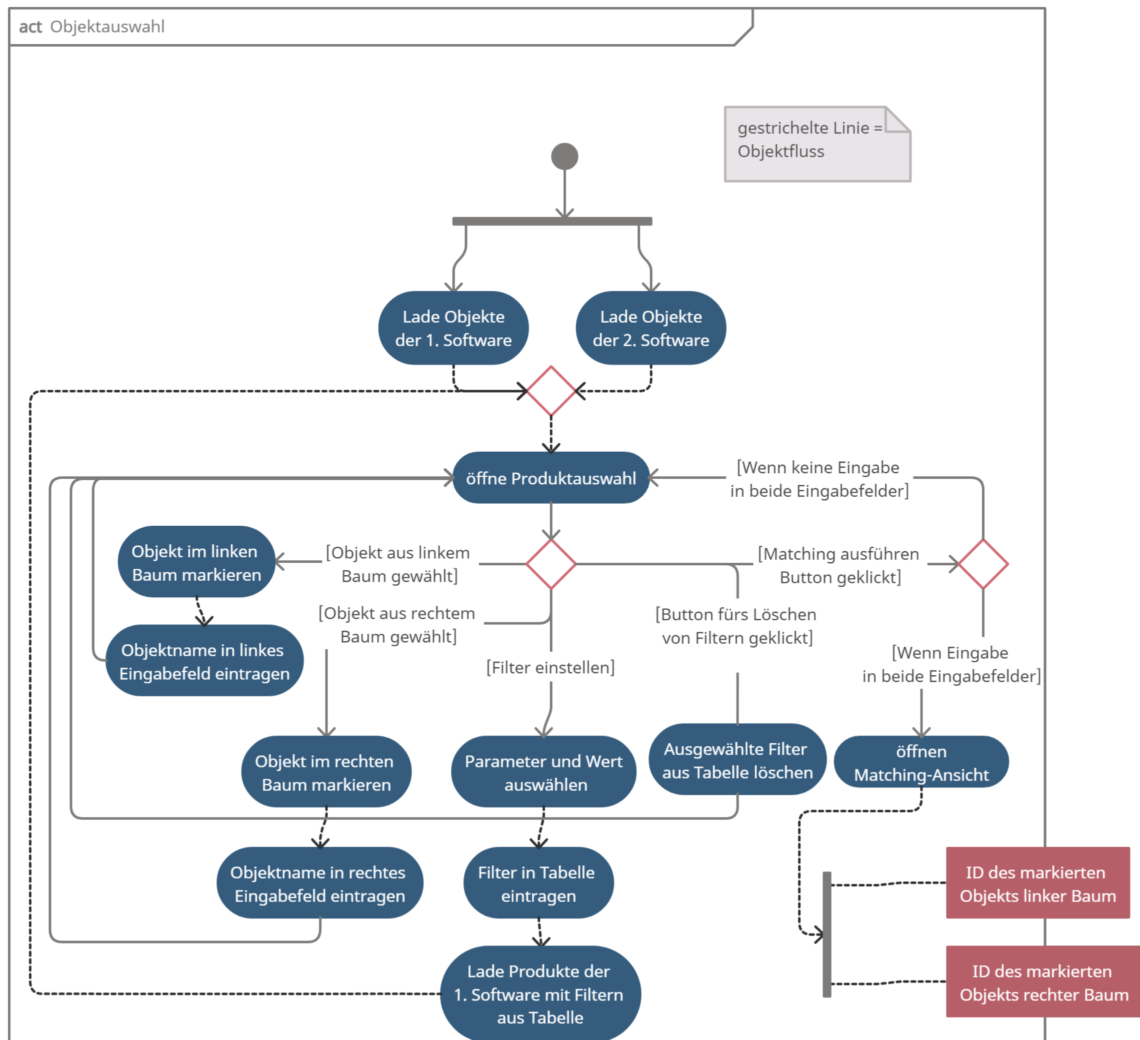

Abbildung 11: Aktivitätsdiagramm Objektauswahl

Zum Verständnis dient hier das Aktivitätsdiagramm 11. Auf der Webseite sind links und rechts Daten in Baumstruktur dargestellt. Der linke Baum enthält die Objekte der **1. Software** und der rechte Baum die der **2. Software** (Daten werden über die gekoppelten APIs geladen). Beide Bäume können beliebig mit den Pfeilen neben den Namen ausgeklappt werden. Außerdem dienen die Bäume zur Auswahl der Objekte der jeweiligen Software, die zusammen gematcht werden sollen. Ausgegraute Namen sind dabei Objekte, die nicht für das Matching verwendet werden können. Für die Auswahl der Objekte klickt der Nutzer in diesen Bäumen die Namen für die Objekte an. Diese werden markiert und stehen dann auch in den Eingabefeldern in der Mitte. Diese Eingabefelder stellen eine erneute Bestätigung dafür dar, dass ein Objekt ausgewählt wurde und dass es auch für das Matching genutzt werden kann.
Der Nutzer hat für den linken Baum auch die Möglichkeit des Filterns (einfach Erweiterbar für rechten Baum). Dazu wählt dieser oben links (siehe Abbildung 22) im Dropdown Menü einen Parameter, gibt den gewünschten Wertebereich im Eingabefeld daneben ein und klickt den Button **Auswahl speichern**. Der Filter wird in der Tabelle darunter gespeichert bzw. angezeigt und auf die Objekte des linken Baumes angewendet (Auswahl wird dementsprechend angepasst). Der Nutzer kann in der Tabelle jedoch auch einzelne Filter

über die Checkboxes auswählen oder mit dem Button **Delete** löschen.
Wenn aus jedem Baum ein matchbares Objekt ausgewählt wurde bzw. in beiden Eingabefeldern ein Namen steht, wird der Nutzer mit dem Button **Matching ausführen** auf die Webseite **Matching-Ansicht** weitergeleitet. Zusätzlich werden die IDs der gewählten Objekte mit übergeben.

Abbildung 12: Beispielansicht der Matching-Ansicht

Die Matching-Ansicht in der Abbildung 12 ist nur eine Beispielansicht, da sie sich mit den ausgewählten Objekten verändert (Großansicht in Abbildung 23).

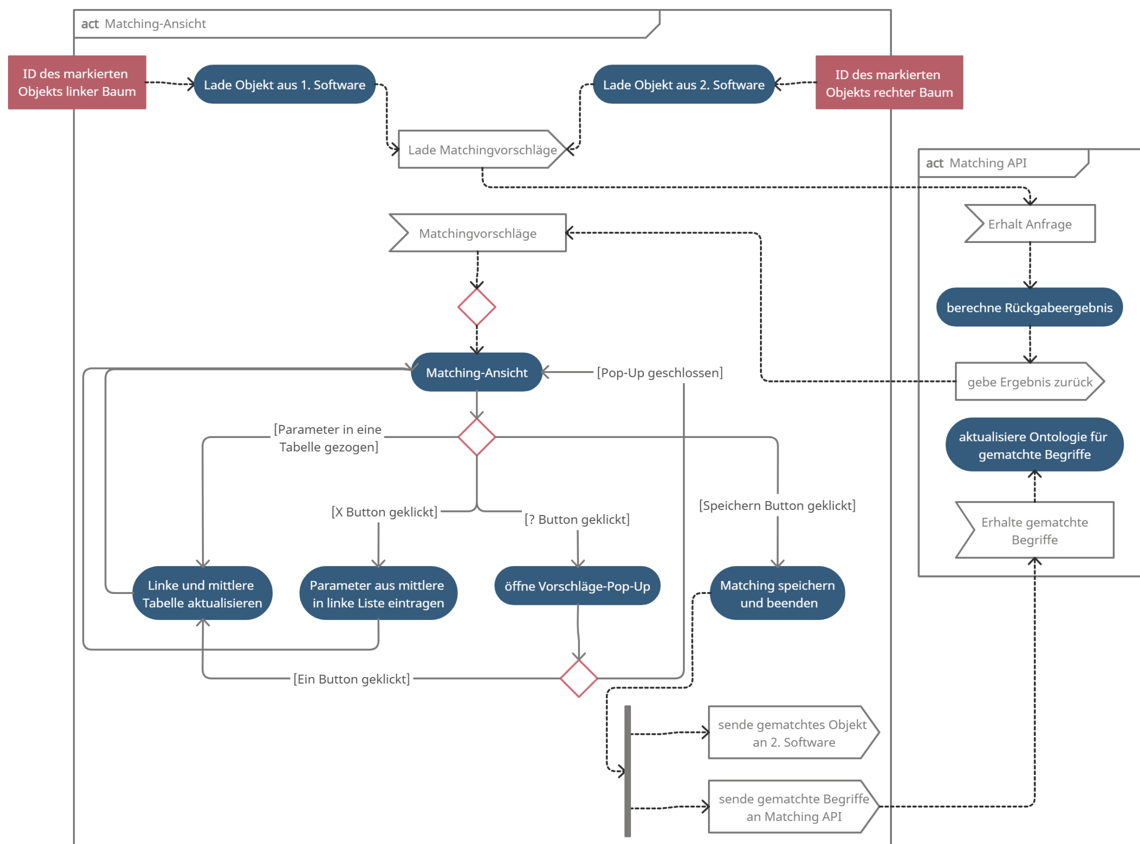

Abbildung 13: Aktivitätsdiagramm Matching-Ansicht

Die Funktionalität ist über das Aktivitätsdiagramm 13 einsehbar (Großansicht in Abbildung 13). Die übertragenen IDs dienen dazu, die Objekte aus den APIs zu laden. Deren Parameter werden in den Tabellen dargestellt mit Name und Wert. Links sind dabei die nicht zugeordneten Parameter der **1. Software** (darüber der Name des Objekts) und in der rechten Tabelle die Parameter der **2. Software** (darüber der Name des Objekts). Parameter werden dort per **Drag and Drop** zugeteilt – hierzu dient die Tabelle in der Mitte. Parameter aus der linken Tabelle können dazu in die Zeilen der mittleren oder auf die Parameter der rechten Tabelle gezogen werden, diese werden dann in der mittleren Tabelle an dem entsprechenden Index eingeordnet. Ein Parameter aus der **1. Software** ist zu einem aus der **2. Software** zugeordnet, wenn dieser sich auf demselben Index in der mittleren Tabelle befindet. So muss der Nutzer, um dem ersten Parameter der rechten Tabelle einen Parameter zuzuordnen, diesen auf die erste Position der mittlere Tabelle setzen.
Die Parameter können durch Drag and Drop aus der mittleren in die linke Tabelle entfernt werden. Einfacher geht es mit dem in jeder Zeile befindlichen Button **X**.
Wenn die Webseite die **Matching-Ansicht** öffnet, werden die ausgewählten Objekte an die **Matching-API** weitergeben, damit diese die Matching-Vorschläge berechnet und zurückgibt.

