AN ONTOLOGY-BASED RESTAURANT INFORMATION SYSTEM

A REPORT SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF BACHELOR OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

2023

Thomas Hayter

Department of Computer Science

Contents

Word Count: 4837

List of Tables

List of Figures

Abstract

AN ONTOLOGY-BASED RESTAURANT INFORMATION SYSTEM
Thomas Hayter
A report submitted to The University of Manchester
for the degree of Bachelor of Science, 2023

The aim of the thesis is to ...

Declaration

No portion of the work referred to in this report has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

Acknowledgements

First of all, I would like to thank my supervisor who has guided me through the process of this third year project. They have constantly kept me on track, reassuring me when I needed it while also emphasizing the urgency of the project. I would also like to thank all of the participants of my user studies, who picked out so many problems that I missed and whose feedback drastically improved the system in many ways.

Chapter 1

Introduction

1.1 Overview

My project, *"An ontology-based restaurant information system"*, explores how ontologies can be used to store information about a restaurant's menu. In a world where over half of the UK population are eating out at restaurants and fast food chains multiple times a month [20], and a far greater emphasis is being placed on dietary requirements, it is vital that food vendors are able to provide such information to customers. While some customers have specific diets such as vegetarianism and veganism, others have specific allergies or are simply looking to reduce their calorific intake. Religious diets also play a huge part in what we can and cannot eat. Therefore, the restaurant staff want to be able to provide sufficient information to customers about what dishes are available to them based on their requirements. There is also the aspect of modifying dishes. For example, *"Is a plant burger with extra bacon still suitable for vegetarians?"* and *"How many calories are in a cheese burger without the cheese?"*.

In this project I will explore to what extent it is possible to store the necessary information from a menu in an ontology. For example, whether you can store calorific information about ingredients, calculate dishes suitable for certain diets and calculate new information about a dish with certain ingredients removed or extra ingredients added. I will also compare different ways of storing the same information and find the method that is both simple and effective. I will then evaluate whether an ontology can be used as a way to store information in a system that is used by those who are not familiar with ontologies and how they work.

The final product will consist of two user interfaces, the ontology itself, and the code to link the two together. The first user interface will be targeted at the customer

of the restaurant. They will need to view and filter the menu based on their dietary requirements. They will also be able to modify dishes by adding or removing ingredients. Then, they will be able to see the updated information for the modified dish. The second will be aimed at the owner of the restaurant. They need to be able to create and edit the menu for the customers.

1.2 This Report

This report is broken down into five sections. In the remaining part of the introduction chapter I will describe the planning of the project. The second chapter focuses on the background to the project, including information about diets, technologies that I have used and how ontologies work.

The third chapter discusses all of my design choices for the project and evaluation. It details the methodology used to create the final system, and also lists my requirements and success criteria for the project. The fourth chapter is where I describe the final product and describe the results of the user studies that I conducted. The final chapter evaluates the project as a whole, discusses whether I met my objectives, what I could have done better and explores where this project could be developed further.

1.3 Project Plan

The project ran from September 2022 until April 2023. I broke the project down into multiple stages: research, development and testing. I would spend the first few months researching how ontologies work and following tutorials to understand how different technologies works. This would include getting some hands on experience by using the technologies myself to make sure I truly understood how they worked. Then I would design the system that I wanted to create, and develop a minimal viable product that fits this. Then I would pad the system out with all the required features. The next step was to run a pilot test, to recieve some early feedback about the system. Then, in the final weeks of development, I will finish the system and make changes based on the feedback from the pilot test. Once it is finished, I will conduct another user study to evaluate the success of the system.

Chapter 2

Background

2.1 Diets

After suffering from the COVID-19 pandemic, restaurants are keen to get as much business as possible. To do this, they need to cater to all customers and all dietary requirements. To achieve that, they need to easily allow customers to calculate which dishes are suitable for them.

Almost 30% of adults in the UK have some sort of vegetarian-based diet [10], and the percentage is greater in the younger generations than the older ones. With an increasing concern about the environmental, ethical and health impacts of the food that we eat - people are looking at reducing and removing meat from their diets (Flexitarian or Vegetarian), or removing all animal products completely (Vegan).

