Ingredients and Recipe for a Robust Mobile Speech-enabled Cooking Assistant for German

Ulrich Schäfer[1], Frederik Arnold[2], Simon Ostermann[2], and Saskia Reifers[2⋆]

[1] ulrich.schaefer@dfki.de, German Research Center for Artificial Intelligence (DFKI), Language Technology Lab, Campus D 3 1, D-66123 Saarbrücken, Germany
[2] {arnold,ostermann,reifers}@kochbot.de, Saarland University, Computational Linguistics Department, D-66041 Saarbrücken, Germany

Abstract. We describe the concept and implementation of *Kochbot*, a cooking assistant application for smartphones and tablet devices that robustly processes speech I/O and supports German recipes. Its main functions are (1) helping searching in a large recipe collection, (2) reading out loud the cooking instructions step-by-step, and (3) answering questions during cooking. Our goal was to investigate and demonstrate the use of speech assistance in a task-oriented, hands-free scenario. Furthermore, we investigate rapid domain adaptation by utilizing shallow natural language processing techniques such as part-of-speech tagging, morphological analysis and sentence boundary detection on the domain text corpus of 32,000 recipes. The system is fully implemented and scales up well with respect to the number of users and recipes.

Keywords: natural language processing, speech, mobile application, cooking assistant

1 Introduction and Motivation

With the advent of mobile devices such as smart phones and tablets, as well as robust, speaker-independent, cloud-based speech recognition in the last few years, new applications become feasible and marketable that researchers and end users a decade ago could only dream of. In this paper, we describe the components (ingredients) and methods (the recipe) of a mobile cooking assistant application that runs with out-of-the box hardware, assuming a speech recognition in the cloud such as the Google Speech API that comes with the Android mobile operating system[3].

Moreover, the cooking assistant application itself runs on the mobile phone as an "app". Only the rather large recipe collection (32,000 recipes) currently resides on a standard PC that can serve thousands of mobile clients with recipe search via Internet, given that each recipe's text only needs a few kilobytes. Considering the memory reserve in current mobile devices, this recipe collection

⋆ Preprint. The final publication is available at link.springer.com, LNAI 8077, pp. 212–223, 2013. We thank Ulrike Schweitzer for permitting us to use her collection of more than 32,000 recipes at http://ulrikesrezeptesammlung.de, and our pioneer users for testing earlier versions of the app. A beta is available at http://www.kochbot.de.

[3] http://dev.android.com/reference/android/speech/SpeechRecognizer.html

could in principle also easily be moved to the clients. In other words, the current system already now scales up with respect to the number of users and recipes.

The main tasks of the app are (1) helping searching in a large recipe collection, (2) reading out loud the cooking instructions step-by-step, and (3) answering questions during cooking. The purpose of our project was to investigate and demonstrate the feasibility and use of speech assistance in a task-oriented, hands-free scenario with state-of-art hardware. Furthermore, we investigate rapid, corpus-based domain adaptation by utilizing shallow natural language processing (NLP) techniques such as part-of-speech tagging, morphological analysis and sentence boundary detection on the domain text corpus (the recipes). By pre-processing the corpus of German recipes with these NLP tools, we can quickly access and utilize domain knowledge instead of modeling everything by hand (which some of the earlier approaches to cooking assistants did through domain ontologies).

This paper is organized as follows. In Section 2, we discuss previous and related work. Section 3 deals with the (networked) architecture of the overall system. Section 4 presents the offline pre-processing stages. Section 5 explains the mobile cooking assistant app itself: the user interface and different stages of speech interaction in search and recipe reading out mode. We conclude in Section 6 and give an outlook to future work.

