Automatisches Auszeichnen mit *cueML*

CRF-based Prototyp

Dictionary- and rule-based Prototyp

Inspiriert von dem vorherigen Abschnitt Information Extraction in der Koch-Domäne (IEKochDomain.html) stellen wir hier zwei Prototypen vor, um aus den Zubereitungs-Anweisungen von unserem Kochbuch eine Zutatenliste zu extrahieren. Da wir auf keine vorhandene Semi-Struktur aufbauen können, ist ein Regular Expression-based Ansatz nicht möglich. Wir implementieren einen Conditional Random Field-based (CRF-based) und einen dictionary- and rule-based Prototypen. Der CRF-based Prototyp ist sehr simpel. Uns ist bereits während der Konstruktion klar geworden, dass wir andere Voraussetzungen als (Greene, 2015) (Quellen.html#CRFNYT) haben und dies daher nicht der richtige Ansatz für uns ist.

CRF-based Prototyp

Unser CRF-based Prototyp ist in unserem Git-Repository unter dem Tag CRF-BasedPrototypeV0.1 (https://git.informatik.uni-kiel.de/stu121256/Torsten_Knauf_Master-Arbeit_Davidises_Kochbuch_Source_Code/tree/CRF-BasedPrototypeV0.1) zu finden. Er verwendet das SWIG (http://www.swig.org/) Interface für *Python 3* von CRFsuite 0.12 (http://www.chokkan.org/software/crfsuite/).

Wie in CRF erklärt (IEKochDomain.html#CRFTheorie), muss der Prototyp zuerst antrainiert werden. Abb. 1 zeigt einen kleinen Ausschnitt unserer Trainingsdaten für den CRF-based Prototypen. Nebenbei sei erwähnt, dass wir diese Anweisungen heute als eine Mehlschwitze bezeichnen würden.

Aus diesem kleinen Ausschnitt der Trainingsdaten sind bereits viele Problematiken ersichtlich, was später in der Evaluierung dieses Prototypen (Evaluierung.html#CRFPrototyp) genauer erläutert wird. Beispielsweise wird aus den Labels nicht ersichtlich, dass die Mengenanagbe „eine“ zu der Zutat „Zwiebel“ gehört. Daher haben wir zunächst einen simplen Prototypen gebaut, um zu schauen, ob CRF wirklich der richtige algorithmische Ansatz für unser Problem ist. Wie in Abb. 1 haben wir 10 Rezepte gelabelt. Mit 9 (images/AutomatischesAnreichrenCueML/Trainings-Rezepte.txt) davon haben wir den CRF-based Prototypen trainiert und an dem 10. (images/AutomatischesAnreichrenCueML/Test-Rezept.txt) getestet. Als features verwenden wir ausschließlich die Wort-Identität, sowie den Übergang von zwei benachbarten Labels (B-Template). Für die Gewichte der features erlauben wir auch negative Werte.

Abb. 1: Ausschnitt unserer CRF Trainingsdaten

```
Man             O
schmort         O
eine            B-Quantity
fein            O
geschnittene    O
Zwiebel         B-Ingredient
mit             O
reichlich       O
frischer        O
Butter          B-Ingredient
,               O
läßt            O
ein             B-Quantity
bis             I-Quantity
zwei            I-Quantity
Eßlöffel        B-Unit
voll            O
feines          O
Mehl            B-Ingredient
darin           O
anziehen        O
,               O
und             O
[...]
```

Dictionary- and rule-based Prototyp

Nachdem wir zu der Erkenntnis gekommen sind, dass der vorherige Ansatz für diese Arbeit nicht zielführend ist (Evaluierung.html#CRFPrototyp), stellen wir hier einen dictionary- and rule-based Prototypen vor. Unsere Lemmatisierung für den dictionary-based Teil baut auf treetagger Version 1.0.1 (https://github.com/miotto/treetagger-python) auf.

