Ontologies for QA on Restricted Domains:
usefulness and how to build them

Wilko Wieringa
w.k.wieringa@student.utwente.nl

ABSTRACT
Coreference and anaphoric references are necessary to find answers to questions. This paper describes the development of an ontology for the domain of recipes to improve references for this domain. First we will discuss the usefulness of ontologies for a Question Answering System, after which we describe how we create or build an ontology. We must extract knowledge from cooking recipes and translate that knowledge to an ontology. This will be done with the use of the Dutch WordNet and statistical analysis of Dutch cooking recipes.

Keywords
Ontology, Question Answering, Knowledge Extraction, Cooking Recipes, Dutch Wordnet, Coreference, Anaphora.

1. INTRODUCTION
When you read the weather report in a newspaper you must have an understanding of the different concepts and words used in the forecast. If you read it to check if you can go tomorrow to a theme park and the weather reports predicts it’s getting to rain you must know that you get wet from rain. If you get soaked you feel uncomfortable and when you feel uncomfortable you will enjoy the theme park experience less in comparison if you went when it wouldn’t rain. With this knowledge you can decide if you are going or not. See figure 1 for a (simple and subjective) visual representation of a weather model.

Figure 1. Simple weather model

If a computer program can parse a weather report and can distill that it would likely rain it could make a decision if you should go to a theme park if it also had access to our weather model. Because this model has an (indirect) relation between uncomfortable and rain and if the program knows you don’t want to get uncomfortable it decides you shouldn’t go tomorrow. Of course, you can do this for yourself but this example is here to show how a model can help a computer to decide certain questions. A model of the world or a certain part of the world is called an ontology in the field of AI or knowledge sharing (Gruber, 1993).

An other example of an ontology is the relation between ingredients. Such an ontology could be used for the cooking domain. An example of a relation between ingredients is pieces and chicken, if you cut a chicken you get pieces of chicken. This kind of relation describes which ‘other’ ingredients you get when you perform a certain action upon an ingredient, in this case cutting. An other relation between ingredients is to which class it belongs, for example meat and steak. We will explain more about ontologies in Chapter 2.

We want to determine if an ontology is useful for a Question Answering (QA) System. A QA System is a system which answers questions from users by first analyzing the question with Natural Language Processing (NLP) techniques and when the system understands the question it searches for the answer in a collection of documents. A QA System is build for either an open or a closed domain. One could say that an open domain is (nearly) everything, an example would be a system which uses the world wide web to answer any question. When a system works with a closed domain it already (roughly) knows the ‘theme’ of the question. Examples of QA systems on a closed domain would be a system on the medical domain such as IMIX3 or in our case a QA system on the domain of cooking. In Chapter 4 we will discuss how ontologies can help QA Systems. In Chapter 5 we describe how we created an ontology for the cooking domain and in Chapter 6 we discuss how useful it is.

Unfortunately there is isn’t a cooking QA system, so how can we determine the usefulness of the created ontology? The ontology that we are building for the cooking domain will be tested in a program that was written by van der Ham and Jansen, called ReceptenParser4. The ReceptenParser was made as an assignment for the class Conversational Agents 1. With the program it is possible to find coreferences in recipes (van der Ham and Jansen, 2006). Coreferences are certain relations between words and ReceptenParser can find some of these coreferences. With the ontology we want to improve the coreferences and anaphoric references.
relations. In Chapter 3 we will explain what these relations precisely are and we will explain in what way we want to improve the ReceptenParser.

2. ONTOLOGIES

We started this research with the idea that an ontology would be helpful in an QA System. In this chapter we will explain what an ontology is and in Chapter 4 we discuss how it helps a QA System. Let’s first see what ontology means, the term is taken from philosophy and (Craig, 1998) gives the following explanation:

The word ‘ontology’ is used to refer to philosophical investigation of existence, or being. Such investigation may be directed towards the concept of being, asking what ‘being’ means, or what it is for something to exist; it may also (or instead) be concerned with the question ‘what exists?’, or ‘what general sorts of thing are there?’