2 Previous and Related Work

Various approaches to home cooking assistants and cooking tutoring dialog systems have been described in the literature. One of the earliest studies is Home Cooking Assistant [8]. Its author conducted a small usability study. According to this, over 80% of the testers thought that speech recognition is beneficial although 50% also observed that the program had difficulties in understanding them—it certainly suffered from the limited speech recognition capabilities on a Laptop PC in the kitchen scenario in 2001. The Cooking Navi [4] was meant to become a multimedia assistant and cooking teacher for beginners and experienced users. Instead of speech, it used video material and added speech balloons to explain cooking instructions. Cooking Navi also found a simplified commercial descendant as Nintendo “Shaberu! DS Oryōri Navi” that reads out loud 200 recipes in Japanese, and features instructional video clips, a cooking timer, ingredients calculator and a very simple speech recognition to turn pages. English versions with the names “Cooking Guide: Can’t Decide What to Eat?”, “Personal Trainer: Cooking” and “Let’s Get Cooking” with 200-300 recipes each appeared in 2008 and 2010, a few more similar sequels appeared in the US, Japan and Europe. They all have in common editorially processed recipes, read out loud by (famous) professionals, and only very limited interaction. However, the commercial success of the Nintendo cooking guide series shows that there is market potential.

More intelligent dialog-based approaches have been investigated in various research projects. CookCoach [5] is a cooking tutoring assistant that supports speech interaction and read out recipes. Its authors observed the necessity to use a domain ontology, which they developed in OntoChef [7]. OntoChef contains very general concepts such as recipe classes, food, part of food, ingredient, unit, action, compound action, task and so on. However, it does not contain an elaborate list of ingredients and their classification. Except the fact that he uses a local speech recognizer, the scenario we have implemented is similar to the one described by [2] for a cooking robot. Although the study work is written in German, its author has implemented a cooking assistant for English speech I/O.

3 Architecture

The architecture of our cooking assistant is depicted in Fig. 1. The cooking assistant code, the app, resides on the mobile device, indicated by the Android logo at the top of the diagram. It contains all the program logic for user interface, speech interaction, and natural language processing to split cooking instructions into sentences, recognized quantities and units of ingredients, etc.

The app requires access to the Internet only for two purposes: (1) **recipe text server**: the recipe collection is hosted on an Apache Solr[4] structured fulltext search server at our premises. Using standard Lucene/Solr queries, it provides fulltext search to find recipes with specific ingredients or categories such as country; (2) **speech recognition**: as mentioned in the introduction, we use the Google Speech API to perform ASR (automatic speech recognition) remotely.

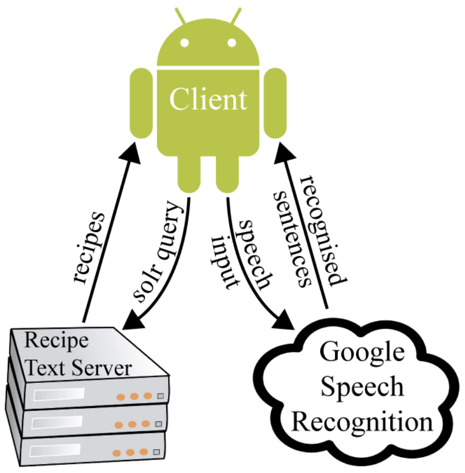
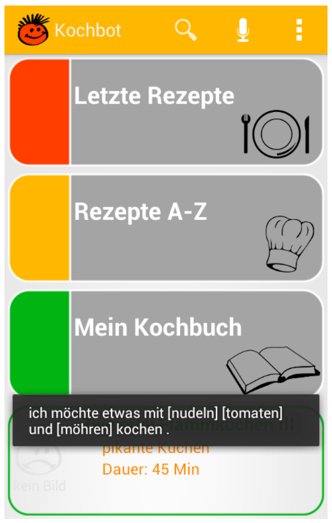


Fig. 1: Architecture overview.



ich möchte etwas mit [nudeln] [tomaten] und [möhren] kochen .

Fig. 2: Main view with recognized speech; identified ingredients are shown in brackets.

[4] http://lucene.apache.org/solr/

4 Offline Pre-processing

In this section, we describe how we pre-process the recipe corpus. Pre-processing serves (1) as input for a static text analysis that e.g. computes what ingredients, quantities, units etc. are, both globally for the complete recipe collection and locally at runtime when a specific recipe is being cooked (Sect. 5.2), (2) as input for filling the recipe text search index.