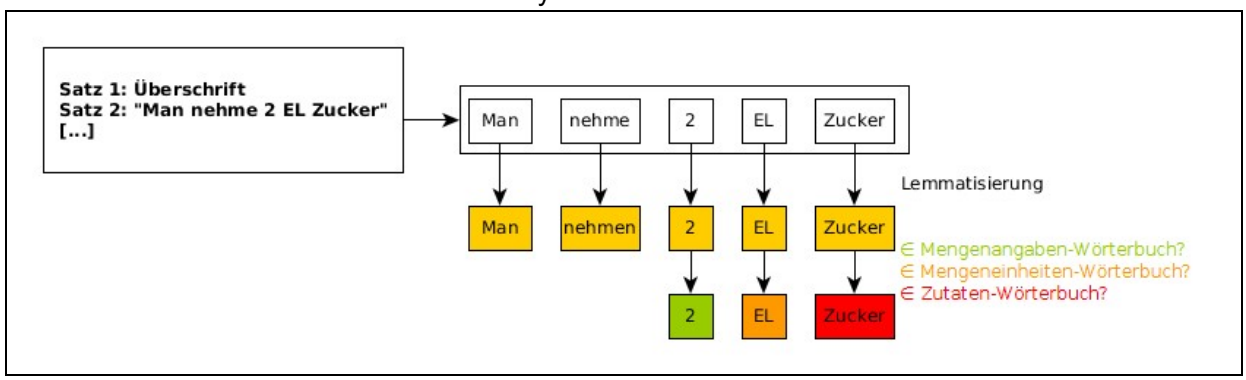
Version 0.1

Ein erster Prototyp ist in unserem Git-Repository unter dem Tag Dict-AndRule-BasedPrototypeV0.1 (https://git.informatik.uni-kiel.de/stu121256/Torsten_Knauf_Master-Arbeit_Davidises_Kochbuch_Source_Code/tree/Dict-AndRule-BasedPrototypeV0.1) zu finden. Das Ziel dieses Prototypen ist es, erst einmal ausschließlich zu testen, ob das dictionary-based Extrahieren unserer gesuchten Entities (Zutaten, Mengenangaben und Einheiten) funktioniert. Deswegen ist er nach dem Motto *quick and dirty* implementiert. Nachdem die erste Evaluierung (Evaluierung.html#DictBasedV01) positiv stimmt, haben wir den Prototypen quasi weggeworfen und in der folgenden Version 0.2 redesigned, wie es sich für einen quick and dirty Prototypen gehört. Daher wird hier auch nicht weiter auf Implementierung-Details eingegangen sondern nur auf konzeptionelle Ideen.

Abb. 2 zeigt den nahezu identischen Ablauf der dictionary-based Extraktion aus der vorherigen Sektion (IEKochDomain.html#DictBased). Die Überschrift eines Rezeptes wird als normaler Satz gesehen. Sie muss bei der Extraktion berücksichtigt werden, da manchmal eine Zutat nur in dieser

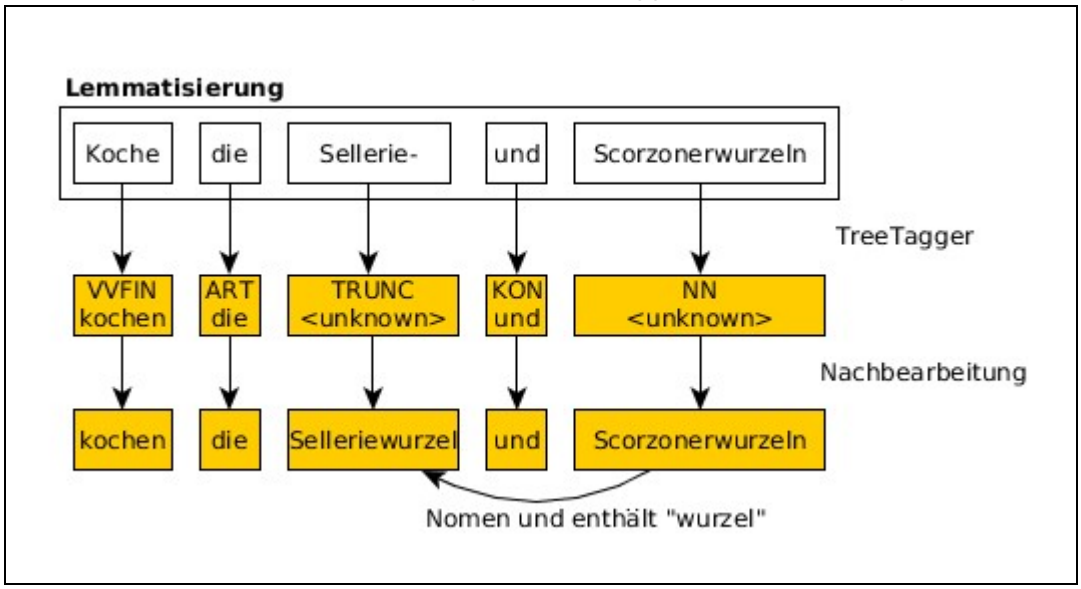
erwähnt wird. Die Sätze splitten wir anhand von Satzzeichen. Um beispielsweise aus *Nach Nro. 2 wird [...]* nicht zwei Sätze zu machen, haben wir eine Menge von Abkürzungen definiert, bei denen der Split wieder zusammengeführt wird.