Ontology has a different meaning in the field of AI or knowledge sharing, (Gruber, 1993) gives the following meaning in the AI field:

For AI systems, what “exists” is that which can be represented. When the knowledge of a domain is represented in a declarative formalism, the set of objects that can be represented is called the universe of discourse. This set of objects, and the describable relationships among them, are reflected in the representational vocabulary with which a knowledge-based program represents knowledge. Thus, in the context of AI, we can describe the ontology of a program by defining a set of representational terms. In such an ontology, definitions associate the names of entities in the universe of discourse (e.g., classes, relations, functions, or other objects) with human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms. Formally, an ontology is the statement of a logical theory.

An ontology is a structured web of words and concepts which are connected with each other. The top of an ontology are the concepts, and concepts can have more specific concepts. This determines the structure of the ontology. An example of this from the cooking ontology would be Ingredient - Animal - Meat. These are 3 concepts, but it gets more specific. Words should be connected to these concepts, otherwise the ontology won’t have any use. Concepts give meaning by their structure to words, but without words connected to these concepts the program can’t ‘understand’ the words in the recipes. Steak doesn’t mean anything to a computer, but if it looks up steak in the ontology it knows that it is part of the category Meat, which is part of the category Animal, etc. It now can answer for example the (simple) question Is steak meat? - given it has a good answer parsing mechanism - because it knows steak belongs to the category meat. We will discuss the usefulness more in Chapter 4, but first we must know with which kind of problems the ontology should help with.

3. REFERENCES IN TEXTS

ReceptenParser finds coreferences in recipes. They use the definition of coreference as used by (van Deemter and Kibble, 1999), which is:

\[ a_i \text{ and } a_j \text{ cohere if and only if Reference}(a_i) = \text{Reference}(a_j) \]

With Reference(a) is meant the entity referred to by a. An example of coreference is:

\[ \text{Harm went to the bakery, where he bought one loaf of coarse whole-wheat bread.} \]

Where Harm and he refer to the same entity.

In comparison with a reference called anaphora, this is relatively easy since no world knowledge is necessary. An example of an anaphoric relation is the following:

Cut the chicken and put the pieces in the pan.

Chicken refers to pieces, but they aren’t interchangeable without changing the semantics of the sentence. Anaphoric relations occur often with whole/part-relations. (van der Ham and Janssen, 2006) use this as a definition for anaphoric relations:

Anaphora are a specific case of reference to an earlier named expression, where those two expressions aren’t interchangeable without changing the semantic meaning of the sentence.

3.1 Improving references in ReceptenParser

If you load a recipe in the ReceptenParser it shows the recipe with certain words underlined and certain words made bold. Blue words are connected to the first reference it has in the recipe. When you click on a blue word, the first reference of that word is made bold. It finds relative easy references such as 2 boterhammen (2 sandwiches) and de boterhammen (the sandwiches). ReceptenParser doesn’t find a relation between Fish and salmon or steak and Meat, this means it doesn’t know to which class ingredients belong. Our goal is to improve this with the help of an ontology.

An other relation it doesn’t find is the already mentioned example of chicken and pieces. This means it doesn’t find anaphoric relations which relates to cutting, the second goal of our ontology would be to help in these cutting relations.

4. QUESTION ANSWERING AND ONTOLOGIES

A Question Answering System deals with questions posed in natural language. Its efficiency is determined by the quality of the corpus, question classification, question processing, answer extraction and answer formulation. Knowledge based components with World Knowledge in QA Systems could have benefits as stated by (John Burger, 2003):

Customization The answers can be customized to a particular situation of the person when answers are being put together from a knowledge base.

Controllable level of detail By controlling how much information from the knowledge base could be included in the answer it is possible to let an expert user and a beginner use the same QA System. The beginner gets a less complicated answer and the expert a more complicated answer or an answer with more information.

Robustness By inferring answers instead of extracting them the QA System can respond to unexpected questions from the user and when no answer can be found in the corpus it could resolve the situation by deducing the answer.

Of course, this is still a bit far away from the reality and this cannot be done only with an ontology, but ontologies are used for representing World Knowledge. An ontology can have several different functions in a QA system or it could use several different ontologies each with its own field of specialty. It the next two subsections it will be about two types of ontologies for a cooking QA system. The third subsection will describe how an incomplete ontology can hinder a QA System.
4.1 Recipe Ontology

This ontology has a recipe as the concept under which everything falls. Under the concept recipe there are other concepts such as ingredients, cooking gear, number of persons and so on. Also it should know the different steps in a recipe. The ontology now can determine which ingredients go well together, because it knows which ingredients are often used together - which must mean something. Of course, the level of how often ingredients are together should be relative high, otherwise you would get terrible suggestions. It could also determine from a given set of ingredients which recipes could be made. A Recipe Ontology could assist in questions such as:

- What can I make with ingredients X, Y and Z?
- Please give me a vegetarian recipe.
- Which instruments do I need for recipe X?
- What should I do next after I added X in recipe Y?
- How much of X is needed in recipe Y?