4.1 Recipe Markup Format

The purpose of our cooking app is to provide assistance that guides the user in the cooking process step-by-step. Therefore, some minimal structuring of recipes is required. We call it lightly stuctured recipe markup. Basically, it contains the recipe title, ingredient lists, and cooking instructions, and optionally some categories such as the country the recipe originates from, or vegetarian, for children, etc. Finer-grained structure such as instruction steps (sentences), units and quantities of ingredients will be 'parsed' by our system on the fly.

Several recipe markup languages based on XML have been proposed, for example RecipeML[5], formerly known as DESSERT, RecipeBook XML[6], CookML[7] and REML[8]. Compared to these partly very rich markup formats, our light markup can be considered as least common denominator to which all other markups could easily be transformed. This also means that we could incorporate other recipe resources with little effort. Of course, parsing of units and quantities as we do it is not necessary in case markup already provides this information in separate fields. Figure 3 contains an example for the target markup of a single recipe. We do not provide the full specification here, as its structure is simple and shallow. The same markup is used to store recipes in the Apache Solr server (Sect. 5.4), which returns found recipes in the same way, plus some additional markup for query-processing related information.

4.2 Creating a Corpus with Recipe Markup

Originally, the recipes texts we used came in HTML format, structured in categories and in German. We downloaded all recipes and transformed the HTML into our light recipe XML format using XPath, extracting the information we needed. We then applied morphological analysis on the recipe texts in order to normalize and abstract from morphological and spelling variants. During this process, we discovered that many recipes in the collection contain OCR (optical character recognition) or typographic errors. We use three layers to standardize and refine the recipes in this offline stage.

[5] `http://www.formatdata.com/recipeml/`
[6] `http://www.happy-monkey.net/recipebook/`
[7] `http://www.kalorio.de/cml/cookml.html`
[8] `http://reml.sourceforge.net/`

```
<doc>
    <field name="category">Teigwaren</field>
    <field name="subcategory">Lasagne, Canneloni und Maultaschen</field>
    <field name="title">Cannelloni al forno</field>
    <field name="ingredients">50 g geräucherter durchwachsener Speck
        100 g gekochter Schinken
        2 mittelgroße Zwiebeln
        1 Zucchini (ca. 200 g)
        1 Möhre (ca. 150g)
        200 g gemischtes Hackfleisch
        125 g Mozzarella-Käse
        250 g Cannelloni (dicke Nudelröhren zum Füllen)
        [...]
    </field>
    <field name="preparation">
       1. Speck und Schinken würfeln. Zwiebeln schälen, Gemüse putzen,
           waschen. Alles fein würfeln. Speck und Hack in 1 EL Öl anbraten.
           Schinken, Gemüse und Hälfte Zwiebeln mitdünsten.  3 gehackte
           Tomaten, 1 EL Mark und Wein zufügen, würzen.</str>
    </field>
    <field name="preparation_time">1 Std 15 Min</field>
    <field name="degree_of_difficulty">normal</field>
    <field name="servings">vier Portionen</field>
    <field name="vegetarian">Nein</field>
</doc>
```

Fig. 3: Example of recipe markup as used in the Apache Solr index.

Step 1: Regular expressions. The first step consists in removing markup and unwanted characters using regular expressions. Furthermore, we separate the cooking instructions from the ingredients list.
Step 2: Morphology. We use the SProUT system[3] with its built-in MMORPH [6] lemmatizer and morphological analysis tool for German to get parts of speech, genus, etc. for every word. We extract a simplified format and suppress all analysis details not needed for our purposes. At the same time, we extract e.g. all possible ingredients along with their number of occurrences in the corpus. We describe in Sect. 5 how this information is used to answer questions on ingredients and their quantity, and in speech recognition at multiple stages to rank ambiguous ASR output. Moreover, we also extract recipe titles which we need for search index preparation.
Step 3: Part-of-speech tagging. To select the most probable morphological reading according to context, we use the trigram-based part-of-speech (PoS) tagger TnT [1]. TnT takes care of the context and delivers a probability for each possible PoS tag. We use it to filter the morphological variants in the MMORPH output by choosing the PoS tag with the highest probability for each word in a recipe which considerably reduces the number of morphological variants.