Abb. 2: Dictionary-based Extraktion der Entities



In Abb. 3 ist in einem ersten Schritt der Output eines Satzes von TreeTagger zu sehen. Neben dem Lemma gibt er zusätzlich ein Part-of-Speech-Tag (PoS) zu jedem Wort aus. Wie bei *Sellerie-* und *Scorzonerwurzeln* kann TreeTagger allerdings manchmal kein Lemma finden. Deswegen nehmen wir zuerst von jedem Wort ohne Lemma die Wort-Identität als Lemma. Anschließend versuchen wir bei jedem abgekürzten Wort die Endung zu finden. Abgekürzte Wörter haben das PoS TRUNC für truncation. Da alle abgekürzten Wörter, an denen wir interessiert sind, Zutaten und somit Nomen sind, suchen wir das nächste Nomen im Satz, welches eine typisch abgekürzte Endung hat. Ein Beispiel für so eine Endung ist *wurzel*. Ist dies der Fall, wird in dem abgekürzten Lemma der Bindestrich durch die entsprechende Endung ersetzt.

Abb. 3: Nachbearbeitung von TreeTaggers Lemmatisierung



Was jetzt noch fehlt, sind die einzelnen Wörterbücher der zu extrahierenden Entities:

- Das Wörterbuch für **Mengeneinheiten** erstellen wir manuell. Es enthält beispielsweise *Pfund* und *Maß*.
- Für die **Mengenangaben** verwenden wir kein echtes Wörterbuch sondern eine Funktion. Diese bekommt als Argument ein Lemma und returniert in folgenden drei Fällen *True*:
    - Das Lemma ist eine Zahl.
    - Das Lemma hat die Form *Zahl-Zahl*.
    - Das Lemma ist in einer Menge von vordefinierter Mengen-Wörter *(ein, eine, einige, ...)*.

Ansonsten returniert die Funktion *False*.

- Wie in Domänen-spezifisches Vokabular cueML (cueML.html#cueML) erläutert, haben wir alle ausgezeichneten Zutaten mittels einer Liste von *ingredient*-Elementen an den BLS angeschlossen. Unten stehende Abb. 4 zeigt, wie wir aus diesen ein **Wörterbuch von Zutaten** extrahieren. Jedes Nomen einer Basisform ist ein Eintrag im Wörterbuch. Zu jedem Eintrag speichern wir zusätzlich eine Liste von möglichen xml:ids ab. Wein könnte z. B. Weißwein wie auch Rotwein sein. Bei einer rule-based Weiterverarbeitung kann aus der Liste der möglichen *xml:id*s ggf. eine eindeutige *xml:id* bestimmt werden. Dies werden wir in der Version 0.2 in Angriff nehmen. Der letzte Eintrag in Abb. 4b ist besonders. Frau Davidis verwendet oft Fleisch als Zutat und aus dem Kontext wird ersichtlich, was für ein Fleisch gemeint ist. Bei einer Rindfleischsuppe zum Beispiel ist mit Fleisch ziemlich sicher Rindkochfleisch gemeint. Um auch das Lemma Fleisch als Zutat extrahieren zu können, fügen wir dieses dem Wörterbuch hinzu. Der hinterlegte Wert ist eine Liste von allen xml:ids, in dessen Einträgen das Wort Fleisch vorkommt.