One could add to the Recipe Ontology a degree of complexity connected to a recipe based on the number of steps in the recipe and the number of different ingredients. One factor could also be the number of words which are in the recipes, as we think that most of the time lesser words means an easier recipe. Preparation time could also be a factor if known, though one should detract times such as oven times, etc. With this you can give recipes to beginners and experts, the controllable level of detail as mentioned above.

4.2 Reference Ontology

This ontology is specialised in relations between words, such as ingredients and what kind of actions you can perform on them and what will be the result of that action. This could help in the analysis of questions or the corpus and it could spot ambiguity and could ask for clarification. It should also contain which instruments are used for which cooking action, so cooking gear can be recognized. A question like Which instrument do I need for frying? could be answered. As stated in section 3.1 we want to create such an Reference Ontology.

4.3 Possible Problems

If a QA system is depended on an ontology for question analyses and there are errors in the ontology or connections aren’t made it will lead to mistakes of the system. An example where (some) anaphoric relations weren’t right was with a QA system build by Flycht-Eriksson and Jönsson. Their QA system was about birds and the domain knowledge in the system is represented in an ontology. In their ontology the concept “bird” and the property “Colour” via the body parts wasn’t connected. So the question What colour is a blue tit? was answered by Failed to interpret the question. Please try again! If it was better connected it could have dealt with the question, because birds are made up of body parts and body parts have colours. A body part was missing in the question and (Flycht-Eriksson and Jönsson, 2003) say with an ontology with the aforementioned link it could have replied with A blue tit has several colours for different body parts. What part of the blue tit are you interested in?

5. ONTOLOGY CREATION

We want to create an ontology on a closed domain, that of cooking and the domain is focused on the Dutch language. Knowledge must come from cooking recipes. But how are we going to accomplish this? There are other frameworks for ontology learning from textual sources, such as (Mikheev and Finch, 1995) and (Cimiano and Volker, ). Since these are large frameworks, we have decided to do it in our own way because of the time learning how to use those frameworks. There are similarities between our aproach and theirs such as using Wordnet to find SubClasses of relations (Cimiano and Volker) or finding patterns in text (Mikheev and Finch, 1995).

How we have created the ontology will be told step by step. See figure 2 for a visualisation of all the steps.

![Flow diagram for creating the ontology.](image)

5.1 Parsing the recipes

First we have to find recipes, thankfully the Internet is a big help in this. There are a lot of recipes to find, even in Dutch. The main site we used for getting recipes is recepten.net. The gathered recipes were first in HTML form, so they had to be stripped to clean text format.

This was done via regular expressions in Java. Regular Expressions are a language for specifying text search strings. To search with regular expressions one must make a pattern where one wants to search for and of course a corpus of texts to search through - in our case the cooking recipes (Jurafsky and Martin, 2000).

‘Clean’ recipes aren’t useful for information extraction, they have to be annotated so we know what words are nouns and verbs. This is done with the Alpino Parser. This a dependency parser for the Dutch language and was developed at the University of Groningen. It returns a dependency tree of a sentence in XML format. Dependency structures represent the grammatical relations that

---

5 We gathered also recipes from receptjes.be and http://people.zeelandnet.nl/vdbindt/koken/koken.htm. Those aren’t used in the ontology creation.
6 http://ziu.let.rug.nl/vannoord/alp/Alpino/, 28 May 2006
are between constituents. They are more abstract than syntactic trees (word order for example isn’t expressed), but are more explicit about the dependency relations. There are indices to denote that constituents may have multiple (possibly different) dependency relations with different words (Leonoor van der Beek, 2002). See Apèndix A for an sentence returned by the Alpino Parser.

