Application of Semantic Web Technology: Recipe Finder

Project Report

presented by
Jing Sun, Kuan Min Chen, Michael Monych

submitted to the
Data and Web Science Group
Prof. Dr. Heiko Paulheim
University of Mannheim

30.11.2018

1 Introduction

"Throw-away society" is a term often used to describe today's society. The German center for nutrition estimates, that every year eleven million tonnes of food is thrown away in Germany alone [1]. For the U.S., the figures are even worse: 150,000 tonnes of food is thrown away every day [2].

The biggest reason why ingredients are thrown away is that the quantity initially bought exceeded the actually required quantity for a given period of time. This might be either because of a special promotion, the product not being available in smaller units or just due to a lack of planning for the purchase [8]. One way to reduce some of that waste would be to provide a way to use leftover food.

In our project, we achieved just that, by providing households with an easy way to find recipes incorporating food they need to either consume today or throw away tomorrow. We tapped into the power of the Semantic Web and developed an application which allows its users to browse recipes based on leftovers they might have in their kitchen, ultimately reducing food waste.

2 Application domain and goals

Our application is helpful for cooking beginners. As for cooking beginners, they don't know which ingredients they need for dishes. They sometimes buy too many or too few ingredients. This application is also targeted at Mensa and restaurants. These institutions need to buy a large number of ingredients every day. The number of people dining is uncertain. The ingredients are not always used up. Therefore, how to use the unused ingredients to cook another dish is an interesting use-case for them.

Our goal is to help users to manage the left-over ingredients. Among all the possible ways to achieve the goal, we focus on:
1. How to complete a dish with minimum needs for extra ingredients;
2. How to complete a dish with maximum usage of existing ingredients.

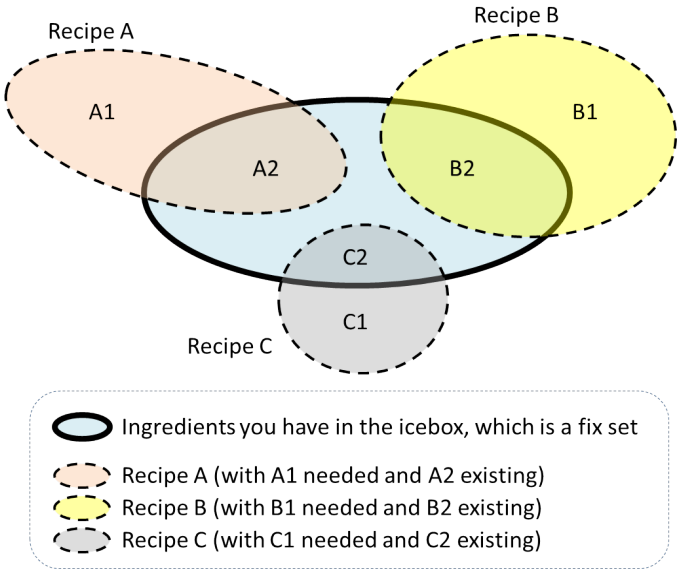


Figure 1: Intersections with Existing and Needed Ingredients

Our application requires the user to input their wanted and/or unwanted ingredients. Wanted ingredients are supposed to get what kinds of food are left in users refrigerators. Unwanted ingredients are supposed to get what kinds of food users don't want to appear in the recipes. After getting the input information, our application will generate SPARQL queries automatically and send queries to a fuseki instance, which serves a SPARQL endpoint for our dataset, and look for the correspondences in the dataset. Finally, the titles of relevant recipes and ingredients will be returned for the user to see.

3 Datasets

To realize this project, we used the Foodista dataset export. It is based on a crawl of a previous version of the website `www.foodista.com` (foodista). The website's layout has been changed in such a way, that crawling recipe data is no longer feasible. The original host, Kasabi, was shut down as well [10] so that we are relying on the version of the dataset uploaded to the internet archive.

```
<http://.../food/222YXBYC> <http://purl.org/dc/terms/description>
    "A Cherry Native To ..." .
<http://.../food/222YXBYC> <http://.../22-rdf-syntax-ns#type>
    <http://linkedrecipes.org/schema/Food>.
<http://.../food/222YXBYC> <http://.../rdf-schema#label>
    "Catalina Island Cherry".
<http://.../food/222YXBYC> <http://.../foaf/0.1/depiction>
    "http://.../content/misc/placeholders/food_big".
<http://.../food/222YXBYC> <http://.../foaf/0.1/isPrimaryTopicOf>
    <http://.../food/222YXBYC/catalina-island-cherry>.
<http://.../food/223N2653> <http://.../dc/terms/description>
```