On top of this, 8% of children and 6% of adults will have an allergy [18]. The most common allergies are cow's milk, eggs, peanuts, soybeans, wheat, fish, shellfish and tree nuts [19]. Allergic reactions can vary from mild to severe, and therefore, it is important that a restaurant is able to offer allergen information about their dishes. Moreover, while Coeliacs disease is not an allergy but a autoimmune disease, those who suffer from the disease want to avoid foods that contain gluten.

With 63% of people in the UK stating they have some religion [16], and 84% of the population worldwide [6], it is vital to consider religious dietary requirements. I have used this guide [7] to understand what dietary requirements different religions have. This means that least 45% of the global population will have a dietary requirement purely based on their religous belief, which makes it essential that they are considered. While some religions simply follow or encourage vegetarianism or veganism (such as Buddhism), others are more complex. Hinduism and Sikhism will specifically not eat

beef or beef products. Furthermore, Muslims, Jews, and Rastafarians will not eat pork or pork products, with the former two not eating shellfish either. Therefore, specfic measures should be taken to ensure that these customers can determine which dishes are suitable for them.

In addition to specific ingredients, there are other religious beliefs concerning food. Muslims are only permitted to eat Halal (permissable) foods, or foods that are not Haram (forbidden). Halal meats must come from an animal that has been slaughtered in a specific way according to the Halal Monitoring Committee [2]. The animal must be alive and healthy at the time of slaughter, and after invoking the name of Allah on the animal, at least three of the Trachea, Oesophagus and both Jugular Viens must be severed. In Judaism, followers may only eat Kosher foods [12]. This specifies that they must not eat dairy and meat products together, and the two should not be prepared together either to avoid cross-contamination. Meat should also be sourced from an animal that has been slaughtered in a specific way.

Both Halal and Kosher diets do not solely revolve around what ingredients are in a dish, but also how that dish has been prepared. It should also be noted that not all religions are strict. Some diets are stricter than others, so there is not one single catch-all diet for a religion.

2.2 Ontologies

When researching ontologies, I constantly referred to the pizza tutorial [4]. It provides a comprehensive understanding of ontologies and how they work, while also providing examples and a tutorial to follow along with.

An ontology is a knowledge-based way of storing information. They are stored in a *.owl* file, which is similar to an XML. It works in a object oriented kind-of way, where every thing is a class. Ontologies make use of the concept of inheritance, so classes can have subclasses where the subclass *"is a"* superclass. For example, the class *"House"* could be a subclass of *"Building"*, and could have *"Mansion"*, *"Bungalow"* and *"Cottage"* as it's subclasses. This works because a house *"is a"* building and a cottage *"is a"* house. Everything that holds true for a superclass also holds true for all the subclasses. In an ontology, every class is a subclass of *"Thing"*. It is convention to name a subclass with a suffix to make it clear what it is a subclass of (other than when creating a subclass of *"Thing"*). In the previous example, all subclasses of *"House"* would

have the suffix *"House"* (e.g. *"CottageHouse"* and *"BungalowHouse"*) to differentiate them from anything else that could have the same name. For example, having one class called *"Cottage"* could lead to ambiguity between the building and the type of cheese - if both happened to be stored in the same ontology.

Ontologies contain facts, properties and rules, and uses those rules and properties to infer more facts. There is a tool called a reasoner which infers these facts. While many are available, I have used Hermit [13] as it is the most up to date.

2.2.1 Object and Data Properties

There are two kinds of properties: object properties and data properties. An object property relates two classes. The property can have a domain and a range class, which includes the class and all of it's subclasses. Following on from the previous example, the object property *"hasRoom"* could have the domain *"House"* and the range *"Room"*. Specifically, you could represent a cottage having a kitchen with: *"CottageHouse hasRoom KitchenRoom"*. A data property links a class to a literal value, such as an integer, boolean value or string. They can also have a domain class and a range data type (integer, string etc.). For example, the data property *"hasNumberOfWindows"* could be used to link a *"House"* class to an integer value of how many windows the house has. You could represent a bungalow having 3 windows as: *"BungalowHouse hasNumberOfWindows 3"*.