5.2 Starting the Ontology

The notation of our ontology will be in OWL. OWL stands for Web Ontology Language and is a vocabulary extension of RDF and is derived from the DAML + OIL Web Ontology Language (http://en.wikipedia.org/wiki/Web_Ontology_Language), 28 May 2006. There are three flavours of OWL, namely OWL-Lite, OWL-DL, and OWL-Full. OWL-Lite is intended to be used when there is a simple class hierarchy and where simple constraints are needed. You can express more in OWL-DL and it is based on Description Logics, with which automatic reasoning is possible. Because of this it is possible to check for inconsistencies in the ontology. OWL-Full is used when expressiveness is more important than to guarantee the decidability or computational completeness of the language (Horridge et al., 2004). For our ontology we take OWL-DL, because - as already stated - it is possible to check for inconsistencies and it is more expressive than OWL-Lite.

There are two kinds of entries in the ontology - concepts and words. Concepts are the highest level of the ontology and the words are connected to these concepts. There are a number of higher concept ontologies already made, such as the SUMO knowledge base. SUMO stands for Suggested Upper Merged Ontology, it gives definitions for general-purpose terms and can be used as a foundation for more specific domain ontologies. The idea behind higher concept ontologies is that if people make ontologies based on a common ontology, it will gain some degree of interoperability with other compliant systems (Niles and Pease, 2001). They expect that it will eventually contain between 1000 and 2500 terms. Our cooking ontology doesn’t make use of it, because it has no real use for our system and it makes the ontology much bigger than necessary. It’s relative easy to connect our ontology to SUMO if there is the need for it.

5.3 Concepts of the Ontology

How did we determine the concepts of the ontology? There are two top concepts in the ontology - Cooking Action and Ingredient. Under each concept it becomes more and more specific. Every ‘lowest’ action has - most of the times - an Ingredient and an Instrument. Take for example Cutting, Ingredient, the words connected to this concept are the result of a cutting action or the ingredient could be cut. Think of Take the chicken and cut it into pieces, both chicken and pieces should be placed under Cutting - Ingredient, how this is done will be explained in section 5.4. See figure 3 for an image of the Cooking Action part.

The preparation terms are mostly inspired by wikipedia, since it’s quite an extensive list. Not everything was used as a concept in order to not overspecify the ontology. If necessary concepts can always be added easily - or deleted. We think we have the principal parts covered in the ontology. What is meant by the different Cooking Actions:

Assembling Putting Ingredients together to get a new ‘bigger’ ingredient, such as a mixture.

Cooling Speaks for itself, from water to ice for example.

Disassembling Making Ingredients smaller, from paprika to strips. Has more specific disassembling acts beneath it. Cutting is not the same as Grating, you grate different ingredients than you cut and there is a difference between the instruments used.

Heating Speaks for itself, also has more specific acts beneath it.

Measuring How much gramme of flour or to which degree must the oven preheat?

On Ingredient This category is meant for actions on the ingredient itself, but not disassembling or assembling the ingredient. Actions such as kneading, washing, peeling, etc.

Figure 3. A graphical depiction of Cooking Action concepts viewed in Protégé 3.1.1

The other top concept, Ingredient, is there for capturing the structure of all the different ingredients. This is done to get anaphoric references such as salmon and fish. Sometimes in recipes they write salmon in the ingredients list and use the word fish in the preparation list. But with a structure like this we can reference those two words. Again, Wikipedia was the biggest inspiration for the Ingredient concepts. See figure 4 for a visualisation of the Ingredients concepts.

7A constituent is a group of words that may behave as a single unit or phrase. The most common grouping is the noun phrase, which is a sequence of words surrounding a noun (Jurafsky and Martin, 2000).

8Every Ingredient and Instrument has to have a unique name, since Protégé enforces this - hence the prefix before the words.


10Peeling is in my opinion removal of the outer layer of the ingredient. When you peel an onion, you throw away the covering after which you cut the onion into pieces.

The concept names are in English as this is standard with other ontologies and it could be used by non-dutch users. Though, in the comments of the concept the dutch word for the concept is written.

5.4 Populating the Ontology

We have used the Dutch WordNet in order to populate a part of the ontology. The Dutch Wordnet is a semantic lexicon\textsuperscript{12} for the Dutch language and is developed by the University of Amsterdam in the EuroWordNet project (Vossen et al., 1999). There is a Java interface for the English version, only this can’t be used with the Dutch version. Fortunately, Erwin Marsi wrote a command line interface in Python\textsuperscript{14} for the Dutch WordNet called ewnpy\textsuperscript{15}.