  Die Einschränkung auf das Nomen einer Zutat ist eine große Beschleunigung der Laufzeit der dictionary-based Extraktion. Bei einem Satz der Länge *n* werden so *n look ups* im Wörterbuch benötigt. Um Zutaten der Länge eins und zwei nachschlagen zu können, wären bereits *n * (n-1) look ups* nötig. Problematisch wären auch Fälle, in denen die Wörter einer Zutat nicht benachbart sind wie z. B. bei *Man nehme Wein; bevorzugt Weißen*.

  Wenn ein Lemma nicht im Zutaten-Wörterbuch ist, überprüfen wir zusätzlich, ob es eine zusammengesetzte Zutat ist, bzw. ein eigenes Rezept. Ist im Wort eins der Wörter *bouillon, klöße oder kloß* enthalten, ist dies auch ein positives Ergebnis und es wird eine leere Liste von möglichen BLS zurückgegeben.

Abb. 4: Extraktion eines Zutaten-Wörterbuches aus den *cueML ingredient*-Elementen

Abb. 4a: Beispielhafter Aussschnitt von cueML ingredient-Elementen

```
<cue:listIngredient xmlns="http://www.tei-c.org/ns/1.0" xmlns:cue="http://cueML/ns">
    <cue:ingredient xml:id="Rindkochfleisch" BLSref="U180100">
        <cue:prefBasicForm>Rindfleisch</cue:prefBasicForm>
    </cue:ingredient>
    <cue:ingredient xml:id="Midder" BLSref="V582100">
        <cue:prefBasicForm>Midder</cue:prefBasicForm>
        <cue:altBasicForm>Kalbsmidder</cue:altBasicForm>
        <note>"Kalbsmidder ist auch unter dem Synonym [...]</note>
    </cue:ingredient>
    <cue:ingredient xml:id="Weißwein" BLSref="P220000">
        <cue:prefBasicForm>Weißwein</cue:prefBasicForm>
        <cue:altBasicForm>weißer Wein</cue:altBasicForm>
    </cue:ingredient>
    <cue:ingredient xml:id="Butter" BLSref="Q610000">
        <cue:prefBasicForm>Butter</cue:prefBasicForm>
    </cue:ingredient>
    <cue:ingredient xml:id="Rotwein" BLSref="P240000">
        <cue:prefBasicForm>Rotwein</cue:prefBasicForm>
        <cue:altBasicForm>roter Wein</cue:altBasicForm>
    </cue:ingredient>
    <cue:ingredient xml:id="Kalbkochfleisch" BLSref="U380100">
        <cue:prefBasicForm>Kalbfleisch</cue:prefBasicForm>
    </cue:ingredient>
    [...]
</cue:listIngredient>
```

Abb. 4b: Aus Abb. 4a extrahiertes Zutaten-Wörterbuch

```
{
        "Rindfleisch"   : "Rindkochfleisch",
        "Midder"        : "Midder",
        "Kalbsmidder"   : "Midder",
        "Weißwein"      : "Weißwein"
        "Wein"          : [
                            "Weißwein",
                            "Rotwein"
                          ],
        "Butter"        : "Butter",
        "Rotwein"       : "Rotwein",
        "Kalbfleisch"   : "Kalbkochfleisch",

        "Fleisch"       : [
                            "Rindkochfleisch",
                            "Kalbkochfleisch"
                          ],
        [...]
}
```

Version 0.2

Die Evaluierung der Version 0.1 (Evaluierung.html#DictBasedV01) zeigt, dass die dictionary-based Extraktion funktioniert. Daher lohnt sich die Weiterverfolgung des dictionary- and rule-based Ansatzes. Bevor wir uns jedoch mit der rule-based Weiterverarbeitung beschäftigen, erinnern wir uns daran, dass der Prototyp bis jetzt eine quick and dirty Machbarkeitsanalyse war, welche wir

zuerst redesignen wollen. Die finale Version ist in unserem Git-Repository unter dem Tag Dict-AndRule-BasedPrototypeV0.2 (https://git.informatik.uni-kiel.de/stu121256/Torsten_Knauf_Master-Arbeit_Davidises_Kochbuch_Source_Code/tree/Dict-AndRule-BasedPrototypeV0.2) zu finden.