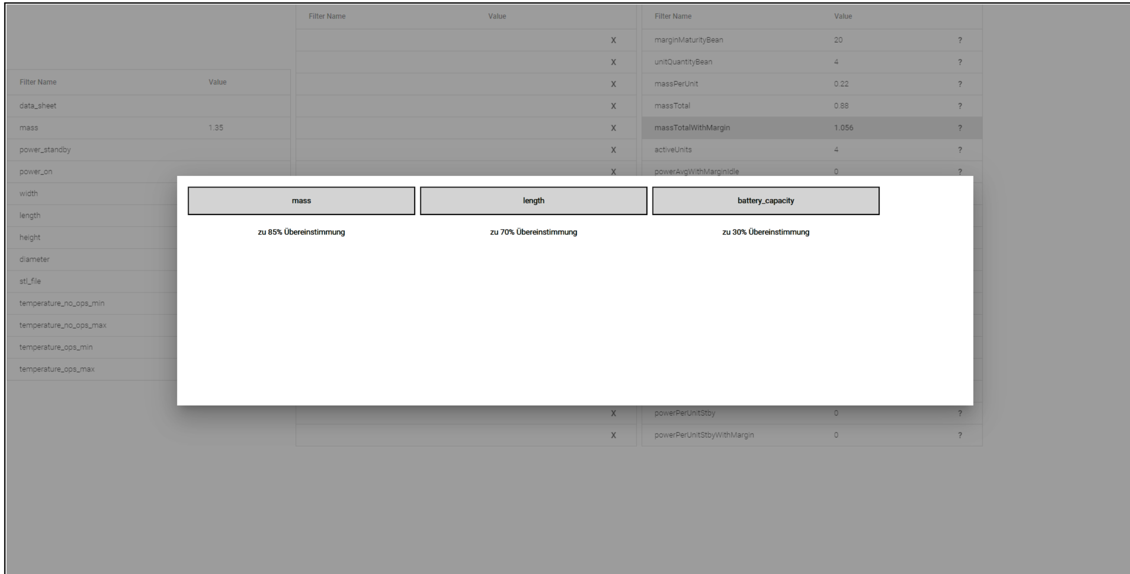

Abbildung 14: Beispielansicht des Vorschläge-Pop-Ups

Der Nutzer kann für jeden Parameter der rechten Liste mit dem Button **?** ein **Pop-Up** öffnen (siehe Abbildung 14, Großansicht in Abbildung 24), das drei Matching-Vorschläge für diesen Parameter anzeigt. Wenn einer dieser Parameter, die vorgeschlagen werden, schon zugeordnet ist, wird dazu einer Kommentar in rot angezeigt. Zusätzlich steht zu jedem Vorschlag die Information, wie ähnlich sich die Parameter zueinander sind, damit der Nutzer anhand dessen entscheiden kann, welchen Vorschlag dieser wählen möchte. Wenn der Nutzer einen der drei Buttons klickt, also ein Vorschlag anwenden möchte, schließt sich das Pop-Up und der Parameter wird direkt an den spezifischen Index in der mittleren Tabelle zugeordnet. Will der Nutzer keinen Vorschlag wählen, kann dieser durch ein Klicken außerhalb des Pop-Ups dieses einfach schließen.
Mit dem Button **Speichern** unten rechts, wird das Matching gespeichert, an die **2. Software** gesendet, die gematchten Begriffe werden in einer Ontologie der **Matching-API** gespeichert und das Matching für den Nutzer wird beendet, er wird also zurück auf die **Startansicht** gebracht.

4.3 Matching-API

Die Matching-API wird in der Sprache **Java** [13] programmiert. Zur Umsetzung als Schnittstelle dient das Framework **Spring Boot** [18]. Dieses stellt REST Funktionen zur Verfügung, um über diese Anfrage an die Matching-API von z.B. der UI aus zu stellen. Damit in Java die Einbindung und Aktualisierung von Bibliotheken einfacher ist, wird ein **Maven Projekt** [8] genutzt. Eine Bibliothek ist **Gson** [9], welche JSON Dateien verarbeiten kann. Die zu matchenden Objekte werden von der UI immer als JSON übergeben und die Bibliothek dient zur Verarbeitung der Eingaben und Erstellung der Ausgaben. Die zweite verwendete Bibliothek ist **Apache Jena** [7], welche Ontologien wie in dieser Software die Ontologie für Einheiten und die für Synonyme verarbeitet. Als letztes wird noch **Java String Similarity** [15] verwendet. Diese Bibliothek stellt bekannte Algorithmen zur Berechnung von Ähnlichkeiten zwischen Strings zur Verfügung.

4.3.1 Aufbau und Funktionsweise

Das gesamte Klassendiagramm zeigt den Aufbau und die Funktionsweise der Matching API und befindet sich in den Anlagen (Abbildung 26).

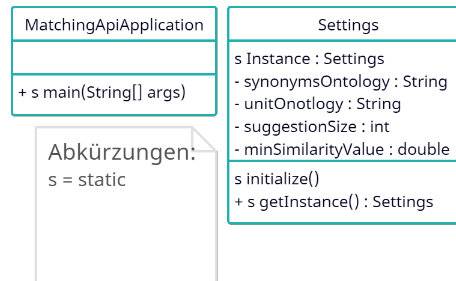

Abbildung 15: Klassen für Einstellungen und Start des Servers

Wichtig ist, dass **s** für **static** vor Variablen und Methoden steht. **MatchingApi-Application** und **Settings** sind für das Matching grundsätzlich eher unwichtig. **MatchingApiApplication** dient nur zum Start der Matching API indem es einen Server aufsetzt, mit welchem die webbasierte Software kommunizieren kann. **Settings** liest **application.properties** (Settings File) aus und speichert deren Variablen in einer Instanz von sich selbst. So können die Variablen direkt von dieser Instanz beim Start der API abgerufen werden.

Der Zweck der weiteren Klassen kurz erläutert:

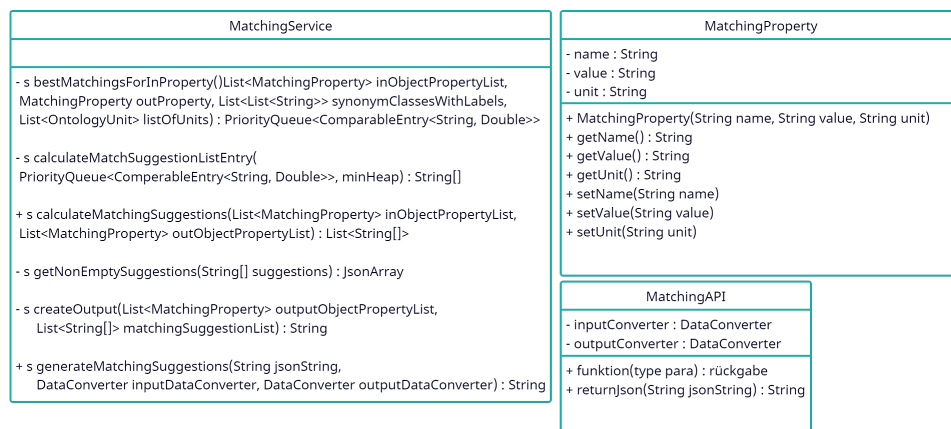

Abbildung 16: Klassen für das Matching

- **MatchingAPI** → ist eine Klasse und dient als Schnittstelle, die die Anfragen verarbeitet, die an die Matching API gestellt werden.
- **MatchingService** → nimmt Objekte an und matcht diese miteinander, überträgt also deren Parameter in das interne Format, berechnet das Matching und gibt die Matching-Vorschläge aus.
- **MatchingProperty** → ist das interne Datenformat für die Parameter der Objekte und speichert jeweils den Namen, Wert und die Einheit eines Parameters.