The Dutch WordNet was used for the extraction of ingredients,\textsuperscript{16} (extra) cooking actions and cooking instruments. This was done in Python and the script contains a translation array, which contains the English concept and the Dutch counterpart - or what was used in the Dutch WordNet. The script also contained the Dutch Wordnet which could be loaded as a pickle since ewnpy was also written in python. A pickle is a database file optimized for Python. In the script the hyponyms were inserted in the Dutch Wordnet, which returned a string of words which fell under the particular hyponym. Every word was outputted in OWL format with as Class ID the word itself and with subClassOf ID the particular hyponym.

The relations we can make after we populated the ontology with the Dutch Wordnet are anaphoric references which deal to which class an ingredient belongs to. References such as mai̇s (corn) and graan (cereal) and basilicum (basil) and kruid (herb) can be made.

Another way to populate the ontology is through statistical analysis or datamining. You should look for patterns in recipes and see how often certain nouns in those patterns appear. We will do this for one pattern in order to show how such a thing must be done. Chapter ?? discusses other patterns which could be used to populate the ontology.

The example which we work out is checking which nouns appear often with the verb cutting. Sentences for instructions in cooking recipes are often short and to the point. Take the following two sentences:

De kip in stukken snijden.
De paprika’s halveren, schoonmaken en in reepjes snijden.

As you can see before the verb snijden there stands two nouns - kip (chicken - stukken (pieces) and paprika’s (paprikas) - reepjes (slivers) - which could be cut or are the result of the cutting proces. If you take enough recipes, you can find out which ingredients should be cut and what the result of the cutting action is. An other often seen pattern is two nouns behind the verb snijden. See the following two sentences:

Snij de kipfilet in blokjes.
Snijd de worteltjes in schuine stukjes.

Looking for this pattern gives you more ingredients or more certainty about ingredients which are connected to cutting. The top results of searching for these two patterns in 783 recipes can be found in figure 5.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure4.png}
\caption{A graphical depiction of Ingredient concepts viewed in Protégé 3.1.1}
\end{figure}

As you can see the word minuten is the only non-cutting word, which can easily be removed in Protégé, which is an ontology editor. Also some words have their plural form, this is done because ReceptenTagger uses the WOTAN Postagger, which doesn’t stem.

So how does such a concept or word look in OWL? A concept is specified as follows:

\begin{verbatim}
<owl:Class rdf:ID="Animal">
     <rdfs:subClassOf>
         <owl:Class rdf:ID="Ingredient"/>
     </rdfs:subClassOf>
     <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">dier</rdfs:comment>
</owl:Class>
\end{verbatim}

This is the class Animal which is a subclass of Ingredient. rdfs:datatype says which datatype the comment is denoted, in this case a normal String. The link leads to a website, which has rddl\textsuperscript{17} information and provides a package of information about the target.\textsuperscript{18} A word is just the same, only it doesn’t have a comment field in our case.

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID="Ingredient"/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
\end{verbatim}

\begin{verbatim}
<owl:Class rdf:ID=""/>
The size of the biggest common substring in each string is calculated, and it's an improved version of XProlog by van der Ham.

The number of actions to get from one string to the other is calculated.

Explains Prolog as follows, but these two relations don’t have to appear in a recipe. Also as noted by, but these two relations occur in a recipe, our ontology helps with coreferences.

Also as noted by (Flycht-Eriksson and Jonsson, 2003) anaphoric relations improve analysis of questions in a QA system. References about the classification of ingredients are also found, which could answer questions as Which fish should I use in recipe Viss-kotel?.

A discourse is a group of sentences which are related to each other, in our case of course the recipe. The matching of references are being done with fuzzy string comparison. Strings are getting compared on:

- The number of common characters.
- The number of actions to get from one string to the other.
- The size of the biggest common substring in each string.

With the above methods in mind, two strings get a score between 0 and 1. If an other NP is found, the relation is added in the list of references. Otherwise it gets added to the list of found NP’s.

Our ontology gets into play when checking for references. In order to easily check the ontology it has been converted to Prolog rules and facts. This was done by writing a Java script which parsed the OWL file with regular expressions and then made Prolog rules and facts of it. Wikipedia explains Prolog as follows, it is a logic programming language and is based on first-order predicate calculus. Executing a Prolog programming is proving a theorem with first-order resolution. In Prolog you give a database of facts and rules, after which you can query the database. A fact or a predicate looks as follows:

```
Meat(lamsvlees).
```

Which says lamsvlees is Meat and is used for words in our case. The connection of the concepts is being expressed via rules, such as the following one:

```
animal(X) :- meat(X).
```

This says, if something belongs to the class meat it also belongs to the class animal.