5 Online System (Cooking Assistant App)

This section describes the core of the cooking assistant app, its user interface with different views for recipe search, step-by-step cooking and reading out mode. We also discuss details such as its ingredients (quantity) parser and the cooking step parser. The app needs Internet connection for speech recognition (Sect. 5.3) and for downloading recipes. Downloading only happens once when a recipe is viewed for the first time. There is no need for downloading again as long as a recipe is stored as one of the last 20 viewed recipes (*Letzte Rezepte*) or marked as a favorite recipe by the user (*Mein Kochbuch*). Apart from speech recognition and downloading recipes, an Internet connection is not required—everything else is processed on the mobile device.

5.1 User Interface

The UI design goal was to make an easy-to-use application which can be controlled using voice commands but also through standard touch interactions. The app consists of several views of which the most important ones will be described in detail below.

Initial view. The first view, which appears upon startup, presents the user with four different buttons (Fig. 2). The topmost one gives access to the last viewed recipes (*Letzte Rezepte*). *Mein Kochbuch* is the user's personal cookbook where favorite recipes can be stored. *Rezepte A-Z* gives the user access to three lists in columns, a list of categories, a list of countries and a list of ingredients from which she or he can choose an item to start a search for that particular category, country or ingredient. The button at the bottom leads to a random recipe that changes on every reload or when shaking the device.

Menubar. Most of the views have an "Action Bar"[9], a menubar on the top of the screen (see top of Fig. 2 and 5). It shows one or more icons that give the user quick access to important functionality. The icons are: a microphone for starting speech recognition, a magnifying glass for opening a search field, and three dots for opening a menu.

Search view. Starting a search, either by using the search field, voice command or the *Rezepte A-Z* view, will take the user to the search view (Fig. 5). The search view shows a list of recipes matching the given search criteria. On top of the list is an area that can be expanded and collapsed. It shows information such a the number of recipes found, search words, the category, or ingredients to occur or not to occur in the recipe. Scrolling to the bottom of the list will load more recipes if available. Choosing one of the recipes will take the user to another view showing an overview over the recipe.

Recipe overview. The recipe overview shows information such as general information about the recipe, a list of ingredients and a list of instructions. At the bottom of the screen, there are two buttons. They both take the user to the same screen, the step-by-step view.

Step-by-step view. The step-by-step view is different from the other views. It uses the full screen and therefore does not have a menubar (Fig. 7). There are two versions of the step-by-step view. The version accessible from the right button is meant as a 'silent preview' of the step-by-step mode, it comes without continuous speech recognition (Sect. 5.3) and the steps are not read out loud.

[9] `http://developer.android.com/guide/topics/ui/actionbar.html`

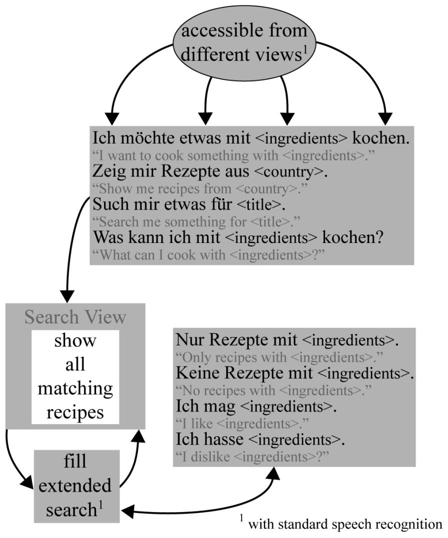
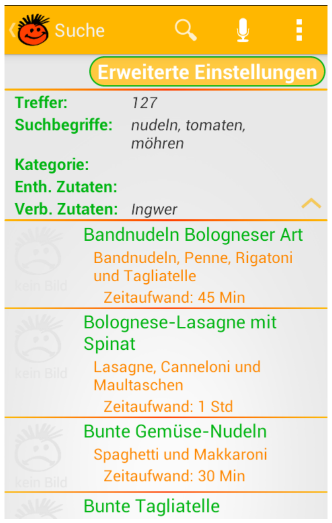

accessible from different views[1]
Ich möchte etwas mit <ingredients> kochen.
"I want to cook something with <ingredients>."
Zeig mir Rezepte aus <country>.
"Show me recipes from <country>."
Such mir etwas für <title>.
"Search me something for <title>."
Was kann ich mit <ingredients> kochen?
"What can I cook with <ingredients>?"
Search View
show all matching recipes
fill extended search[1]
Nur Rezepte mit <ingredients>.
"Only recipes with <ingredients>."
Keine Rezepte mit <ingredients>.
"No recipes with <ingredients>."
Ich mag <ingredients>.
"I like <ingredients>."
Ich hasse <ingredients>.
"I dislike <ingredients>?"
[1] with standard speech recognition

Fig. 4: Dialog for search.