Both these properties can have many characteristics. These properties are also stored in an object oriented fashion, so if the *"hasKitchen"* property was a subproperty of *"hasRoom"*, and it is stated that *CottageHouse hasKitchen ModernKitchen*, then the reasoner would be able to infer that *CottageHouse hasRoom ModernKitchen*. Properties can be transitive: if **X** is related to **Y** via **PropertyA**, and **Y** is related to **Z** via **PropertyA**, then **X** is related to **Z** via **PropertyA**. They can also be functional. This means that there is at most one object that is related to an object via that property. Therefore, if two objects are related to an object via a functional property, then it will be reasoned that they must be the same object. An example of a functional property would be *"hasNationalInsuranceNumber"*, because people can only have one national insurance number. Properties can also have inverses. If **X** is related to **Y** via **PropertyA** and **PropertyB** is the opposite of **PropertyA**, the **Y** is related to **X** via **PropertyB**. The domain of **PropertyA** will be the range of **PropertyB** and the range of **PropertyA** will be the domain of **PropertyB**. A property can be symmetric. If **X** is related to **Y** via **PropertyA**, then **Y** is related to **X** via **PropertyA**. An example of that

would be *"hasNeighbour"*. Properties can also be asymmetric, which means they can never be symmetric. Properties can also be reflexive, meaning they always relate an individual to itself. They can also be irreflexive, which means an individual can never be related to itself.

2.2.2 Defined Classes

Defined Classes are classes that contain rules. For example, a *"VegetarianDish"* could be defined as a dish that contains no *"MeatIngredient"*. In an ontology, you would say that a *"VegetarianDish"* is a subclass of *'Dish'* and a subclass of *"not(containsIngredient some MeatIngredient)"*.

2.2.3 Instances

Instances are just that, an instance of a class. You can have many instances of a class, each with different object properties and data properties. This way, you could create two instances of a bungalow, but with different numbers of bedrooms and different addresses. Instances are useful when you want to have many occurences of a class that are slightly different.

2.2.4 Enumerations

Enumerations are also a concept from object oriented programming that is utilised in ontologies. When a property can only take one of a few possible values, it can be convenient to create an enumeration - which explicitly defines those few possible values. An example would be the seasons, which can only be winter, spring, summer or autumn.

There are other aspects of ontologies that are not used in this project, such as annotations and DL and SPARL querys.

2.2.5 Open World Assumption

Ontologies have an open world assumption, so all constraints on classes must be declared. It is not sufficient to say that a dish contains some ingredients, but also that it contains only these ingredients. Otherwise, a dish with those ingredients plus more would be valid. This is important when calculating things like a vegetarian dish. Take the example of a Margherita pizza, which contains a pizza base, tomato sauce and

cheese. If the ontology only states that it contains at least these ingredients, then while a Margherita pizza would be classed as a margherita pizza, so would a meat feast pizza that contains those three ingredients plus more. It is for this reason why the reasoner cannot deduce that a margherita pizza is a vegetarian dish, unless the ontology states that a margherita can only contain those three ingredients.

2.3 Technologies

I initially used Protégé to get used to working with ontologies. The user interfaces are written in Java as it has the useful and comprehensive Web Ontology Language (OWL) Application Programming Interface (API) [3], which can be used to interact with ontologies. I used Java Swing to create the user interfaces themselves. The project is built using Maven and produces a runnable .jar file that should be placed in the same folder as the .owl file.

2.3.1 Protégé

Protégé [5] is a free, open-source software interface for interacting with ontologies. The workspace is customizable and makes it easy to navigate ontologies and the relationships within them. It also supports the querying of ontologies.

I used Protégé throughout my development to monitor my menu ontology. When my user interfaces made changes to the ontology, I used Protégé to ensure the expected changes had been made to the ontology. I also employed Protégé to help me debug problems, or manually fix the ontology if one of my user interfaces broke it.