Figure 2: Fraction of the dataset

The plain format of the triples are all presented in a full form, which are URLs, as shown above. Every subject is presented as an ID, such as "CTHGZTN7" and "PKG4QCWJ", which are random combinations of letters and numbers. And all the predicates in the dataset are shown as below. They can be interpreted as "`isTypeOf`" (reciple or food), "`hasIngredientId`" (ingredientId), "`hasTitle`" (name of the recipe or name of the food), "`hasLable`" (name of the ingredient), and "`hasDescription`" (description of the recipe and the food), and so on.

```
<http://xmlns.com/foaf/0.1/isPrimaryTopicOf>
<http://www.w3.org/2004/02/skos/core#altLabel>
<http://linkedrecipes.org/schema/uses>
<http://purl.org/dc/terms/related>
<http://linkedrecipes.org/schema/servings>
<http://www.w3.org/2004/02/skos/core#prefLabel>
<http://xmlns.com/foaf/0.1/primaryTopic>
<http://purl.org/dc/terms/title>
<http://linkedrecipes.org/schema/ingredient>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://linkedrecipes.org/schema/category>
<http://xmlns.com/foaf/0.1/depiction>
<http://purl.org/dc/terms/description>
```

Figure 3: Predicates in Foodista dataset

We loaded the foodista data into Protege. There are descriptions and pictures showed in some recipes. We found the subject of food, category, ingredients and so on. We accessed to the category and ingredients by defining the properties like category and ingredient or rdfs:label. We found there are 204 recipes, 2036 ingredients and 7296 categories in total in this dataset.

The categories are not only cuisines of different countries, but also the categories of main ingredients. For example, the "222K34KT" is labeled as meats, Mexico, soups, hamburger, beef and so on. Therefore, users could

choose multiple categories to get the recommend results as each recipe has multiple labels. However, we decided as the categories are not of the same domain (and due to time constraints), it will be less confusing to leave out filtering recipes based on category.

We have also checked the data of DBpedia using SPARQL. There are not many recipes in DBpedia and most of the categorization of these recipes is either missing or very inconsistent. Therefore, we decided to focus on foodista in the end.

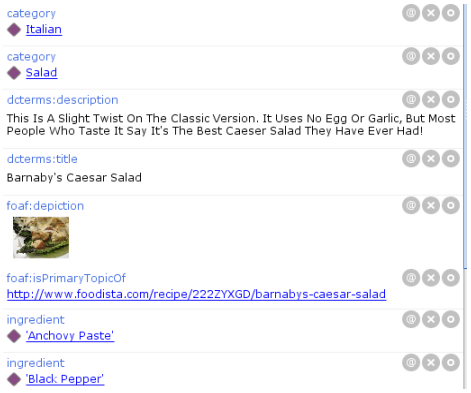


Figure 4: Example of one Recipe of Foodista

4 Techniques

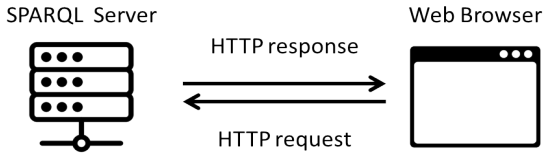


Figure 5: Architecture of Application

- **The front-end**

  First, as our IDE, we used Visual Studio Code (VSCode) to create the HTML and JavaScript files. It's a simple, light-weighted, and cross-platform tool for coding. HTML and JavaScript are the 2 main

languages we used to build the front-end web page. In our JavaScript, we use the JQuery library to manipulate the user interface. Plain JavaScript is used, to build the SPARQL-query to get the right information from the back-end. With the power of JQuery, we can efficiently manage the sending and receiving process. Moreover, it allows us to retrieve and organize the information, and conveniently produce outputs in the format that we want our client to see. With further help of the basic CSS libraries provided by Bootstrap (which are completely cosmetic), we set up the the interface for users to search for their recipes based on wanted/unwanted ingredients.