Treffer: 127
Suchbegriffe: nudeln, tomaten, möhren
Kategorie:
Enth. Zutaten:
Verb. Zutaten: Ingwer

Bandnudeln Bolognese Art
Bandnudeln, Penne, Rigatoni und Tagliatelle
Zeitaufwand: 45 Min

Bolognese-Lasagne mit Spinat
Lasagne, Canneloni und Maultaschen
Zeitaufwand: 1 Std

Bunte Gemüse-Nudeln
Spaghetti und Makkaroni
Zeitaufwand: 30 Min

Bunte Tagliatelle

Fig. 5: Search view.

The other version reachable from the left button is the "real" step-by-step view, each step is read out loud and the user can interact with the app by using swipe gestures or voice commands (Sect. 5.3) to go to the next or previous step, or initiate question dialog.

5.2 Recipe Processing

Each recipe XML document (example in Fig. 3) has a number of fields, e.g. the recipe title, a category, a subcategory and possibly a subsubcategory, a field containing the ingredients, a field containing the instructions and some fields for other information such as calories or preparation time.

Ingredients parser. In order to be able to answer questions such as *Wie viele Tomaten brauche ich?* 'How many tomatoes do I need?', the ingredients field needs to be parsed. Normally every line contains one ingredient, consisting of an ingredient name, an amount and a unit (example: *evtl. 2-3 Teel. Öl* 'optionally 2-3 tbs oil'). However, there are many exceptions in the actual recipes. Parsing an ingredient line is therefore divided into different steps to recognize the (partially optional) fields ingredient, quantity and unit.

Step parser. For a hands-free interaction with the app while cooking, the steps are read out loud in the step-by-step view (Sect. 5.1). To be able to do this, sentences in the field *preparation* (Fig. 3) are separated. This is done by splitting the text after each full stop, except if the full stop belongs to a word from a list of stop words including common abbreviations. To give an example, the preparation field from Fig. 3 is divided into 22 steps/sentences (shortened here for space reasons):

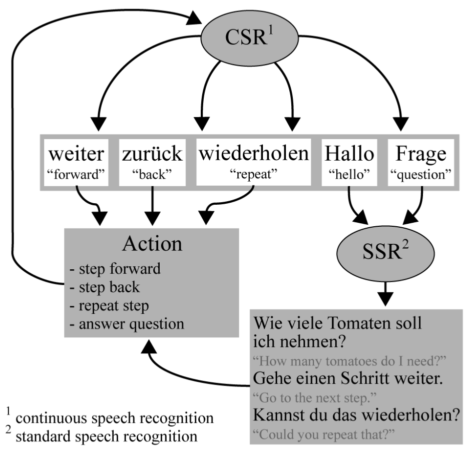


Fig. 6: Command dialog in the step-by-step view.

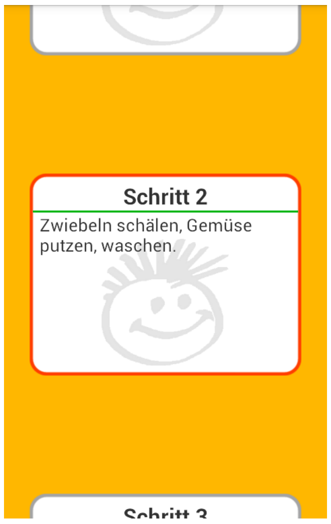


Fig. 7: Step-by-step view.

```
  1. Speck und Schinken würfeln
  2. Zwiebeln schälen, Gemüse putzen, waschen.
  3. Alles fein würfeln.
  4. Speck und Hack in 1 EL Öl anbraten.
  5. Schinken, Gemüse und Hälfte Zwiebeln mitdünsten.
  6. 3 gehackte Tomaten, 1 EL Mark und Wein zufügen, würzen.
[...]
```