2.3.2 The OWL API

The OWL API is a Java API that allows you to create and manage OWL ontologies. It can be installed to a project using maven. I followed the following tutorial [9] to begin to get to grips with the OWL API, along with the documentation that is available here [14]. I chose to use this API because I am familiar with Java and find it much easier to use than Python when working with new, less documented APIs as it is clear what types are being returned from methods. On top of this, it has been developed by The University of Manchester, so It would be easier to find someone to answer my questions.

2.3.3 Java Swing

Java Swing [15] is a Graphical User Interface (GUI) widget library for Java. It contains many lightweight components that can be utilised and customized easily. It provides a familiar and intuitive aesthetic to components that make users instictively know what to do with them. The library is platform independant, which means GUIs developed using it are portable to many devices.

I chose Java Swing because it has a lot of components already set up that can be used. JFrames are the windows that contain JPanels (a view). JPanels have a layout, and you can add many types of Components to display, such as a JTextField which allows the user to enter text, a JButton which makes actions for the user to take obvious, JLists to display data and JCheckboxes to select specific options. A JSlider can be used to choose a particular value from a set range, and a JOptionPane which displays a message in a pop-up window - which is useful for giving informative feedback. They also allow for clear consistency between objects of similar types.

Chapter 3

Design & Methodology

3.1 Requirements

To evaluate the system, I have come up with a list of functional and non-functional requirements that the system should comply to. These are broken down into those which are compulsory, and those which would be good to include if time permits.

3.1.1 Functional Requirements

Functional requirements for the system:

- The system needs to be able to store all dishes from the menu of a restaurant, along with the ingredients that make up that dish.
- The system should be able to produce a list of these dishes to the customer.
- The owner of a restaurant should be able to add ingredients and dishes to the system.
- A customer needs to be able to query the system for dishes they desire.
- Therefore, the system should filter the dishes by various dietry requirements, such as diets, allergies and filter by calories.
- The system should have the option to hide calorie information to those who do not want to see it.

Possible additions to the system:

- The system could filter dishes by how a meal has been prepared and cooked.
- The system could calculate new information for a dish based on whether a customer would like to add or remove an ingredient.
- The system could allow for a restaurant owner to register new allergens and diets to filter by.

3.1.2 Non-Functional Requirements

The non-functional requirements of the project are that it must:

- The system must be intuitive, easy to use, and provide appropriate feedback when changes are made to the system.
- The interface must conform to Ben Shneiderman's 8 golden rules [17].
- A restaurant owner should require minimal training to use the system.
- A customer should need no training to use the system.
- The system should allow customers to search for dishes within an appropriate response time. Performance is important as the restaurant and the customer will want the customer to make a decision quickly on what dish they would like to order.

3.1.3 Filters

Dietary Requirements to filter by:

- Vegetarian
- Vegan
- Pescetarian
- Religious diets e.g. Halal / Kosher preparation, Hindu and Sikh.

The most common allergies [1]:

- Peanut

- Wheat
- Cow's milk
- Eggs
- Fish
- Shellfish
- Tree nuts
- Soybeans

3.2 My Ontology Design

My first ontology design was laid out as can be seen in Figure X and was heavily inspired by [8]. It consisted of Dishes, Components, Ingredients and Allergens with an obejct property *"contains"* to link them in that order. A Dish contains Components, which contains Ingredients which contains Allergens. Then there are Customers, who *"canEat"* dishes - another obejct property. Calorie information was stored on an ingredient level, with a data property *"hasCalories"*. On top of this, a *"PreparationMethod"* class would be created to store whether a dish was suitable for halal or kosher diets based on how it was prepared. This could also be extended to encompass unhealthy cooking methods, such as deep frying.

After reading the paper that inspired that diagram [8], I realised how easy it was to create redundant data. For example, the Customer class is not required at all - we have all we need from the defined dish classes.