To capture references such as vlees (meat) - biefstuk (steak) or vis (fish) - zalm (salmon) the member to which each ingredient belongs to gets checked and when found added to the references. To find anaphoric references, for example that deals with cutting every last seen ingredient that could be cut is being remembered. If a noun gets found that is the result of the cutting process a relation gets drawn between that noun and the last cuttable ingredient. The idea behind this is that a word such as reepjes (strips) always points to the last mentioned cuttable ingredients, for example kipfilet (fillet of chicken). If this wouldn’t be the case the recipe in itself would be unclear and a human being also wouldn’t understand it to which ingredient it would point. See figure 6 for the Receptenparser in action, with de vis (the fish) selected and 250 gram kabeljauw of zalm (250 grams codfish or salmon) highlighted.

We used YProlog to query the prolog database with the ReceptenParser because both are written in Java. YProlog is written by Boris van Schooten and it’s an improved version of XProlog which is an improved version of WProlog.

## 6. CONCLUSION

The goal of this research was to see if an ontology was useful in QA systems on closed domains, but since there isn’t a Cooking QA system it is a bit hard to check. We wanted to improve the ReceptenParser so that it could find anaphoric relations with cutting and it would be able to know about which class ingredients belong to. It isn’t possible to test the working of the ontology in ReceptenParser automatically, because you can only test it by hand. This is done by starting the version which doesn’t use an ontology and loading a recipe, then you start the version which uses an ontology, load the same recipe and see if there are more or better references.

We have done this with several recipes and it made relations between classes of ingredients (such as meat or fish) and ingredients (such as steak or salmon) or (anaphoric) cutting relations, such as reepjes (slivers) and paprika (paprika). There is an improvement in the ReceptenParser when these two relations occur in a recipe, but these two relations don’t have to appear in a recipe.

Also as noted by (Flycht-Eriksson and Jonsson, 2003) anaphoric relations improve analysis of questions in a QA system. References about the classification of ingredients are also found, which could answer questions as Which fish should I use in recipe Viss-kotel?.

19Noun Phrase

Figure 5. The result of determining cutting references

<table>
<thead>
<tr>
<th>Noun Phrase</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>tomaat</td>
<td>206</td>
</tr>
<tr>
<td>stukjes</td>
<td>192</td>
</tr>
<tr>
<td>stukje</td>
<td>192</td>
</tr>
<tr>
<td>blokjes</td>
<td>191</td>
</tr>
<tr>
<td>blokke</td>
<td>191</td>
</tr>
<tr>
<td>tomaten</td>
<td>157</td>
</tr>
<tr>
<td>knooflook</td>
<td>148</td>
</tr>
<tr>
<td>paprika</td>
<td>137</td>
</tr>
<tr>
<td>ui</td>
<td>132</td>
</tr>
<tr>
<td>stukken</td>
<td>132</td>
</tr>
<tr>
<td>stuk</td>
<td>132</td>
</tr>
<tr>
<td>plakje</td>
<td>131</td>
</tr>
<tr>
<td>plakjes</td>
<td>127</td>
</tr>
<tr>
<td>appel</td>
<td>126</td>
</tr>
<tr>
<td>champignon</td>
<td>124</td>
</tr>
<tr>
<td>champignons</td>
<td>124</td>
</tr>
<tr>
<td>minuten</td>
<td>121</td>
</tr>
<tr>
<td>reepjes</td>
<td>119</td>
</tr>
<tr>
<td>reepje</td>
<td>119</td>
</tr>
</tbody>
</table>

The ontology consists of 100 concepts and 647 words. Some words are double though, but they hang under multiple categories. With prefixes the unique name requirement of Protege was satisfied. 100 Concepts seems like quite a lot, though compared to SUMO which is expected to have between 1000 and 2500 it is relative little. We must admit that on the Cooking Action side a few concepts could be left, since these aren’t populated such as smoking and chopping and their respectable Ingredient and Instrument.