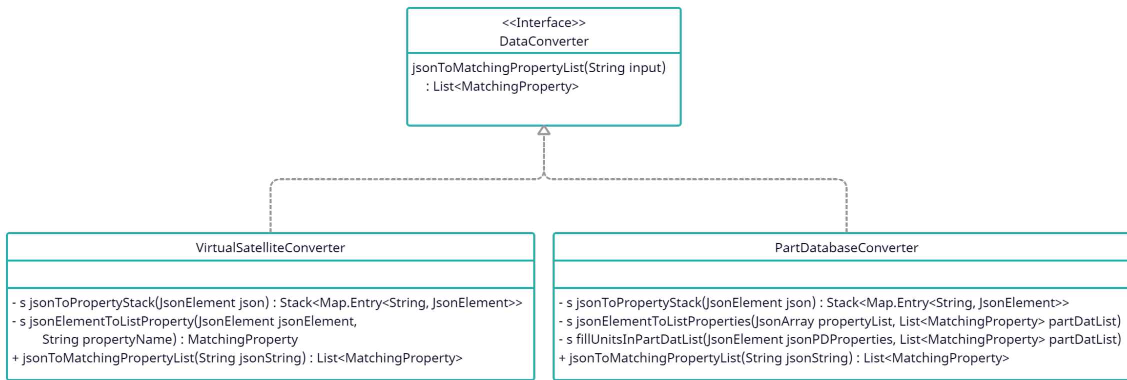

Abbildung 17: Klassen zur Datenkonvertierung

- **DataConverter** → ist ein Interface zur Erstellung von Konvertern, welche die spezifischen externen Formate der zu matchenden Objekte in das interne Format (MatchingProperty) umwandelt.
- **PartDatabaseConverter** → eine Implementierung des **DataConverters**, welche Objekte der **Part Database API** ins interne Format umwandelt (stellt Eingabeobjekte).
- **VirtualSatelliteConverter** → eine Implementierung des **DataConverters**, welche Objekte der **Virtual Satellite API** ins interne Format umwandelt (stellt Ausgabeobjekte).

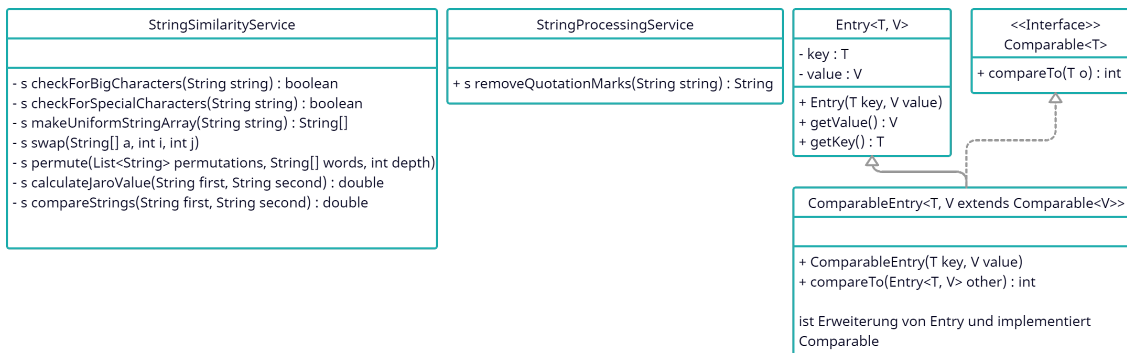

Abbildung 18: Klassen zur Verarbeitung von Strings und Erstellen von Paaren

- **StringProcessingService** → enthält Funktionen, die Strings verarbeitet.
- **StringSimilarityService** → berechnet für zwei Strings die Ähnlichkeit.
- **Entry** → ist eine generische Klasse, die genutzt wird um Paare beliebiger zweier Typen zu erstellen.
- **Comparable** → ist ein generisches , von Java vordefiniertes Interface, mit dem man Klassen erstellen kann, die Objekte beliebig miteinander vergleichen können (wenn diese vergleichbar sind).

- **ComparableEntry** → erweitert die Klasse **Entry** und implementiert das Interface **Comparable**. Damit kann man nur Paare erstellen, bei denen die Typen miteinander vergleichbar sind.

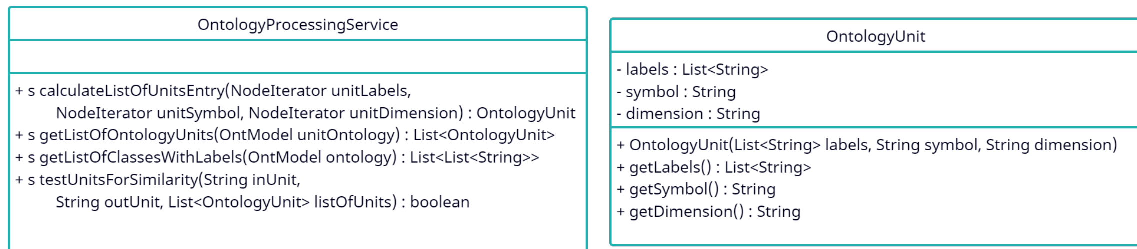

Abbildung 19: Klassen zur Verarbeitung von Ontologien

- **OntologyUnit** → ist ein Datenformat, welches genutzt wird, um die Labels bzw. Synonyme, das Symbol und die Dimension (z.B. Millimeter = Längendimension) einer Einheit zu speichern.

- **OntologyProcessingService** → enthält Funktionen, um Ontologien zu verarbeiten.

Grund des Aufbaus: Die Klasse **MatchingAPI** dient als zusätzliche Schnittstelle zwischen der webbasierten UI und dem **MatchingService** und gibt für die Eingaben vor, wie diese in das interne Format umgewandelt werden. Die Klasse **MatchingService** ist für das Matching selbst zuständig, verarbeitet also die Eingabe in das interne Format, führt das Matching der Parameter aus und wandelt die Matching-Vorschläge in das Ausgabeformat **JSON** um. Die Funktionen für das Umwandeln ins interne Format sind dabei ausgelagert, damit diese leichter geändert werden können. Grund dafür ist, dass, wenn mit der UI andere APIs gekoppelt werden, sich das Format der Objekte ändern kann, die als Eingabe an die Matching API gehen. So kann für jede gekoppelte API eine Implementation des **DataConverters** geschrieben werden, die das spezifische Format in das interne der Matching API umwandeln kann. Die Funktion zum Umwandeln der Matching-Vorschläge in die Ausgabe ist nicht ausgelagert, da hier das Ausgabeformat festlegt ist. Die anderen Klassen dienen hauptsächlich dazu, Instanzen von Objekten zu erstellen oder Funktionen für das Matching bereitzustellen.

4.3.2 Bestimmung der Matching-Vorschläge

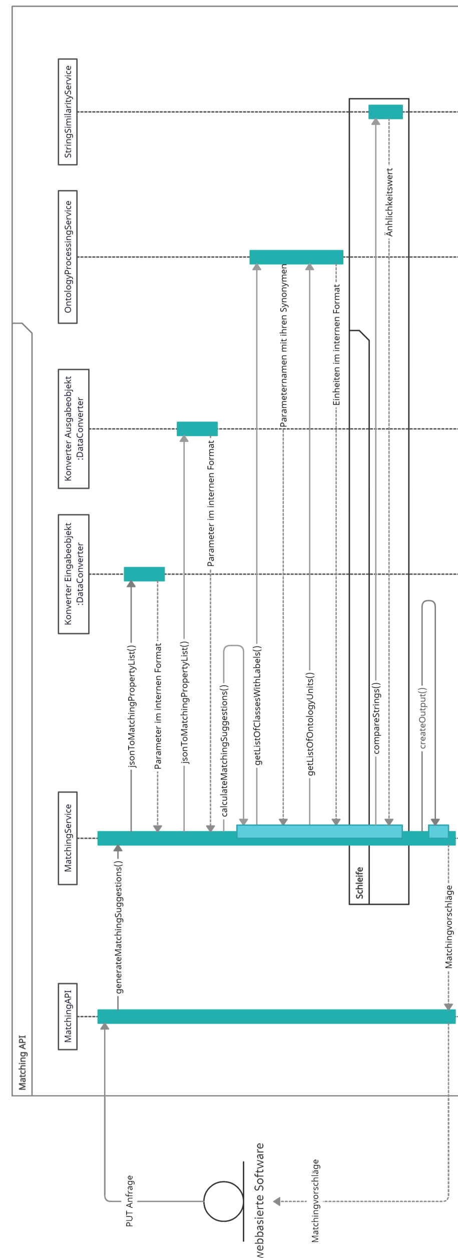

Abbildung 20: Sequenzdiagramm Matching API

Von der Anfrage bis zu den tatsächlichen Matching-Vorschlägen liegt ein langer Weg. So ist der Ablauf einer Anfrage in Abbildung 20 dargestellt und in den Anlagen nochmal in groß zu finden (Abbildung 27). Dort sind jedoch nicht alle Klassen dargestellt, da einige weniger wichtig für die Verarbeitung einer Anfrage sind. Eine Anfrage für das Matching zwischen Objekten wird als PUT Anfrage von der webbasierten Software an die Matching API gestellt. Hierbei sendet diese noch ein **Body** im Datentyp JSON als String mit, der die zu matchenden Objekte (Eingabe- und Ausgabeobjekt) enthält. Falls nötig, enthält dieser auch weitere benötigte Dateien, wie beispielsweise alle Parameter einer Schnittstelle. Die Anfrage wird dann von der Schnittstelle **MatchingAPI** angenommen und verarbeitet.