3.3 Evaluation Plan

To test and evaluate the ontology system, I plan to conduct user studies, where I will ask people that I know to carry out certain tasks using the system, such as adding or removing a dish from the system, and searching for dishes with certain parameters. These tasks will cover each aspect of the functional requirements of the system to completely evaluate how successfully the system meets the criteria. I will then ask a few short questions to cover the non-functional requirements and assess the success of the system at meeting those.

The user studies will be caried out in two rounds. The first pilot study will be carried out much earlier, so I can gauge what improvements need to be made. This is important as I will be able to identify whether I am on the right track with the project, and gain feedback on what does and does not work. Then, I will carry out a second user study after development on the system has finished to see how the system fits the requirements set out. Ideally, I would carry out a series of continuous feedback sessions - where users can repeatedly give feedback on the system for me to improve.

Each user study will be split into two smaller evaluations, one of the user interface for the customer, and one of the user interface for the restaurant owner. The participants were split in half and instructed to act the role of either a customer or restaurant owner. Each study will be tailored towards what those two roles should be able to achieve from their separate interfaces. For example, for the customer study, the participant will not be given any instruction on how to use the system, as real customers will not and should not need to require training to use the system. However, the owner of the restaurant will be told how the system works initially, and then not during the second phase of the final study. This is because in the real world, they will only recieve training initially, and should not need to relearn every time they want to use the system. A restaurant does not change it's menu that often, so they will not be using the system frequently.

It is important that the participants of the study are comfortable and give an honest review of the system. To do this, I aim to avoid putting them under any pressures. I will not place them under any time constraints to complete the tasks, and make sure that when I propose the study to them that the time estimation is accurate, while also a slight overestimate. It is also important that the participant does not feel any pressure to falsely support the system, when it is in fact failing. To do this, before the study I made it clear to the participants that they were not being judged based on whether they were able to complete the tasks, and that the results are purely reflective of how well the system is built and not their abilities. I also made sure they knew that I wanted to hear about problems as I want to improve the system where it is failing. Especially during the pilot I emphasized that there are improvements that need to be made and encouraged the participant to point out problems, as there was still plenty of time left to fix them.

I chose to have 3 participants per role for my study, as I found in the pilot that this number was the most appropriate. The big issues that I expected were brought up by all participants, and everything else was small or personal preferences that were low priority issues to fix.

3.3.1 Ethics

I will carry the studies out in an ethical way. All participants will be over 18 and able to fully consent to taking part. No participant will be coerced into completing the study, and they are free to withdraw at any point. Also no personal information will be stored about them, or any information that can be used to identify them. The only information that will be collected will be whether they were able to complete all the tasks, and the feedback given, both qualitative and quantitative.

On top of this, there will be no harm inflicted on the participants, and I will make it clear what I am using the information I collect is for.

3.3.2 Interpreting the Results

With the results of the user study, I adopted a mildly agile workflow to move forward. I listed all of the problems that were identified during the study, and then a list of solutions. I scored each solution from 1-5 with an urgnecy score - based on how important it was that this change was made - and a time score - how long I estimated it would take to implement the solution. I then made sure to tackle the high urgency issues first, especially those with a small time estimate. This way, I could make sure to get as many of the more impactful problems solved within the time I had left.

3.4 Design Choices

Initially, I started creating the user interface entirely in one class, with a method that would create and return a JFrame for each of the different pages. I would then change Frame by closing the old one and opening the new one. After making a few pages I quickly realised this was a bad idea. I ended up making 19 different pages, and this would create one very difficult to read and convoluted class. After doing some research [**?**], I found the better way to approach this was to make each page it's own class which extends JPanel, and then use a CardLayout to switch between each page. The CardLayout gives takes each page with a String that can be used to identify it (I made this a class variable of the page), and then you can show the panel you desire by using it's identifier. Despite the time it took to learn about, it made my code much more navigable and saved me time in the long run.