There are a few plurals in the ontology, this doesn’t seem like a big deal, but when its gets more and more populated it is. The RecepenParser should make use of the Alpino parser if we want to remove these plurals. This is because the Alpino parser stems the words and removes the necessity of plurals - though sometimes this goes wrong or it returns the plural form as root.

Checking of the consistency was done in Protege, first there was checking of the sentences wouldn’t be dependend on the scriptwriter for seeing patterns. It could look for cutting or whipping and use the tree which nouns are connected to the verb. Though populating the ontology with an expert is time-consuming because the words have to be added by hand. We didn’t use the dependency structure which was returned by the Alpino Parser, from the parsed sentences we only used the Part of Speech of words and their stem. When the dependency structures are used during the population of the ontology it could help which concepts are necessary and it could help filling the ontology.

One could also try to find more translations of classifications in the Dutch WordNet, since sometimes the hyponyms are different than one suspects of ingredients. A good idea would be to gather ingredients from ingredients lists and put them into the Dutch WordNet. After this the different hyponyms should be turfed (of course the ones already used should be discarded) and the top hyponyms could be used to match more ingredients and classifications.

It would be beneficial for the creation of every ontology by creating an ontology with an expert of the domain, because he or she has more knowledge about it and knows it better than a ontology creator who is relative new to the field. The expert could help which concepts are necessary and it could help filling the ontology. Because we use more or less the concepts or structure of Wikipedia it would be really nice to parse Wikipedia and populate automatically the ontology. Though populating the ontology with an expert is time-consuming because the words have to be added by hand.

The ontology contains 100 ingredients and 647 words. Some words are double though, but they hang under multiple categories. With prefixes the unique name requirement of Protege was satisfied. 100 Concepts seems like quite a lot, though compared to SUMO which is expected to have between 1000 and 2500 it is relative little. We must admit that on the Cooking Action side a few concepts could be left, since these aren’t populated - such as smoking and chopping and their respectable Ingredient and Instrument.

There are a few plurals in the ontology, this doesn’t seem like a big deal, but when its gets more and more populated it is. The RecepenParser should make use of the Alpino parser if we want to remove these plurals. This is because the Alpino parser stems the words and removes the necessity of plurals - though sometimes this goes wrong or it returns the plural form as root.

Checking of the consistency was done in Protege, first there was the idea to use RacerPro which is a reasoner to check the ontology. It was discarded because it would add work to figure out yet another program and because Protege can run some ontology tests by itself. Also it would be difficult to create logical errors in the ontology since Protege doesn’t allow loops, it enforces unique tests by itself. Also it would be difficult to create logical errors in the ontology since Protege doesn’t allow loops, it enforces unique tests by itself.

Acknowledgements

Thanks to Rieks op den Akker for giving useful critique and giving lots of helpful papers and web sites. Thanks to Anno Perk and Harm op den Akker for commenting on my work and to the rest of the QA-group for the discussions during the meetings.

References


Idea from CA 1 assignment from Nijdam and Heuts.

Idea from Rieks op den Akker.


**APPENDIX A: Example Sentence**

The (short) sentence *Direct serveren* is parsed as followed by Alpino:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<alpino_ds version="1.1">
  <node begin="0" cat="top" end="3" id="0" rel="top">
    <node begin="0" cat="inf" end="2" id="1" rel="--">
      <node begin="0" end="1" frame="adjective(no_e(adv))" id="2" infl="no_e(adv)" pos="adj" rel="mod" root="direct" word="Direct"/>
      <node begin="1" end="2" frame="verb(hebben,inf,intransitive)" id="3" infl="inf" pos="verb" rel="hd" root="serveer" sc="intransitive" word="serveren"/>
    </node>
  </node>
</alpino_ds>
```

The sentence *Direct serveren* is parsed as followed by Alpino:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<alpino_ds version="1.1">
  <node begin="0" cat="top" end="3" id="0" rel="top">
    <node begin="0" cat="inf" end="2" id="1" rel="--">
      <node begin="0" end="1" frame="adjective(no_e(adv))" id="2" infl="no_e(adv)" pos="adj" rel="mod" root="direct" word="Direct"/>
      <node begin="1" end="2" frame="verb(hebben,inf,intransitive)" id="3" infl="inf" pos="verb" rel="hd" root="serveer" sc="intransitive" word="serveren"/>
    </node>
  </node>
</alpino_ds>
```