Dazu wird die Funktion **generateMatchingSuggestion()** aus **MatchingService** aufgerufen. Diese nimmt den Body der Anfrage und zwei Konverter an und führt das Matching aus. Die beiden Konverter sind abgestimmt auf das externe Format der Objekte von **1. Software** und **2. Software** (siehe Abbildung 25). Sie enthalten also die Information um die eingehenden Objekte in das interne Format **MatchingProperty** umzuwandeln.
MatchingService ruft dann **jsonToMatchingPropertyList()** aus dem Eingabobjekt- und Ausgabeobjektkonverter auf. Diese extrahieren aus dem „Body“ die beiden Objekte und wandeln diese in das interne Format um. An **MatchingService** wird jeweils eine Liste der Parameter der Objekte gesendet, die nur Instanzen von **MatchingProperty** enthalten.
Danach wird die Methode **calculateMatchingSuggestions()** aufgerufen. Diese berechnet für jeden Ausgabeparameter die drei ähnlichsten Eingabeparameter. Zuerst holt sie mit **getListOfClassesWithLabels()** Synonyme für verschiedene Parameter (falls Ontologie für Synonyme vorhanden), um anhand dieser besser die Ähnlichkeit der Parameter zu bestimmen. Außerdem holt diese mit **getListOfOntologyUnits()** alle möglichen Einheiten, um vorab prüfen zu können, ob Parameter die verglichen werden überhaupt dieselbe Einheit aufweisen. Zur Berechnung der Ähnlichkeit der Parameter wird dann die Methode **compareStrings()** angewendet, welche die Ähnlichkeit zwischen zwei Strings berechnet.
MatchingService verarbeitet mit **createOutput()** die Matching-Vorschläge zu einem Output (als JSON) und gibt diesen dann an die **MatchingAPI**. Die **MatchingAPI** wiederum gibt den Output als Antwort auf die PUT Anfrage zurück an die webbasierte Software.

4.3.3 Algorithmen zur Bestimmung der Ähnlichkeit

Wie in Kapitel 4.3.2 erwähnt, wird die Ähnlichkeit zwischen den Parametern aus dem Eingabe- und Ausgabeobjekt durch **MatchingService.calculateMatchingSuggestions()** bestimmt. Als Eingabeparameter müssen hier dementsprechend auch das Eingabe- und Ausgabeobjekt mit übergeben werden. Zunächst werden Synonyme und Einheiten aus den Ontologien geladen. Dazu wird überprüft, ob eine Ontologie für Synonyme vorhanden ist. Wenn eine Ontologie vorhanden ist, wird
OntologyProcessingService.getListOfClassesWithLabels() aufgerufen. Diese Funktion gibt für jeden Parameternamen, aus der **1. Software** ein Objekt mit allen Synonymen bzw. Labels zurück. Danach werden mit
OntologyProcessingService.getListOfOntologyUnits() alle Einheiten aus der Kategorie **Unit** und **PrefixedUnit**, von der Ontologie **OM 2: Units of Measure** (Webseite: [4]) geladen.
Zur optimalen Bestimmung der Ähnlichkeit werden kompostionale Matching-Verfahren eingesetzt. Diese Verfahren kombinieren simple labelbasierte, instanzbasierte und strukturbasierte Verfahren miteinander. Hier kommen aber nur hybrider Verfahren, also Kombinationen dieser simplen Verfahren, zum Einsatz und diese können noch unterschiedliche gewichtet werden, um die endgültige Ähnlichkeit zwischen zwei Strings optimal zu bestimmen.
Bei der Berechnung der Ähnlichkeiten wird jeder Eingabeparameter mit jedem Ausgabeparameter verglichen. Zuerst wird dazu getestet, ob die Einheiten der Parameter dieselbe Dimension haben (z.B. Dimension Länge). Wenn diese nicht übereinstimmen, werden sie nicht verglichen. Danach wird über die Parameternamen die eigentlich Ähnlichkeit berechnet. Bei den Parameternamen wird grundsätzlich die Struktur mit einbezogen. Bereits

in der UI werden die Parameternamen der Eltern mit denen der Kinder kombiniert. Von links nach rechts kennzeichnet ein Punkt immer das nächste Kind und ganz rechts steht der eigentlich Name eines Parameters. Mit der zusätzlichen Information lassen sich Parameter besser unterscheiden. Zur eigentlichen Ähnlichkeitsbestimmung werden die Ähnlichkeitsmaßalgorithmen: Jaro-Winkler Distanz, Jaccard Index und normalisierte Levenshtein Distanz eingesetzt. Diese berechnen zwischen zwei Strings bzw. hier Parameternamen einen Wert zwischen 0 (keine Ähnlichkeit) und 1 (Identisch). Vor der Berechnung wird immer erst für die Parameter des Eingabeobjektes geprüft, ob sie in der Ontologie der Synonyme stehen, denn so werden nicht nur die Parameternamen des Eingabeobjekts, sondern auch die Synonyme des Namens mit dem Parameternamen des Ausgabeobjekts verglichen. Vor der Berechnung werden die zu vergleichenden Strings auf dieselbe Form gebracht.
Jeder String wird dazu auf Auffälligkeiten wie große Buchstaben oder spezielle Zeichen (`/`, _, `.`, ...) überprüft. Wenn mindestens eins aus diesen Gruppen gefunden wurde, wird der String an der Stelle in einzelne Wörter aufgeteilt. Im Falle von großen Buchstaben werden alle zu kleinen Buchstaben ersetzt und wenn spezielle Zeichen vorhanden sind, werden diese entfernt. Diese einzelne Wörter werden nun abhängig von dem jeweils anderen String in eine bestimmte Reihenfolge gebracht. Zuerst werden alle Wörter der beiden Strings zu Paaren kombiniert, die Ähnlichkeit der Paare bestimmt und die Paare damit in einem Max-Heap eingeordnet. Der Max-Heap gibt so immer die Paare zurück, die sich am ähnlichsten sind. Nun werden die zu vergleichenden Strings neu aufgebaut. Dazu wird immer das nächst-ähnlichste Paar aus dem Max-Heap entnommen und am selben Index in den neuen Strings eingetragen. Dies wird so lange gemacht, bis der String mit weniger Wörtern gefüllt ist, wobei Paare verworfen werden, wo mindestens einer der beiden Wörter schon im neuen String eingetragen ist. Abschließend wird der String mit mehr Wörtern, mit den restlichen fehlenden Wörtern aufgefüllt, falls beide Strings natürlich nicht gleich lang sind. Diese neue Anordnung schafft den optimalen Ähnlichkeitswert zwischen zwei Strings, da die zueinander ähnlichsten Wörter immer nahe beieinander sind.
Beispiel: Ähnlichkeitsberechnung zwischen den Strings `mass_margin_bean` und `marginBeanMass`. Die beiden Strings sollten zueinander eine Ähnlichkeit von 1 aufweisen, aufgrund der Anordnung der Wörter und der speziellen Zeichen tritt dies jedoch nie auf. Wenn nun die Sonderzeichen entfernt werden und die Wörter nach ihrer Ähnlichkeit zu den Wörtern des anderen Strings angeordnet werden, entsteht bei beiden derselbe String und die Ähnlichkeit ist optimal, also 1.
Nun wird für jedes Paar von Strings die Ähnlichkeit mit allen drei Ähnlichkeitsmaßalgorithmen berechnet. Je nach kompositionalen Verfahren werden die Werte unterschiedlich für das Endergebnis gewichtet und kombiniert. In dieser Software gibt es sechs verschiedene Verfahren:

- Jaro-Winkler → gibt nur die Jaro-Winkler Ähnlichkeit zurück
- Jaccard → gibt nur den Jaccard Ähnlichkeit zurück
- normalisierter Levenshtein → gibt nur die normalisierte Levenshtein Ähnlichkeit zurück
- Durchschnittsähnlichkeit → gibt den Durchschnitt der drei Algorithmen zurück

- Minimum → gibt die kleinste Ähnlichkeit zurück
- Maximum → gibt die größte Ähnlichkeit zurück

Die Ähnlichkeiten der Strings werden für alle Verfahren parallel bestimmt.
Die berechneten Ähnlichkeitswerte werden für jedes Verfahren in einem einzelnen **Min-Heap** gespeichert. Ein Min-Heap nimmt dabei immer nur die drei Parameter mit den höchsten Werten an. Sobald alle Eingabeparameter mit einem Ausgabeparameter verglichen wurden, befinden sich im Min-Heap die ähnlichsten null bis drei Eingabeparameter zu einem Ausgabeparameter. Diese werden als Matching-Vorschläge mit dem Ausgabeparameter in einer Liste gespeichert. Jedes Verfahren bestimmt so seine eigene Liste von Matching-Vorschlägen. Wenn sich im Min-Heap keine Einträge befinden, wird **noSimilarity** zurückgeben. Dies wird für alle Ausgabeparameter durchgeführt und danach werden die Listen mit den Matching-Vorschlägen zurück gegeben. Der Nutzer kann in der UI dann auswählen, nach welchen Verfahren dieser die Vorschläge für das Matching angezeigt bekommen soll.