- **The back-end**

  For the "server", we used Apache Jena Fuseki. It is a SPARQL server. It can run as an operating system service, as a Java web application, and as a standalone server [6]. We used Fuseki as the server of our application because it fit our use case perfectly. Writing a back-end which accepts SPARQL queries and returns results would have likely exceeded our capacity. We ran a Fuseki server in our machine, then uploaded the foodista data on to this server. The requirements of users are written into SPARQL queries and sent to Fuseki by an HTTP "post" request. After that, Fuseki processes the query and returns the results as a http-response. The response is then handled by the front-end to display the results in an intuitive manner.

- **SPARQL queries**

  To complete the very 2 tasks we have set up at the beginning, we have to build a proper query so as to attain the idea results. We use basic query as "?recipeId isTypeOf ?ingredientId" and "?ingredientId hasLable ?nameOfIngredient" to find out those fitted recipes which include the wanted ingredient. Finally, we used the query below to show up those extra needed ingredients.

  To manage the unwanted ingredients named by the users, we use "FILTER NOT EXIST" and "VALUES ?lable "A" "B" " (A and B are the unwanted ingredients). With this method, we can then rank the number of the extra needed ingredients, so as complete the first task.

  Example SPARQL for the first task:

```
SELECT DISTINCT ?title
(COUNT (?title) as ?count)
(GROUP_CONCAT(DISTINCT ?Lable; SEPARATOR=",")as ?ingredients)
WHERE
{
    ?itemId :isTypeOf :recipe.
    ?itemId :hasIngredient ?ingredientId.
    ?ingredientId :hasLabel ?Lable.
    FILTER NOT EXISTS{
        VALUES ?Lable { "A" "B"}
        ?itemId :hasTitle ?title.
        ?itemId :isTypeOf :recipe.
        ?itemId :hasIngredient ?ingredientId.
        ?ingredientId :hasLabel ?Lable.
    }
}
GROUP BY ?title ORDER BY (?count)
```

Figure 6: Build SPARQL Queries in Application

For the second task, we used 'VALUES ?Label {"A" "B" }' to help us to find all the recipes which have at least one of the wanted ingredients. And along with the same method to get rid of the unwanted ingredient, we then complete the second task.

5 Example Results

In our application, we provide the user with an interface, where they can type in any leftover they still have in their household as wanted ingredients, and offer a choice of recipes requiring these ingredients. Additionally, they will also be able to put in the ingredients they don't want, so the recipes with the unwanted items will also be excluded from the outcomes as well. For example, a vegan can type in "Pork Chop" or "Lamb".

Moving on, the user can select either they want to minimize the number of extra ingredients they need to buy to complete any dish, or to maximize the number of existing ingredients they have in storage. For the prior purpose, they can click on "Buy less extra!", and if it's for the later purpose, they can click on "Use up more!".

Figure 7: User Interface (UI)

At the outcome, the users have:

Name: the name recipes.

Owned: the number of existing ingredients they have and also as one of the components of the dishes.

Missing: the number of extra ingredients the user have to buy in order to complete the corresponding dishes.

Ingredients: the complete list of ingredients needed for the corresponding dishes, with the names of the already owned ingredients in bold , and the names of the missing ingredients not in bold.

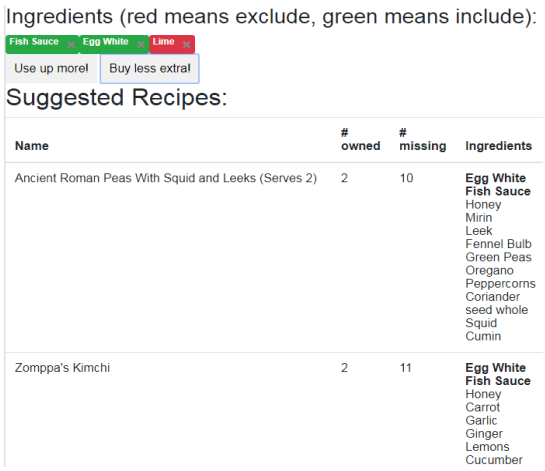


Figure 8: An Example Results

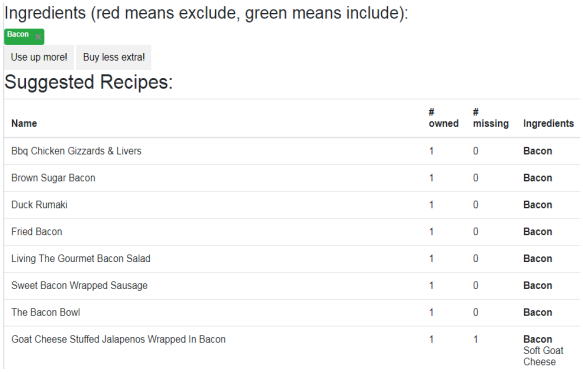
In the example shown above, the user enter "Fish Sauce" and "Egg