5.3 Speech Input

To model user interaction with the cooking assistant, a JavaCC[10] grammar was developed. The grammar is divided into different parts that reflect the interaction stages such as searching, cooking (read out) mode, etc. (Fig. 4, 6, and 8). For speech recognition, we use the Google Speech API in the app. The API offers two different modes. Standard speech recognition (SSR) is used to process full sentences containing questions or commands (Fig. 4). After a speech input pause with a configured threshold length, the input is interpreted as a complete utterance. The second mode, continuous speech recognition (CSR), is used to recognize user interaction in the reading out mode (Fig. 6). Here, only single words are recognized, since there is no defined end of speech.

Standard speech recognition. Before starting speech recognition, the app checks if speech input is available. Then a speech object is created and some options are set, such as the language, the maximum number of recognized sentences to be returned, and whether partial results are to be returned or not. In the standard speech recognition mode, we do not want to receive partial results.

[10] https://javacc.java.net

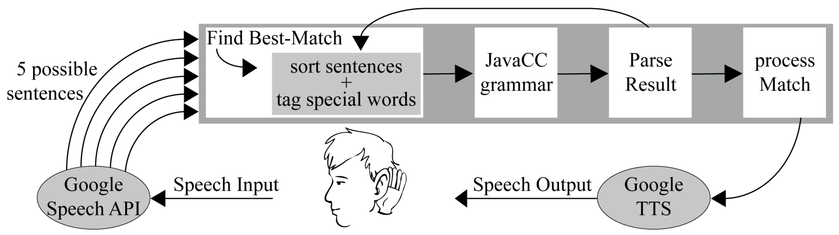


Fig. 8: Speech recognition workflow.

When recognition is started, a window opens and prompts the user for speech input. After silence over threshold time, the window closes automatically and the input is sent to Google ASR.

Continuous speech recognition. The second mode is what we call "continuous speech recognition"[11]. It is important for a hands-free scenario in the kitchen. To initialize CSR, the speech API is initialized in the same way as for SSR, except that it is asked to return partial results. There is no guarantee to receive full sentences anymore or anything useful at all because the speech recognition does not wait for a sentence to be finished but instead just returns results when it sees fit. Everything that is returned by the speech API is split into single words and those are compared with a list of keywords to spot for. Once one of those words is found, the continuous recognition is stopped and some action such as starting the normal recognition is executed. Thus, the continuous recognition is only used to detect single command words as shown in Fig. 6.

Finding the most appropriate ASR match. This applies to the SSR mode (Fig. 4, 8). As mentioned before, we chose the generic Google Speech API for the speech input part of the cooking assistant since it is easy to connect with Java code and is known to be reliable and available on all Android devices. However, the major disadvantage of this speech recognizer is the fact that it cannot be adapted to a specific domain or scenario. Therefore, we decided to establish a rating system that takes into account the current activity where the speech recognition takes place. Additionally, to make word matching more robust, we apply two different stemmers (from Apache Solr/Lucene) concurrently. They work corpus-independently and need only little space and working time.

The general idea is to give a score point for each word in a single match if the activity-specific corpus contains the word. There are three different "scanners" that correspond to three different areas that are checked: complete recipe corpus, current recipe and grammar.

Corpus scanner. This scanner basically uses the general ingredients corpus that was extracted from *all* available recipes. It is used in all kinds of search since we assume that users often will search for ingredients. We use a bundle of four different corpus versions: (1) a stem corpus, extracted using SProUT with the

[11] The "continuous speech recognition" is based on an open source project which can be found at `https://github.com/gast-lib/gast-lib/blob/master/speech.md`

MMORPH stemmer, (2) a string corpus, containing the literal ingredient entries including their morphological variants, (3) a stemmed version of the latter one, stemmed using the 'Lucene' stemmer, (4) the same again, but stemmed using the 'Snowball' stemmer. Moreover, to support search for particular recipes, we also take into account the recipe titles. Thus, we additionally use a title corpus for scoring. The score increases as the title gets longer, i.e. *Spaghetti mit Erbsen-Rahm-Soße und Parmesan* 'Spaghetti with pea-cream-sauce and Parmesan' gets a higher rating than just *Schweinebraten* 'roast pork'.
Example: Assume we would say *Ich möchte Canneloni al forno kochen* 'I want to cook Canneloni al forno'. The corpus scanner would then start a lookup over its corpus bundle and try to find some correspondence and the best possible score. There are two possibilities:

– A rating of +1 for *Canneloni*, which is an ingredient
– A rating of +3 for *Canneloni al forno*, which is a recipe title

The match gets a rating of +3 here since we are looking for the *best* match.
Recipe scanner. Similar to the corpus scanner, the recipe scanner checks for ingredients in a question. However, this time, it is restricted to ingredients that are used in the recipe locally in the step-by-step view. We do not want to take care of ingredients occcurring only in other recipes since questions on those ingredients are irrelevant for the current recipe. Here, the bundle contains only three corpora: (1) the already mentioned recipe text, containing ingredients, (2) a stemmed version of the latter one, stemmed by the 'Lucene' stemmer, (3) the same again, but stemmed by the 'Snowball' stemmer.
Example: Assume we are already cooking and forgot how many zucchinis we need. In this case, we could ask *Wie viele Zucchinis brauche ich?* 'How many zucchinis do I need?'. Afterwards, the scanner would start a look-up and find *Zucchini* as a relevant ingredient and thus increase the match rating by 1. We could also assume that the speech API had returned the potential match *Wie viel Zucker brauche ich?* 'How much sugar do I need?', since both words *Zucchini* and *Zucker* sound similarly in German. This match would get no point, since *Zucker* 'sugar' is an ingredient in fact, but not a relevant one for this recipe.
Grammar scanner. This last scanner is the most important one since it uses the JavaCC grammar belonging to the current recognition for scoring. In a first step, it chooses the appropriate grammar rule that *should* match the current speech input. This matching of the current activity to a grammar rule is hard-coded. When the rule is chosen, we read in the rule and solve all JavaCC tokens that are used and mapped to "real" words. Afterwards, these words are thrown into a bag of words and serve as corpus. This scanner is very important since it simulates an input expectation which is not used in the speech recognition so far, i.e. it partially solves the problem that the Google API is domain-independent. By choosing one rule, we set a special context in which we expect the match to take place in. Tests showed that this step has the highest influence on the general scoring of matches. In contrast to the other scanners, we do not use a stemmer here, since the grammar should take care of different word forms anyway.
Example: Assume that while cooking we forgot how many zucchinis we need.

In this case, we could ask *Wie viele Zucchinis brauche ich?* 'How many zucchinis do I need?'. Afterwards, the scanner would start a look-up in the grammar and find multiple word correspondences in the bag of words for the matching grammar rule: The score should be +4 since the bag of words should contain *wie*, *viel*, *brauche* and *ich*. These three scanners look at every possible match that the ASR is returning and allocate scores to each of them. Afterwards, the matches are sorted according to their rating. The grammar then basically tries to parse the most appropriate match first; if this is not successful, it tries the next one and so on (cf. loop in Fig. 8).

Marking phrases in a sentence. After choosing the best match from speech input as described before, or sorting them according to their relevance, we mark special phrases such as ingredients or recipe titles. The assumption is that these are relevant for search or answering questions. Special markers in the JavaCC grammar use them as pre-terminals. To mark them, we collect all possible titles, ingredients, categories, countries and combinations of the latter that occur in the match. We afterwards choose the most appropriate one out of all possibilities and mark the appropriate parts of the string. *Example:* In *Ich will etwas mit Spaghetti und Tomaten kochen* 'I want to cook something with spaghetti and tomatoes', we would mark *Spaghetti* and *Tomaten* as ingredients: Ich will etwas mit [Spaghetti]$_I$ und [Tomaten]$_I$ kochen (Fig. 2).

5.4 Recipe Text Server

All recipes are stored on an Apache Solr server. An example document is shown in Fig. 3. The searchable fields (i.e. title, ingredients, category or country) are stored in two versions, a stemmed one and a unchanged version. The unchanged version is needed for sorting and searching for categories.