4.4 Ergebnisse des Matchings im Test

Beim Aufbau der Matching-Software (Kapitel 4.3.1) wurden die **Virtual Satellite API** und **Part Database API** erwähnt. Die **Part Database** ist eine Datenbank, die Produkte mit Parametern von verschiedenen Herstellern speichert (z.B. Batterien, Star Tracker, ...) und **Virtual Satellite** ist eine Software zur Modellierung von Satelliten. Virtual Satellite benötigt dazu verschiedene Produkte, deren Daten aus der Part Database entnommen werden. Diese beiden APIs wurden mit der UI gekoppelt, um den Datenaustausch mit der Matching-Software zwischen den beiden APIs zu testen. Für den Test wurden verschiedene Nutzer befragt, wie sie die Part Database Parameter zu den Virtual Satellite Parametern matchen würden und dann mit den Vorschlägen verglichen, die die Matching-API dafür berechnet. Der Test fand dabei nicht mit bestimmten Produkten, sondern nur mit den Parametern der APIs statt.
Das ist die Liste der Parameter beider APIs:

Virtual Satellite Parameter	Part Database Parameter
activeUnits	command_interface
colorBean	data_output_interface
geometryFileBean	data_sheet
marginMaturityBean	input_voltage_max
massPerUnit	input_voltage_min
massTotal	lifetime
massTotalWithMargin	linearity
positionXBean	mass
positionYBean	mechanical_vibration
positionZBean	output_current_max
powerAvgWithMarginCommunication	output_current_min
powerAvgWithMarginIdle	output_power_max
powerAvgWithMarginScience	output_power_min
powerDutyCycleCommunication	output_voltage_max
powerDutyCycleIdle	output_voltage_min
powerDutyCycleScience	power_on
powerPerUnitAvgWithMarginCommunication	power_standby
powerPerUnitAvgWithMarginIdle	diameter
powerPerUnitAvgWithMarginScience	height
powerPerUnitOn	length
powerPerUnitOnWithMargin	width
powerPerUnitStby	stl_file
powerPerUnitStbyWithMargin	temperature_no_ops_max
radiusBean	temperature_no_ops_min
rotationXBean	temperature_ops_max
rotationYBean	temperature_ops_min
rotationZBean	
shapeBean	
sizeXBean	
sizeYBean	
sizeZBean	
temperatureNoOpsMax	
temperatureNoOpsMin	
temperatureOpsMax	
temperatureOpsMin	
transparencyBean	
unitQuantityBean	

Tabelle 1: Parameter Virtual Satellite und Part Database

In der Tabelle 1 stehen alle für das Matching verfügbaren Parameter der beiden APIs (keine Zusammenhänge zwischen nebeneinander stehenden Parametern in der Tabelle). Grundsätzlich besitzen die APIs noch mehr Parameter, die aber keine aktuelle Verwendung besitzen.
Beim Test stellte sich heraus, dass weitere Parameter von Virtual Satellite für das Matching ungeeignet sind. Dazu müssen die Parameter von Virtual Satellite zunächst unterschiedenen werden. Es gibt sowohl Parameter, denen Werte zugeordnet werden können, als auch Parameter, die abhängig von anderen Parametern direkt in Virtual Satellite berechnet werden. Diese berechenbaren Parameter entfallen für das Matching. Außerdem sind die Parameter hinfällig, die direkt in Virtual Satellite anhand des Verwendungszwecks (z.B. abhängig vom Design in der Visualisierung) festgelegt werden. Das zeigt

die folgenden Tabelle, welche das Matching der Parameter durch die Nutzer zusammenfasst:

Virtual Satellite Parameter	Nutzer Matching
geometryFileBean	stl_file
massPerUnit	mass
powerPerUnitOn	power_on
powerPerUnitStby	power_standby
radiusBean	diameter/2
sizeXBean	height
sizeYBean	length
sizeZBean	width
temperatureNoOpsMax	temperature_no_ops_max
temperatureNoOpsMin	temperature_no_ops_min
temperatureOpsMax	temperature_ops_max
temperatureOpsMin	temperature_ops_min

Tabelle 2: Parameter-Matching der Nutzer

Die Tabelle 2 zeigt die matchbaren Virtual Satellite Parameter und wie die Nutzer die verfügbaren Part Database Parameter zuordnen würden. Der Radius (radiusBean) kann jedoch nicht direkt zugeordnet werden, da in der Part Database nur der Durchmesser (diameter) mitgegeben wird. Der Durchmesser wird deshalb mit der Formel zur Umrechnung in den Radius gematcht. Im aktuellen Zustand kann die Matching-Software nur 1:1 Zuordnungen als Vorschläge liefern (keine Formeln), deshalb ist **radiusBean** in dem folgenden Test der Matching-API kein vergleichbarer Parameter. Er wird jedoch mit einbezogen, um generell zu sehen, welcher Vorschlag für das Matching geliefert wird. Außerdem bestand beim Test Uneinigkeit bei der Zuordnung zu den Parametern **sizeXBean**, **sizeYBean** und **sizeZBean**. Die drei Parameter sind Längeneinheiten und liefern keine Information für die richtige Zuordnung von **height**, **length** und **width** mit.
Im Folgenden sind die Matching-Vorschläge mit den höchsten berechneten Ähnlichkeitswerten der Matching-API mit den verschiedenen Matching-Verfahren dargestellt. Zusätzlich wird eine Ontologie für Synonyme der Parameter in der Part Database verwendet (siehe Erklärung der Verwendung von Ontologien in 4.3.2 und 4.3.3):

	Jaro-Winkler	Jaccard	norm. Levenshtein
geometryFileBean	command_interface	command_interface	command_interface
massPerUnit	mass	mass	mass
powerPerUnitOn	power_on	power_on	power_on
powerPerUnitStby	power_standby	power_standby	power_standby
radiusBean	diameter	length	length
sizeXBean	diameter	width	width
sizeYBean	diameter	width	width
sizeZBean	diameter	width	width
temperatureNoOpsMax	temperature_no_ops_max	temperature_no_ops_max	temperature_no_ops_max
temperatureNoOpsMin	temperature_no_ops_min	temperature_no_ops_min	temperature_no_ops_min
temperatureOpsMax	temperature_ops_max	temperature_ops_max	temperature_ops_max
temperatureOpsMin	temperature_ops_min	temperature_ops_min	temperature_ops_min

Tabelle 3: Die besten Matchingvorschläge der Ähnlichkeitsmaßalgorithmen

In der Tabelle 3 sind die berechneten Matching-Vorschläge der verwendeten Ähnlichkeits-

maßalgorithmen einzeln dargestellt. Auffällig ist, dass Zuordnungen zu den Parametern massPerUnit, powerPerUnitOn, powerPerUnitStby, temperatureNoOpsMax, temperatureNoOpsMin, temperatureOpsMax und temperatureOpsMin zwischen den Ähnlichkeitsalgorithmen gleich sind und außerdem den Nutzer-Zuordnungen entsprechen. Die übereinstimmende Zuordnung beruht dabei auf der ähnlichen Benennung der Parameter in der Part Database bzw. den ähnlichen Synonymen in der verwendeten Ontologie und der optimalen Anordnung der Namen für die Ähnlichkeitsberechnung. Die Parameter geometryFileBean, radiusBean, sizeXBean, sizeYBean, sizeZBean sind nicht optimal benannt und die Synonyme liefern keine Information um die Parameter optimal zuzuordnen. Das ist stark bei den Parametern **sizexBean**, **sizeYBean** und **sizeZBean** zu bemerken, die sehr ähnlich benannt sind. Der Jaccard und normalisierte Levenshtein Algorithmus haben dieselben Matching-Vorschläge berechnet. Alle drei Algorithmen haben vier falsche Matching-Vorschläge zurück gegeben, somit kann man aus diesem Test nicht herauslesen, welcher Algorithmus bessere Vorschläge berechnet.
Die folgenden Tabelle zeigt die Matching-Vorschläge mit den höchsten berechneten Ähnlichkeitswerten, die durch drei kompositionale Matching-Verfahren berechnet wurden (erklärt in Kapitel 4.3.3):

	Durchschnittsähnlichkeit	Minimum	Maximum
geometryFileBean	command_interface	command_interface	command_interface
massPerUnit	mass	mass	mass
powerPerUnitOn	power_on	power_on	power_on
powerPerUnitStby	power_standby	power_standby	power_standby
radiusBean	diameter	length	diameter
sizeXBean	diameter	width	diameter
sizeYBean	diameter	width	diameter
sizeZBean	diameter	width	diameter
temperatureNoOpsMax	temperature_no_ops_max	temperature_no_ops_max	temperature_no_ops_max
temperatureNoOpsMin	temperature_no_ops_min	temperature_no_ops_min	temperature_no_ops_min
temperatureOpsMax	temperature_ops_max	temperature_ops_max	temperature_ops_max
temperatureOpsMin	temperature_ops_min	temperature_ops_min	temperature_ops_min

Tabelle 4: Die besten Matchingvorschläge der kompositionalen Matching-Verfahren

Das Verfahren zur Berechnung der Durchschnittsähnlichkeit und das des Maximums haben dieselben Ergebnisse hervorgebracht. Das Verfahren für das Minimum liefert die selben Ergebnisse wie der Jaccard und normalisierte Levenshtein Algorithmus. Alle drei Verfahren liefern teilweise unterschiedliche Ergebnisse, besitzen aber dieselbe Fehlerrate, welche beim Jaccard und normalisierten Levenshtein Algorithmus bei drei falschen Matching-Vorschlägen liegt.