White" as wanted ingredients, and "Lime" as unwanted ingredients. And then the user click on "Buy less extra!". And there are dishes such as "Ancient Roman Peas With Squid and Leeks" and "Zomppa's Kimchi" and other dishes take on "Fish Sauce" and "Egg White" and some more ingredients, and don't have "Lime" as one of the components. And for the first dish, you have to buy 10 more extra ingredients, and 11 more extra ingredients for the second dish, so as that you can complete the dishes. Along with it. those extra the user need to have are named in the last column, which are those labels not in bold.

6 Evaluaion

In the the project outline, the evaluation of the results had been brought up. But after looking into the dataset, we found out some problems with the information provided by the dataset. For example, as the result shown below, we put "Bacon" as the wanted ingredient,and the outcomes includes some dishes which apparently requires ingredients more than just bacon. However, in the table, some ingredients are missing in the last column. For example, "Duck Rumaki" and "Bbq Chicken Gizzards Livers" needs duck, chicken, or many more other ingredients to complete the dishes, and according to our dataset, they both only needs "Bacon".

In the project outline, we mentioned that we can compare the result we have to the result from the website. But, based on the situation we have, it is seemingly less meaningful to do the comparison. At the end, we decided not to do that.


Ingredients (red means exclude, green means include):

Bacon

Use up more! Buy less extra!

Suggested Recipes:

Name	# owned	# missing	Ingredients
Bbq Chicken Gizzards & Livers	1	0	**Bacon**
Brown Sugar Bacon	1	0	**Bacon**
Duck Rumaki	1	0	**Bacon**
Fried Bacon	1	0	**Bacon**
Living The Gourmet Bacon Salad	1	0	**Bacon**
Sweet Bacon Wrapped Sausage	1	0	**Bacon**
The Bacon Bowl	1	0	**Bacon**
Goat Cheese Stuffed Jalapenos Wrapped In Bacon	1	1	**Bacon** Soft Goat Cheese

Figure 9: Issues with the dataset

7 Known Limitations

There are several limitations to our application.

- Our application cannot show the amount of each kind of ingredients users should use. Because in our foodista ontology, there are no values of the amount of each ingredient. We should merge our foodista dataset with other ontologies to get more information if we have more time.
- Our application cannot show the methods and tools for cooking the dishes. This information is included in our foodista ontology. we can implement these function if we have more time.
- Our application cannot query something that does not appears in the foodista ontology. As we cannot know all the thing on the web. There is more information that related to recourse that we don't know. This is the "Open World Assumption". For example, how can we give the accuracy feedbacks if the user inputs an wanted ingredient which is not included in the foodista. What we can do is extend our datasets and include resources and values as more as possible,
- Our application cannot distinguish the same ingredients but typed in different names or languages. For example, it will give correct feedback of "Potato" but cannot query "Kartoffel". The is the Non-unique Name Assumption.

8 Lessons Learned

8.1 Challenges

- to fully understand the users' needs, In order to actually help users to deal with their left overs, we need to step into their shoes to think about which way to present the information is the most helpful. For example, at the last column of the result, we decide to not just put in the name of the missing ingredients (names not in bold), but the whole list of ingredients required (including names in bold). In this way, the information is more complete, and also the user can use the application as just a regular books or websites where the full list of the ingredients are presented.

- to fully understand the capability and limits the tool we use have, Before we get our hands on the query, the understanding of what the tool can and cannot do is very important. For example, as what we learned in the lecture, the "Open World Assumption", "Not Unique name", and "Anyone can say Anything about Any topic" play huge roles. We must know that the information we have is possibly not completed, such as the ingredients shown in certain recipe, in real world or with different standards, are not being able to make the dish. And also the correctness is also need to be concern with, such as errors like miss typing while producing and processing the data.