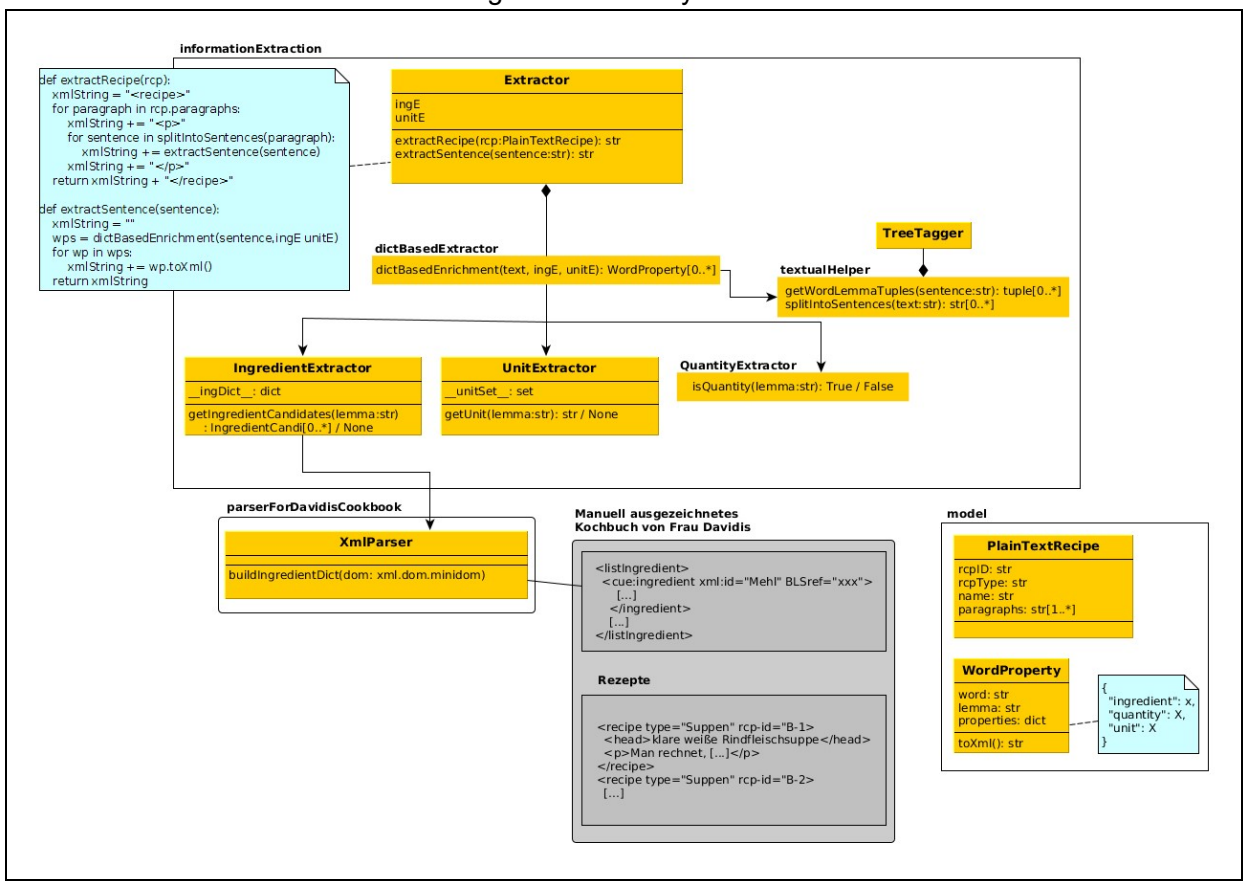
Design der dictionary-based Extraktion

Abb. 5 zeigt die aufs wesentliche reduzierten Bestandteile des Designs. Jedes Rezept wird zunächst in ein **PlainTextRecipe** umgewandelt, welches zu jedem Rezept die rcp-id, den Rezept-Typ, den Namen und eine Liste von Zubereitungs-Anweisungen speichert. Satzweise führt der **Extractor** dann mittels **dictBasedEnrichment** eine Extraktion der gesuchten Entities durch. Dazu werden zunächst mit der Hilfe von **TreeTagger** die Lemmata zu jedem Wort gebildet. Diese Lemmata werden dann durch den **IngredientExtractor**, **UnitExtractor** und **QuantityExtractor** getestet. Ist einer dieser Tests positiv, wird in der entsprechenden **WordProperty** im **properties**-dict das Ergebnis des Tests hinterlegt. Der **QuantityExtractor** besteht nur aus der Funktion **isQuantity**, welche ein Lemma bekommt und *True* bzw. *False* wiedergibt. Das Wörterbuch des **IngredientExtractor** wird mittels dem **XmlParser**, wie zuvor beschrieben, aus dem manuell mit cueML angereicherten Kochbuch extrahiert. Das Wörterbuch des **UnitExtractor** ist manuell im Attribut ***__unitSet__*** eingetragen.

Das Ergebnis der Extraktion des Satzes *„Schütte 1 Maß Wein hinzu.“* mit dem Zutaten-Wörterbuch aus Abb. 4b wäre zum Beispiel die Liste folgender WordProperties:

```
[
   WordProperty(word="Schütte", lemma="schütten", properties={} ),
   WordProperty(word="1", lemma="1", properties={"quantity":"1"} ),
   WordProperty(word="Maß", lemma="Maß", properties={"unit":"Maß"} ),
   WordProperty(word="Wein", lemma="Wein", properties={"ingredient":["Weißwein, "Rotwein"]} ),
   WordProperty(word="hinzu", lemma="hinzu", properties={} ),
   WordProperty(word=".", lemma=".", properties={} )
]
```

Abb. 5: Design der dictionary-based Extraktion



Das saubere Design erleichtert auch eine genaueren Analyse der vorherigen Evaluation der Version 0.1. Aufbauend auf dieser Analyse haben wir noch folgende Punkte verbessert:

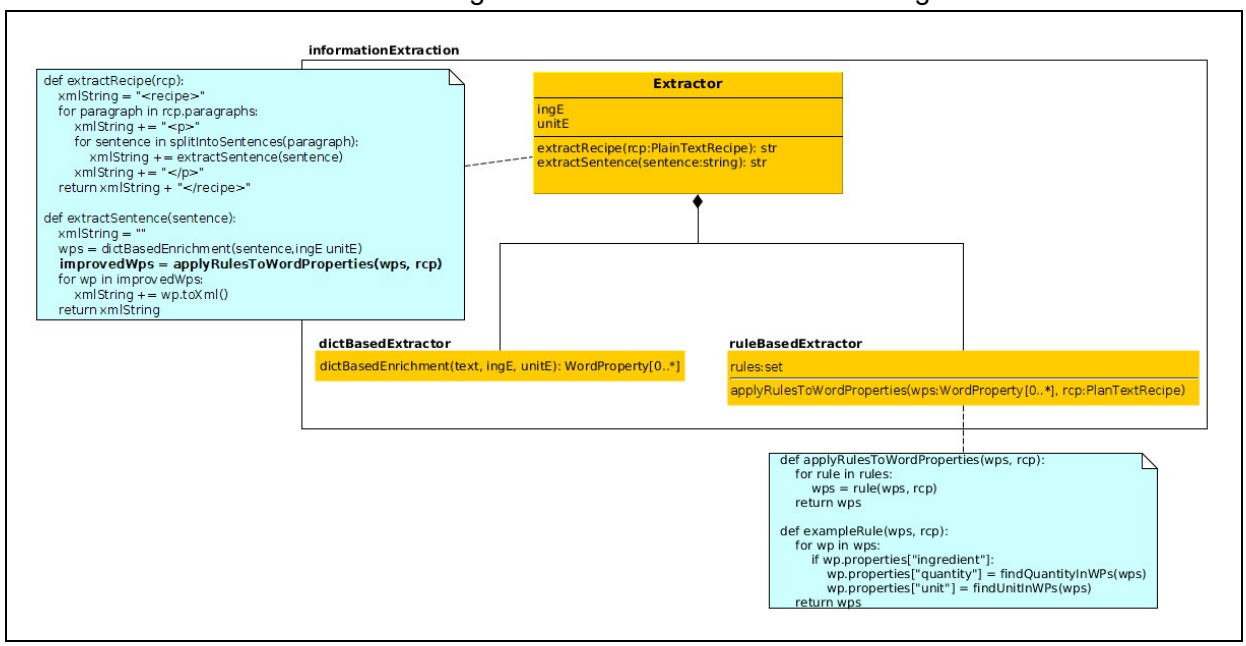
- Um zu vermeiden, dass beispielsweise *abgebrüht* als Zutat extrahiert wird, da es das Teilwort *brühe* enthält, schlägt der Test mit dem IngredientExtractor nun bei klein geschrieben Wörtern immer fehl.
- TreeTagger kann zu Pluralen die mit *n* gebildet werden, oft das Lemma nicht bilden (z. B. bei *Scorzonerwurzeln* oder *Kartoffeln*). Daher überprüfen wir beim Test mit dem Zutaten-Wörterbuch in der Methode *dictBasedEnrichment* das Lemma sowie das Lemma ohne n am Ende, wenn es auf n aufhört.
- TreeTagger findet manchmal mehrere Lemmata durch | getrennt. Beispielsweise kann das Lemma zu *„Linsen“ „Linse|Linsen“* sein, was nicht in unserem Zutaten-Wörterbuch zu finden ist. Ist dies der Fall, nehmen wir daher nur das erste Lemma (Linse).
- Manchmal schreibt Frau Davidis eine Zutat als zusammengesetztes Wort (z. B. *Petersilien-Wurzel*) und manchmal nicht (*Petersilienwurzel*). Man könnte die zusammengesetzte Schreibweise ins Wörterbuch als alternative Schreibweise hinzufügen. Wir berücksichtigen das hingegen im Programm. Wenn das Lemma zu einem Wort nicht gefunden werden konnte, eliminieren wir den Bindestrich aus dem Wort, und versuchen dann erneut ein Lemma zu finden.
- Die Liste der typischen Endungen von abgekürzten Wörtern wurde um *klöße*, *kloß*, *brot* und *brod* erweitert.
- Uns sind einige Fehler im manuell ausgezeichneten Kochbuch aufgefallen, welches unseren Golden Standard bildet, die wir korrigiert haben.
- Wir haben das Zutaten-Wörterbuch mit einer manuell erstellten Liste von Zutaten erweitert. Diese ist hier (DavidisesKochbuch/manualCreatedListOfDavidisIngredients.txt) zu finden.

Rule-based Weiterverarbeitung

In das saubere Design aus Abb. 5 fügen wir nun eine rule-based Weiterverarbeitung ein. Diese ist in Abb. 6 zu sehen. Nachdem die *WordProperties* extrahiert sind, wird auf diese eine Menge von Regeln angewendet. Eine Regel bekommt als Parameter die Liste der WordProperties eines Satzes, sowie das Rezept, zu dem der Satz gehört und gibt eine Liste mit geänderten WordProperties zurück. Beispiele für solche Regeln sind:

- **findQuantityAndUnitOfIngredientRule:** Die Regel versucht zu jeder WordProperty (wp), die als Zutat ausgezeichnet ist (*key "ingredient"* $\in$ *wp.properties*), eine Mengenangabe und Einheit zu finden. Sie sucht dafür den nächsten Nachbarn von links (wp'), dessen WordProperty als Mengenangabe bzw. Einheit ausgezeichnet ist und fügt diese der wp hinzu (*wp.properties["quantity"] = wp'.properties["quantity"]*). Um zu verhindern, dass beispielsweise bei *„Man nehme 2 EL Mehl und Butter“* 2 EL Butter extrahiert wird, bricht diese Regel ab, wenn vor der Mengenangabe bzw. Einheit eine andere Entity wie eine andere Zutat oder Kochzeit gefunden wird.
- **dissolveAmbiguityWein:** Wenn das Lemma zu einer Zutat *wein* enthält, sucht diese Regel nach den Wörtern „rot“ oder „weiß“. Wird eins von beiden gefunden, werden aus der Liste der möglichen xml:ids alle herausgefiltert, die dieses Wort nicht enthalten.
- **dissolveAmbiguityFleisch:** Ist die Zutat *Fleisch*, der Rezept-Typ *Suppen* und „rind“ ist im Namen des Rezeptes enthalten, so wird die xml:id auf Rindkochfleisch gesetzt.
- **dontUseRule:** Wenn vor einer Zutat das Wort „ohne“ steht, wird die Zutaten-Eigenschaft des Wortes zurückgenommen (*wp.properties["ingredient"] = None*).
- **optionalRule:** Sind nach einer Zutat die Lemmata „können“ und „wegbleiben“ im Satz enthalten, so wird diese Zutat als optional ausgezeichnet (*wp.properties["optional"] = True*).
- **altGrpRule:** Folgt auf eine Zutat das Lemma „statt“, werden alle Folgenden durch „und“ separierten Zutaten als Alternative zu der ersten Zutat ausgezeichnet.

Abb. 6: Design mit rule-based Weiterverarbeitung



Automatisches Auszeichnen mit cueML

Nachdem alle Entities mittels den WordProperties extrahiert wurden, müssen diese noch, mit dem *cueML*-Vokabular ausgezeichnet, zurückgeschrieben werden. Aufgrund unserer guten Kapselung der Entities in den WordProperties ist dies trivial. Jede WordProperty hat eine *toXml*-Methode. Ist ein Wort keine Zutat, wird von der Methode die Wort-Identität zurückgegeben. Ansonsten wird die Word-Identität in ein *recipeIngredient*-Element mit den entsprechenden Attributen eingeschlossen.

Transformation in HTML

Durch eine Transformation der mit cueML ausgezeichneten Rezepten in HTML kann das digitalisierte Kochbuch nun übers Internet jedem zugänglich gemacht werden.

Das Python-Script *cueML2Json.py* parst die cueML Rezepte in ein *JSON-Objekt*, welches von einer Webseite ausgelesen werden kann. In der oberen Navigationsleitste im Menüpunkt Rezepte (Rezepte.html) sind aufbauend auf dem von cueML2Json.py erstellten JSON-Objekt die in HTML transformierten Rezepte von Frau Davidis zu finden. Da unser Prototyp noch ausschließlich die Zutaten mit ihren Mengenangaben und Mengeneinheiten auszeichnet, kann die Transformation auch nur diese Entities berücksichtigen. Daher enthält die Webseite beispielsweise keine Verweise zwischen den Rezepten.

Eine andere Möglichkeit die mit cueML ausgezeichneten Rezepte in HTML zu transformieren ist mittels *XSLT*. Während meiner SHK-Tätigkeit habe ich die mit dem experimentellen cueML ausgezeichneten Rezepte mittels XSLT in HTML umgewandelt. Dies hat den Nachteil, dass die Erstellung der Navigationsbar für die Rezepte schwieriger ist, da nicht alle Rezepte in einem Objekt gekapselt sind. Da das Auszeichnen mit dem experimentelle cueML manuell erfoglte, wurden dort alle Entities wie beispielsweise Zubereitungszeiten oder Verweise auf andere Rezepte ausgezeichnet. Als Beispiel wie alle Entites von cueML in HTML dargestellt werden können, sei daher hier (https://git.informatik.uni-kiel.de/nl/Culinary_Analysis/tree/master/Webseite/html) auf die Transformation des experimentellen cueMLs mittels XSLT verlinkt. Da sich das experimentelle cueML sehr von der aktuellen Version unterscheidet, ist eine Anpassung des Scripts jedoch leider aufwendig.