Der Test der Matching-API ist natürlich nur sehr klein, da wenig Parameter zur Verfügung standen, die wirklich miteinander gematcht werden konnten. Zusätzlich können Parameter ungenau benannt sein und Hintergrundinformation könnten fehlen, weshalb die Matching-API bei manchen Parametern nur schwer optimale Vorschläge liefern kann. An den vergleichbaren Parametern ist jedoch festzustellen, dass die optimale Anordnung der Parameternamen, das Einbeziehen der Datentypen und die Ähnlichkeitsmaßalgorithmen in Kombination bezogen auf diesen Beispieldatenaustausch gute Ergebnisse liefern. Es muss dazu erwähnt werden, dass eine Ontologie für Synonyme

mit einbezogen wurde. Diese Ontologie liefert durchschnittlich elf Synonyme für einen Parameter in der Part Database und somit zusätzliche Informationen für das Matching. Es wurde nicht direkt der Einfluss der Synonyme auf das Endergebnis gemessen, aber es ist anzunehmen, dass die Synonyme zu korrekteren Ergebnissen beigetragen haben. Auch die aktuell eingebunden kompositionalen Verfahren haben gute Ergebnisse berechnet. Alle drei Verfahren haben insgesamt vier falsche Ergebnisse geliefert und wichten somit gut die Ergebnisse der Ähnlichkeitsmaßalgorithmen. Das Beispiel kann jedoch schlecht gewählt sein, weshalb die kompositionalen Verfahren keine besseren Ergebnisse liefern, oder es müssen andere Verfahren verwendet werden (z.B. gewichtetes Matching, welches dynamisch die Wichtung der Werte anpasst).
In diesem Test wird nur gezeigt, wie gut die Ergebnisse mit den höchsten berechneten Ähnlichkeiten der Matching-API sind. Die Matching-API liefert dem Nutzer für das Matching jedoch bis zu drei Vorschläge, was stark die Wahrscheinlichkeit erhöht, dass dem Nutzer die Parameter vorgeschlagen werden, die dieser selber beim Matching zuordnen würde. Außerdem werden mit jedem fertigen Matching, die gematchten Begriffe in einer Ontologie gespeichert und die Matching-API kann diese zusätzliche Information nutzen, um die Matching-Vorschläge stetig zu verbessern.

5 Zusammenfassung und Ausblick

Das Ziel der Arbeit war es, eine nutzergestützte Software zu entwickeln, die mithilfe von schematischem Matching den Datenaustausch zwischen verschiedenen APIs ermöglicht. Aufgrund des Problems und auch gewünschter Eigenschaften der Software entstand das Konzept mit verschiedenen Aspekten, die das Endprodukt enthalten sollte. Dabei sollten letztendlich eine webbasierte UI und eine Matching API entstehen, die jeweils miteinander kommunizieren und den Nutzer beim Matching von Daten unterstützen. In der Matching-Software wurden alle Konpzeptideen eingearbeitet – insbesondere wurde im Laufe der Entwicklung keine dieser Ideen verworfen. Mithilfe von Nutzertests konnte festgestellt werden, dass alle Konzeptideen einen sinnvollen Einsatz im Endprodukt fanden.
Die UI bietet eine übersichtliche Oberfläche mit allen grundlegenden Funktionen, um ein Matching mit den beiden Schritten Produktauswahl und Parametermatching durchführen zu können (siehe Abbildungen 21, 22, 23 und 24).
In der Matching-API wurden Ähnlichkeitsmaßalgorithmen zu hybriden und kompositionalen Matching-Verfahren kombiniert, um anhand der Daten der zu matchenden Objekte Matching-Vorschläge zu berechnen. Die Nutzertests haben ergeben, dass die Algorithmen und Verfahren sinnvolle bzw. erwartete Vorschläge berechnen und so dem Nutzer das Matching vereinfachen können.
Die Einbindung von Ontologien bietet die Möglichkeit, bessere Matching-Vorschläge zu liefern. Außerdem werden die durch den Nutzer gematchten Objekte bzw. deren Parameternamen gespeichert, um stetig bessere Vorschläge zu berechnen.

Die Matching-Software ist jedoch noch nicht vollkommen ausgereift. Im Folgenden werden mögliche Verbesserungen für diese aufgelistet:
In der UI gibt es Aspekte, die eine noch übersichtlichere Oberfläche und bessere Nutzbarkeit bieten könnten. Dazu zählen Tabellen, die zusätzliche Information für die Objekte liefern oder auch ein einfacheres und übersichtliches Filtern. Weitere Aspekte sind eher kosmetischer Art und beeinflussen nicht die Qualität der Ergebnisse.
Die Matching-API zeigt auch einige Verbesserungsmöglichkeiten auf. Bei der Ähnlichkeitsberechnung können noch andere simple Algorithmen, etwa weitere Ähnlichkeitsmaßalgorithmen, eingebaut werden. Außerdem könnte die Integration weiterer hybrider und kompositionaler Matching-Verfahren mit einer besseren Wichtung eine hochwertigere Ähnlichkeit gegenüber der aktuellen Algorithmen zurückgeben. Ein kompositionales Verfahren, welches eingebaut werden sollte, ist das gewichtete Matching, welches anhand der Eingabedaten die Gewichte der einzelnen Algorithmen anpasst. Die durch den Nutzer durchgeführten Matchings sollten außerdem genutzt werden, um Vor- und Nachteile der einzelnen Ähnlichkeitsmaßalgorithmen bei bestimmten Strings zu sammeln. Durch diese Information können die Gewichte der Algorithmen in den kompositionalen Verfahren besser angepasst werden.

6 Literaturverzeichnis

[1] Chris Achard. *React vs. Plain JavaScript*. URL: https://www.framer.com/blog/posts/react-vs-vanilla-js/ (besucht am 23. 02. 2021).

[2] Hong-Hai Do und Erhard Rahm. "COMA—a system for flexible combination of schema matching approaches". In: *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier. 2002, S. 610–621.

[3] Matthias Düsel und Ute Schmid. "Semantisches Matching von Freizeitaktivitäten mittels Wikipediabasierter Kategorisierung". Masterarbeit. Otto-Friedrich-Universität Bamberg, 2013.

[4] Wageningen UR/Food & Biobased Research Consumer Science & Intelligent Systems & COMMIT / eFoodLab. *Foodvoc*. URL: http://www.ontology-of-units-of-measure.org/resource/om-2/ (besucht am 24. 03. 2021).

[5] Jérôme Euzenat und Pavel Shvaiko. *Ontology Matching: Second Edition*. 2013. ISBN: 978-3-642-38720-3. DOI: 10.1007/978-3-642-38721-0.

[6] Tobias Fechner. "Hybrides Schema Matching mit COMA". Hausarbeit. Universität Hamburg.

[7] Apache Software Foundation. *Apache Jena*. URL: https://jena.apache.org/ (besucht am 23. 03. 2021).

[8] Apache Software Foundation. *What is Maven?* URL: https://maven.apache.org/what-is-maven.html (besucht am 23. 03. 2021).

[9] Google. *Gson*. URL: https://github.com/google/gson (besucht am 23. 03. 2021).

[10] Facebook Inc. *React*. URL: https://reactjs.org/ (besucht am 23. 03. 2021).

[11] *JSON: The Fat-Free Alternative to XML*. URL: http://www.json.org/xml.html (besucht am 23. 03. 2021).

[12] Suphakit Niwattanakul u. a. "Using of Jaccard coefficient for keywords similarity". In: *Proceedings of the international multiconference of engineers and computer scientists*. Bd. 1. 6. 2013, S. 380–384.

[13] Oracle. *Erfahren Sie mehr über die Java-Technologie*. URL: https://www.java.com/de/about/ (besucht am 24. 03. 2021).

[14] Erhard Rahm und Philip A Bernstein. "On matching schemas automatically". In: *VLDB journal* 10.4 (2001), S. 334–350.

[15] Ralph Allan Rice. *Java-String-Similarity*. URL: https://github.com/rrice/java-string-similarity#:~:text=java%2Dstring%2Dsimilarity%20that%20calculates,each%20the%20two%20strings%20are. (besucht am 23. 03. 2021).

[16] Oleksii Trekhleb. *Dynamic Programming vs Divide-and-Conquer*. URL: https://itnext.io/dynamic-programming-vs-divide-and-conquer-2fea680becbe (besucht am 23. 03. 2021).

[17] uni-leipzig samass uni-leipzig. *COMA Community Edition*. URL: https://sourceforge.net/projects/coma-ce/ (besucht am 23. 03. 2021).

[18] Inc. VMware. *Spring Boot*. URL: https://spring.io/projects/spring-boot (besucht am 23. 03. 2021).

[19] William E. Winkler. *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage [microform] / William E. Winkler*. English. Distributed by ERIC Clearinghouse [Washington, D.C.], 1990, 8 p. URL: https://eric.ed.gov/?id=ED325505.

[20] Li Yujian und Liu Bo. “A normalized Levenshtein distance metric”. In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), S. 1091–1095.