- to properly translate the needs to the language the tool uses, Now, assuming that we have mastered both the needs and the tool. Some problem might occur during the transition. One situation can be that we will come to truly realize or understand what the limits of the tools during the transitioning. The reason is that there sometimes are blind spot when we look into the needs and the tools individually, and the missing connections between these two will emerge as the works involves more from two sides.

  There are two biggest obstacles we druing the project. First is to build up the function of "Use up more!". Before getting into it, we thought the realization of it is easy, but it took us more time than we expected before we finally come up with the solution. The combination of the use of "FILTER NOT EXISTS" and the "VALUES ?lable "A" "B" " can be many way with different order. In one order, we can achieve "Buy less extra!", and "Use up more!" in the other order. However, We had difficulty to find the right combination to successfully produce the result we were looking for. After many try-and-errors, we got the idea result and presented them with the consideration over users' preferences.

  The other part we also spent much time on is the JavaScript. JavaScript is a really powerful tool for the front-end work. And it is also the tool that most of the team members were not familiar with at the beginning. During the transitioning, we had also encountered issues related to how to use the JQuery library to output the result from SPARQL server. The questions such as what information should be presented in what way, and what comments in JQuery we can utilize to realize it.

8.2 Review

After getting through these 2 most difficult issues, we then have been able to complete the 2 task we set up earlier, and ideally present the results. some possible reviews about this projects will be as follow.

- To add more function to the interface Although the application we have now can offer 2 comparatively more useful informations, still there can be more funcitons to facilitate the searching process. For example, we can make filters at the final outcome for users to have it ranked by the number of owned ingredients or missing ingredients.
- To add more datasets In our application, we only apply one dataset. It is mainly because the resources we have reached to have some deficiency, and somehow cannot be complement to the dataset we have. This leaves us out that we don't have chance to experience the actual working process of integrating datasets. We will surely learn more if we have chance to do that.
- To apply reasoning function With the goal we set, we simply need to use search function from the SPARQL query to accomplish. And during the class, the reasoning part can be applied to cases where we store the users' browsing records, and use those records to come up with a new list as for recommendation.

Bibliography

[1] BZfE, *Ernährungsweise und Lebensmittelabfälle in Familienhaushalten*, available online under: https://www.bzfe.de/_data/files/eif_2016_03-04_ernaehrungsweise_lebensmittelabfaelle.pdf, last checked 2018-10-13

[2] Oliver Milman, *Americans waste 150,000 tons of food each day equal to a pound per person*, The Guardian, available online under: https://www.theguardian.com/environment/2018/apr/18/americans-waste-food-fruit-vegetables-study,
last checked 2018-10-13

[3] Wikipedia, available online under: https://en.wikipedia.org/wiki/SPARQL#cite_note-3,
last checked 2018-11-25

[4] Apache Jena, available online under: https://jena.apache.org/documentation/query/index.html,
last checked 2018-11-25

[5] Apache Jena, available online under: https://jena.apache.org/documentation/query/index.html,
last checked 2018-11-25

[6] Apache Jena, available online under: https://jena.apache.org/documentation/fuseki2/index.html,
last checked 2018-11-25

[7] Wikipedia, available online under: https://en.wikipedia.org/wiki/HTML,
last checked 2018-11-25

[8] GfK SE, *SYSTEMATISCHE ERFASSUNG VON LEBENSMITTELABFLLEN DER PRIVATEN HAUSHALTE IN DEUTSCHLAND*, available online under: `https://www.zugutfuerdietonne.de/fileadmin/Neuigkeiten/PDF-Dateien/Studie_GfKBMEL.pdf`, last checked 2018-10-13

[9] Chefkoch Resteverwertung, available online under: `https://www.chefkoch.de/rezept-reste.php`, last checked 2018-10-13

[10] Mannheim Linked Data Catalog, University of Mannheim, Foodista, available online under: `http://linkeddatacatalog.dws.informatik.uni-mannheim.de/de/dataset/foodista`, last checked 2018-10-13

[11] Brank, Janez and Grobelnik, Marko and Mladenić, Dunja. A survey of ontology evaluation techniques, 2005

[12] Vrandečić, Denny. Ontology evaluation, Handbook on ontologies, Springer, 293–313, 2009,

[13] Allgemeine Geschäftsbedingungen, available online under: `https://www.chefkoch.de/terms-of-use.php`, last checked: 2018-10-14