One design choice that was made after my pilot study was to change the way I showed how things in a list were selected. Previously, users had a list of everything

provided, and had to click on an element to select it. If a user wanted to select multiple elements in the list, they would have to hold the Ctrl button to do so. If Ctrl was not held, any previously selected elements would be unselected. This is similar to how file selection works in File Explorer, and the default selection model for a JList. However, as many of lists were long and had to be scrolled, it was not obvious that the previously selected elements were now unselected. My first thought was to solely change the selection model so that you only needed to click on each element to select it, and the only way to unselect an element was to select it again. However I ultimately decided on a different method, where there is one box to select elements from, and another box which contains all the selected elements. To unselect an element, click on it in the second list and it will automatically update. This makes it much more clear what has been selected as you can see them without seeing everything that has not been selected. It also supports the ability to select an element multiple times, which could be desirable in the case where a customer wants to add an ingredient twice to a dish.

One big consideration I had at the end was the purpose of the Component. With the particular menu I have chosen, components are not particularly necessary. A burger is a whole meal and it does not consist of definitive sections, just ingredients. This caused some confusion at what components to make during the user study when the user was given free reign to make a dish. Other menus lend themselves more to the idea of a Component however. If you have pie with mash potatoes and roasted vegetables - you have three very clear components to the final dish made up of many ingredients themselves. It lends itself to larger dishes that contain multiple parts, rather than menus where you order lots of small bits or somethiing that could be a Component itself. This is the reason I decided to keep Components in, as it gives the system flexibility to be used in different kinds of restaurants.

Chapter 4

Results

4.1 Deliverables

4.1.1 Ontology

The final ontology (stored in *'Menu.owl'*) contains the Honest Burgers menu. Each dish on the menu is a dish in the ontology. The dish contains all information regarding how it was prepared.

All the ingredients in a dish are also in the ontology. Each ingredient stores the number of calories that ingredient contains (on a class level), and the allergens it contains. There were times when I had to decide what level an ingredient was, for example the *'PestoIngredient'* which contains multiple ingredients in itself, but is listed as an ingredient in the dish. I thought it wasn't useful to list every single thing that makes up pesto, but just to make sure the allergens are stored.

What ingredients to put in a component is also a subjective task, and for this menu there is no intuitive set of components to make. I decided in this case to store the patty, burger bun, and fillings for each dish as the components.

The object properties in the ontology

The data properties in the ontology are *'hasCalories'*, which

4.1.2 Restaurant User Interface

The user interface (UI) for the restaurant owner opens onto the Main Menu Page (see Figure X). The user is presented with 4 options, to add, remove, search/edit and quit. The quit button will save the ontology and close the application.

The add section contains options to add either an allergen, ingredient, component or dish to the ontology. The add menus can be seen in the figures A, B, C and D. After attempting to add to the ontology, one of the message boxes in Figure A will be shown in response, depending on whether the attempted addition to the ontology is valid or not.

The remove section is very similar to the add section, except they show lists for a user to select an element to remove. Similarly, a message feedback box is shown to inform the user of the changes to the ontology.

search/edit

4.1.3 Customer User Interface

The user interface for the customer is almost a subset of the restaurant owners user interface. It opens onto another main menu page (see Fig X) where the user has 3 options. The user can search the menu, open the settings page or quit the application.

The settings page (see fig X) currently contains one option, to hide the calorific information. This option will disable the ability to filter by calories, and also hide how many calories are in a dish and an edited dish.

search

4.2 User Studies

4.2.1 Pilot User Study

I carried out the pilot user study with 6 participants, 3 who acted as a customer and 3 who acted as a restaurant owner.

The feedback given

4.2.2 Final User Study

Chapter 5

Evaluation

5.1 Results Summary

5.2 Did I Meet My Objectives?

5.3 Improvements

Despite this, there are many improvements that can still be made. First of all, the appearance of the program coulde definitely be improved. The visuals were not the focus of this project and I solely focused on the functionality, so there is great room for improvement. The visuals were mentioned in the final evaluation by one user of the RestaurantUI, who mentioned that it was difficult to differentiate between all the different menus without reading the title as they are all very similar. Making the visuals slightly different would help this, as long as the overall appearance remained consistent.

Another addition that could be made is the inclusion of a text search at the top of lists. The list of ingredients and components can grow very long, even with the ingredient type filter, and it can be easy to miss things - whereas a text search would make it easy to find specific items.