7 Anlagen

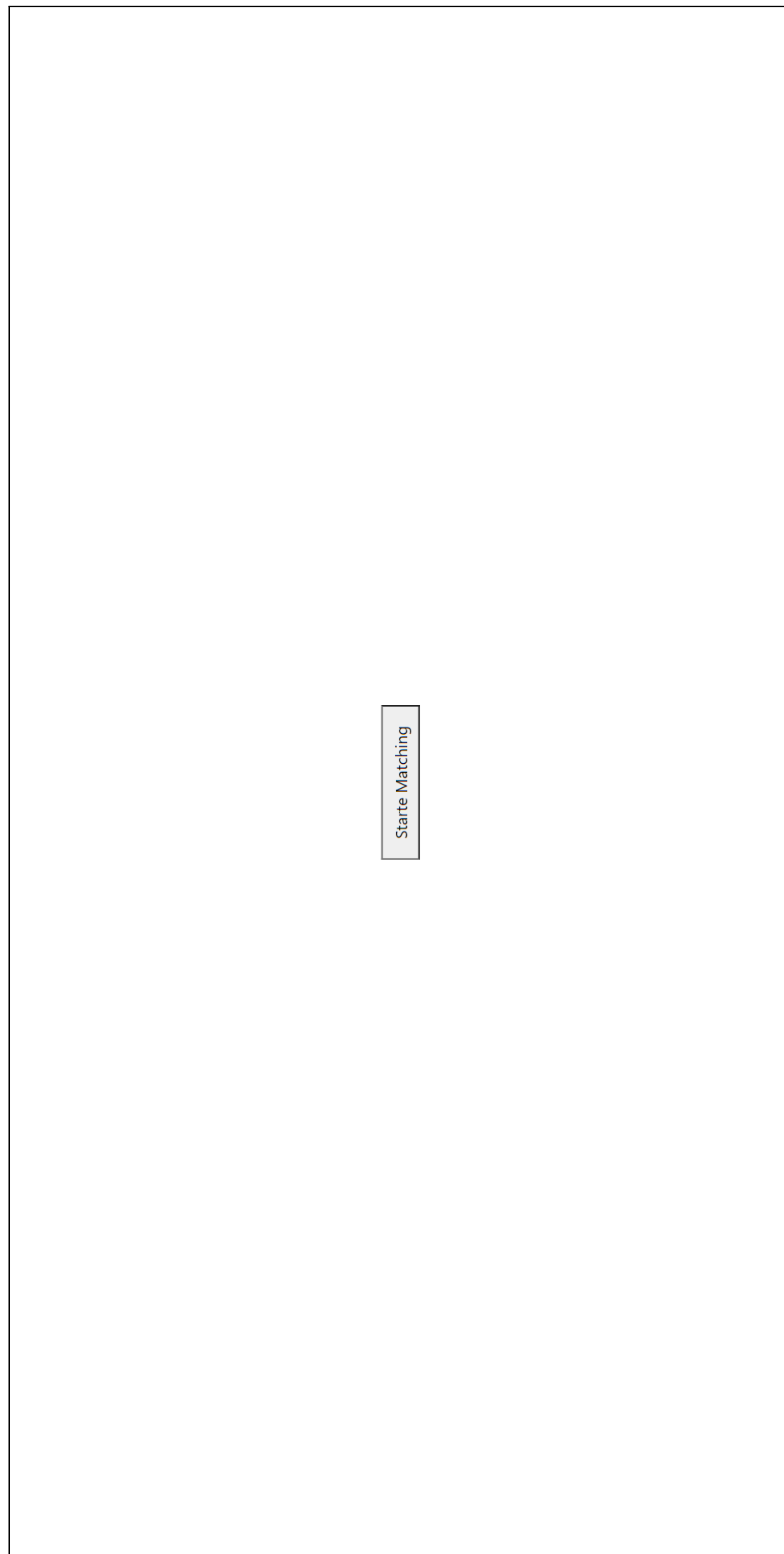

Abbildung 21: Startansicht

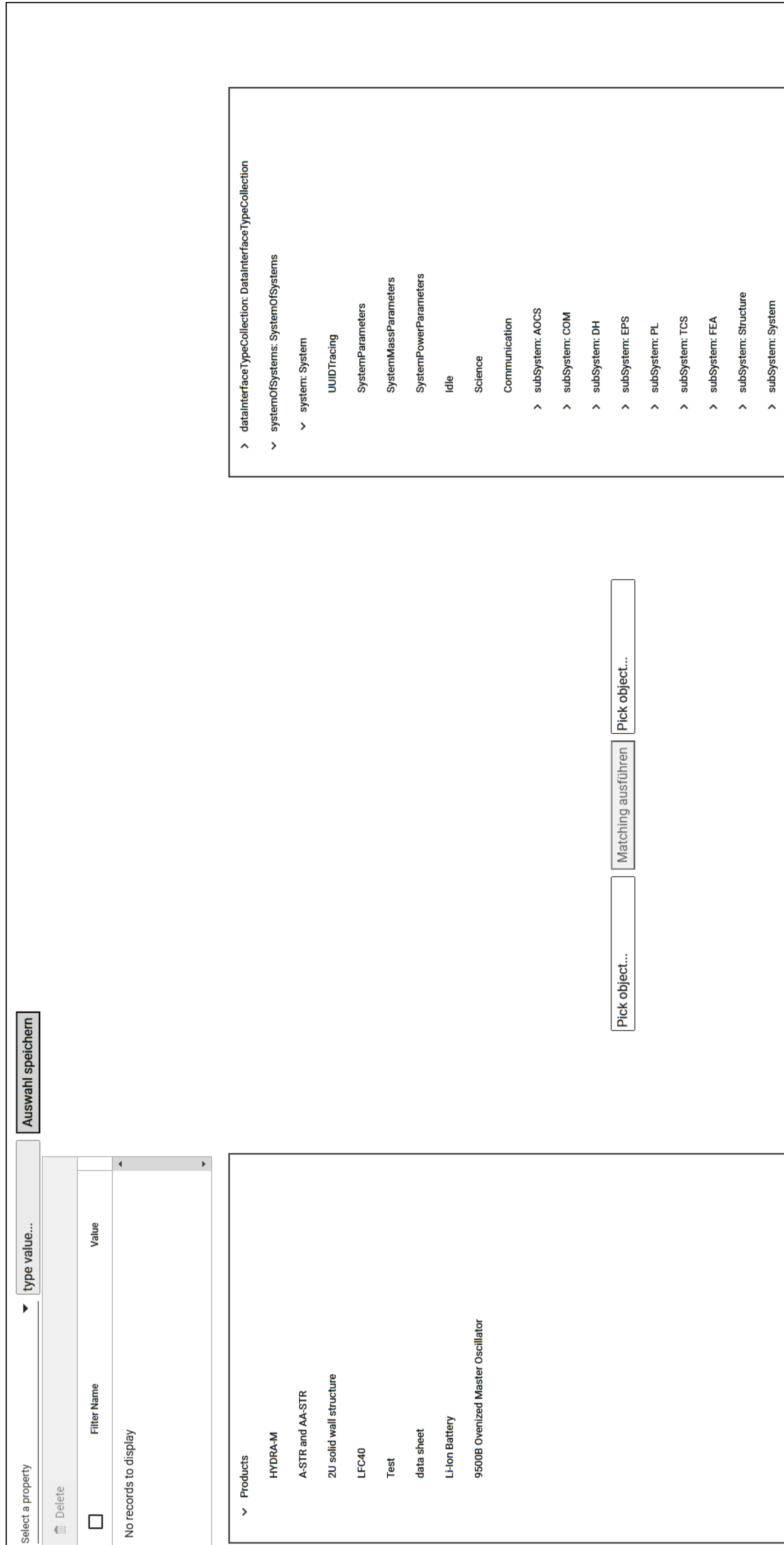

Abbildung 22: Beispielansicht der Objektauswahl

9500B Ovenized Master Oscillator

Filter Name	Value
command_interface	
data_output_interface	
data_sheet	
lifetime	
linearity	
mass	2.4
mechanical_vibration	
output_current.output_current_min	
output_current.output_current_max	
output_power.output_power_min	
output_power.output_power_max	
output_voltage.output_voltage_min	
output_voltage.output_voltage_max	
power.power_standby	
power.power_on	3
size.width	98.3
size.length	227.4
size.height	83.1
size.diameter	
stl_file	
temperature_no_ops.temperature_no_ops_min	-40
temperature_no_ops.temperature_no_ops_max	100
temperature_ops.temperature_ops_min	-24
temperature_ops.temperature_ops_max	60

Filter Name	Value	
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X
		X

RW

Filter Name	Value	
marginMaturityBean	20	?
unitQuantityBean	4	?
massPerUnit	0.22	?
massTotal	0.88	?
massTotalWithMargin	1.056	?
activeUnits	4	?
powerAvgWithMarginIdle	0	?
powerAvgWithMarginScience	34.56	?
powerAvgWithMarginCommunication	19.2	?
powerDutyCycleIdle	0	?
powerDutyCycleScience	90	?
powerDutyCycleCommunication	50	?
powerPerUnitAvgWithMarginIdle	0	?
powerPerUnitAvgWithMarginScience	8.64	?
powerPerUnitAvgWithMarginCommunication	4.8	?
powerPerUnitOn	8	?
powerPerUnitOnWithMargin	9.6	?
powerPerUnitStby	0	?
powerPerUnitStbyWithMargin	0	?
colorBean	65535	?
geometryFileBean		?
positionXBean	0	?
positionYBean	0	?
positionZBean	0	?
radiusBean	0	?
rotationXBean	0	?
rotationYBean	0	?
rotationZBean	0	?
shapeBean		?
sizeXBean	57	?
sizeYBean	57	?