Recipe search. A user search process is divided into different parts. The first step is generating a so called search object. In subsequent user questions (or requests), it can then be altered to fit the user's wishes, e.g. by adding ingredients that have to or must not be contained in the recipe. Each time the search object changes, a new Solr query is constructed from the given information and is sent to the Solr system.

As described earlier, the app has a view where the user can choose a category to search for. Searching for the stemmed category would in some cases lead to a problem. For example, a stemmed search for *torten mit obst* 'tortes with fruit' would match all categories containing one of the three words, except *mit* 'with' which is a stop word, and return not only recipes with the exact same category. To deal with this problem, the search object can be set to search the original fields and not the stemmed ones. Another problem that arose was that a user might want to search for a group of ingredients such as *Nudeln* 'noodles' or *Gemüse* 'vegetables' but that a search for recipes containing the actual word *Nudeln* or *Gemüse* would only return a small number of recipes. We therefore manually extended the Solr server index by a manually curated synonyms list based on the full list of ingredients and categories that was extracted initially. Now, a search for *Nudeln* also returns recipes that contain only the word *Spaghetti*.

5.5 Speech Output

For Text-to-speech (TTS), we use the Google Text-to-speech API. For obvious reasons, this has some limitations. It is for example not possible to modify pronunciation. For domain-specific expressions such as amount range, we therefore change the text string before passing it to Google TTS to pronounce it properly. Out of the box pronunciation of abbreviations (Teel. ‘tbs’, min.), range constructs (‘1–2’) and constructs like ‘5 x 5 cm diameter’ was initially quite bad. We improved it by adding code that replaces abbreviations using regular expressions before sending them to TTS. Here, it was particularly helpful to have a list of abbreviations with their frequencies from ingredient parsing (Sect. 5.2).

6 Summary and Outlook

We have described the components and implementation of a mobile, speech-enabled cooking assistant for 32,000 German recipes. The system is fully implemented and runs stably and fluently. Due to its design with only a lean, well-scaling recipe text server in addition to cloud-based Google ASR, the app can easily be installed on thousands of mobile devices. Future work would extend the approach to utilizing linguistic parsing in both query and answer candidate sentences to further abstract from linguistic variants. Then, more complex question processing (example: “show me recipes where eggs are steamed with milk”) would be possible. Furthermore, integration of sensors and devices such as barcode reader, electronic thermometers, kitchen scales, or stereo camera as further ambient assistance tools could be helpful. Due to the corpus-based approach to recognizing domain keywords, adaptation to further languages and applications such as interactive user manuals or repair guidance systems should be easy.

References

1. Brants, T.: TnT – A statistical part-of-speech tagger. In: Proc. of 6th ANLP. pp. 224–231. Seattle, Washington (2000)
2. Chouambe, L.C.: Dynamische Vokabularerweiterung für ein grammatikbasiertes Dialogsystem durch Online-Ressourcen (2006), Studienarbeit, University of Karlsruhe
3. Drozdzynski, W., Krieger, H.U., Piskorski, J., Schäfer, U., Xu, F.: Shallow processing with unification and typed feature structures — Foundations and applications. Künstliche Intelligenz 1, 17–23 (2004)
4. Hamada, R., Okabe, J., Ide, I.: Cooking navi: Assistant for daily cooking in kitchen. In: Proc. of 13th ACM Int. Conf. on Multimedia. pp. 371–374. Singapore (2005)
5. Martins, F.M., Pardal, J.P., Franqueira, L., Arez, P., Mamede, N.J.: Starting to cook a tutoring dialogue system. In: SLT Workshop, 2008. IEEE. pp. 145–148 (2008)
6. Petitpierre, D., Russell, G.: MMORPH – the Multext morphology program. Tech. rep., ISSCO, University of Geneva (1995)
7. Ribeiro, R., Batista, F., Pardal, J.P., Mamede, N.J., Pinto, H.S.: Cooking an ontology. In: Euzenat, J., Domingue, J. (eds.) Artificial Intelligence: Methodology, Systems, and Applications, LNCS, vol. 4183, pp. 213–221. Springer (2006)
8. Wasinger, R.: Dialog-based user interfaces featuring a home cooking assistant (2001), unpublished manuscript, University of Sydney, Australia