On top of this, a lot of the text handling could be improved. At the minute, no class can have a space in the name of it as the system can't handle it. Spaces are stored differently in the ontology file and not handled when reading from it. The solution I would implement now is the following: When recieving an input of text, remove any spaces and convert the text to pascal case before add/removing or querying the

ontology. Then, whenever the name of something is displayed: enter a space before every capital letter except the first letter. This would work because the standard for storing things in an ontology is in pascal case.

Another text handling issue is that you can't include the words Dish, Component, Ingredient etc. within the name of a class. This is because they are used as suffixes in the ontology, and all instances of the text is removed when reading names from the ontology. This would be a simple fix, by making sure only the final occurance of that text is removed from the name of a class.

5.4 Future Work

There are many possibilities when it comes to extending this project. The first is to add a transactional side to it, so that a restaurant owner would be able to add prices and the cost of adding ingredients. A customer would then be able to order dishes and pay for them, while the order would be send through to the kitchen. This extension could see the project become a viable option for restaurants to use in practice.

Another extension of this project would be to make it more customizable from the perspective of the restaurant owner in terms of diets. Diets are changing over time, and there are new dietary requirements that need to be catered for as time goes on. It would be useful if the owner of the restaurant could add more than just new allergens to the system. For example, Halal and Kosher are currently types of preparation method, but it would be useful if the owner could make more of these. One example would be deep fried foods, which are causing concern because of their high calorific content, high concentration of trans fats and the increase they cause in the risk of developing heart disease, diabetes and obesity[11]. Therefore the restaurant owner might think that it would be useful if customers were able to filter out dishes that have been deep fried at some point. Currently, they would have to contact a software developer who would have to hard code this change into the system. It would be useful if this was not the case.

Bibliography

[1] Wesley Burks, Ricki Helm, Steve Stanley, and Gary A Bannon. Food allergens. *Current opinion in allergy and clinical immunology*, 1(3):243–248, 2001.

[2] Halal Monitoring Committee. What is halal? the meaning of halal explained in reference to food, Dec 2018.

[3] OWL CS. Owl api, Feb 2023.

[4] Michael DeBellis. New protégé pizza tutorial, 2021.

[5] Stanford Center for Biomedical Informatics Research. A free, open-source ontology editor and framework for building intelligent systems, 2023.

[6] Conrad Hackett and Brian J Grim. *The Global Religious Landscape*. Pew Research Center, Dec 2012.

[7] Public Health. Guidance on foods for religious faiths, 2009.

[8] Pablo López-García, Eduardo Mena, and Jesus Bermudez. Some common pitfalls in the design of ontology-driven information systems., 01 2009.

[9] Nicolas Matenzoglu and Ignazio Palmisano. An introduction to the owl api - university of manchester, Apr 2016.

[10] Edouard Mathieu and Hannah Ritchie. What share of people say they are vegetarian, vegan, or flexitarian?, May 2022.

[11] Kayla McDonnell. Why are fried foods bad for you?, Feb 2023.

[12] Sade Meeks. What is kosher? definition, examples, diet, and more, Jan 2023.

[13] Boris Motik, Rob Shearer, Birte Glimm, Giorgos Stoilos, and Ian Horrocks. Hermit owl reasoner.

[14] The University of Manchester. Owl api documentation, 2023.

[15] Oracle. Package javax.swing, Jun 2020.

[16] Michael Roskams. *Religion, England and Wales: Census 2021*. Office for National Statistics, Nov 2022.

[17] Ben Shneiderman. The eight golden rules of interface design.

[18] Scott H. Sicherer and Hugh A. Sampson. Food allergy: A review and update on epidemiology, pathogenesis, diagnosis, prevention, and management. *Journal of Allergy and Clinical Immunology*, 141(1):41–58, Nov 2017.

[19] Helen West. The 8 most common food allergies, Apr 2023.

[20] Nils-Gerrit Wunsch. *How often do you dine out?* Statista, Jun 2022.

Appendix A

Example of operation

An appendix is just like any other chapter, except that it comes after the appendix command in the master file.