Save matching

Abbildung 23: Beispielansicht der Matching-Ansicht

Abbildung 24: Beispielansicht des Vorschläge-Pop-Ups

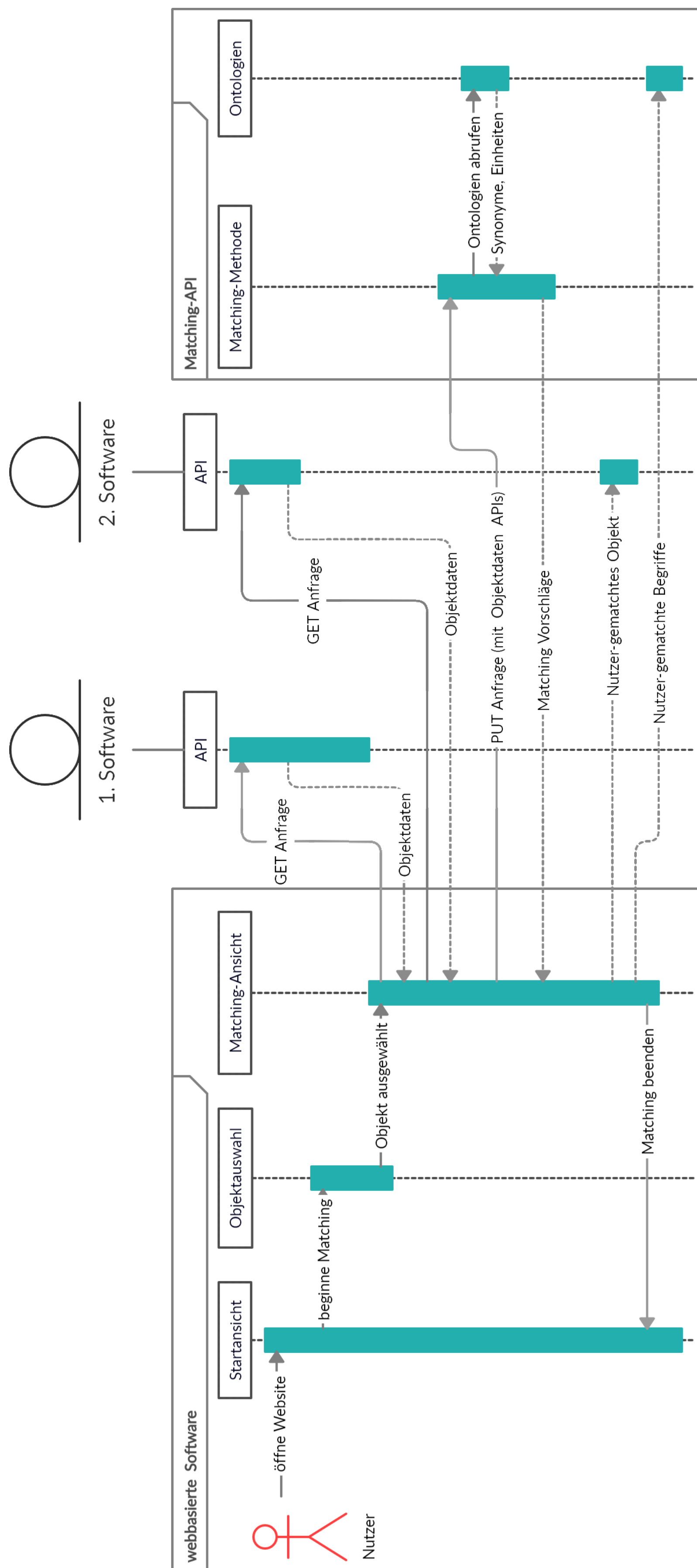

Abbildung 25: Umsetzung des Konzepts der Matching-Software

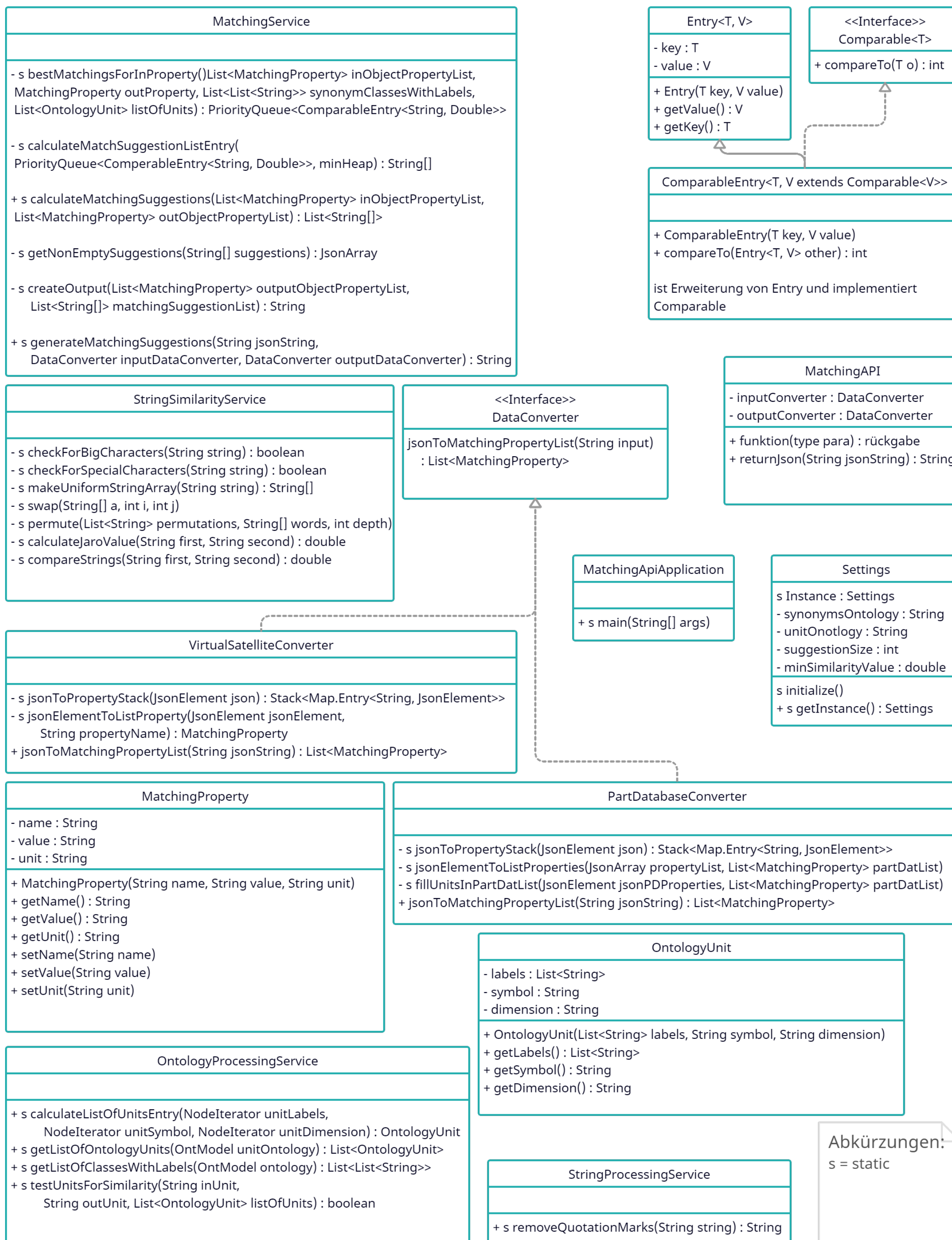

Abbildung 26: Klassendiagramm Matching API

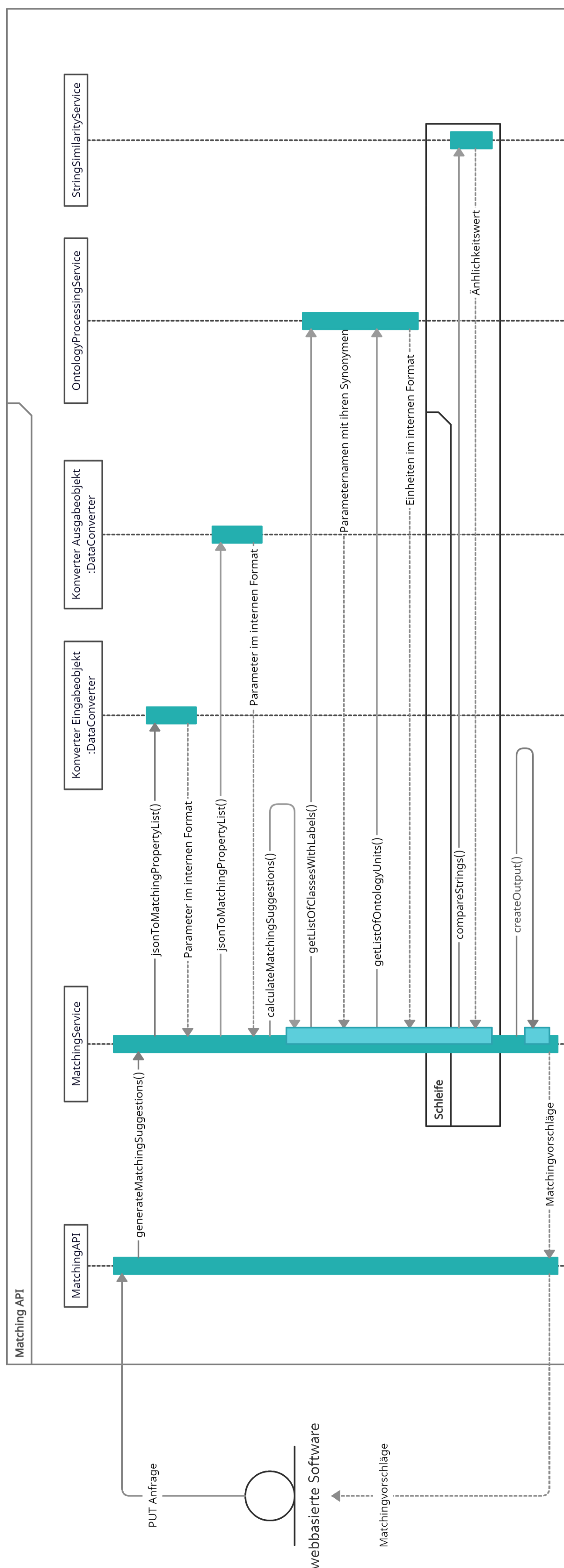

Abbildung 27: Sequenzdiagramm Matching API

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

____

Jena, 24. März 2021