The Manchester Sushi Finder

A dissertation submitted to the University of Manchester for the degree of Master in Advanced Computer Science in the Faculty of Engineering and Physical Sciences

2014

Atheer Algherairy
School of Computer Science
# Table of Contents

List of Figures ........................................................................................................... 5  
List of Tables ............................................................................................................. 7  
Abstract .................................................................................................................... 8  
Declaration ............................................................................................................... 9  
Intellectual Property Statement ............................................................................. 10  
Acknowledgments .................................................................................................... 11  
Chapter 1 Introduction ............................................................................................. 12  
  1.1. Objectives ........................................................................................................ 12  
  1.2. Dissertation outline .......................................................................................... 13  
Chapter 2 Background Research ............................................................................ 14  
  2.1. Chapter Introduction ......................................................................................... 14  
  2.2. Introduction to Ontologies ............................................................................... 14  
  2.3. OWL Ontologies ............................................................................................... 14  
  2.4. Sushi Ontology ................................................................................................ 16  
  2.5. Previous Research on Ontology Querying ..................................................... 19  
    2.5.1. Ontology Visual Querying ......................................................................... 19  
    2.5.2. An Ontology Driven Interface (Sanctioning Approach) .............................. 21  
  2.6. Analysis of Existing Systems .......................................................................... 23  
    2.6.1. The Manchester Pizza Finder ..................................................................... 23  
  2.7. Tools for Querying OWL Ontologies ............................................................. 27  
    2.7.1. OWL API .................................................................................................. 27  
    2.7.2. SPARQL .................................................................................................... 28  
    2.7.3. Comparison between OWL API and SPARQL in Querying OWL Ontologies ...... 29
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.8. Chapter Summary</td>
<td>32</td>
</tr>
<tr>
<td>Chapter 3 System Design</td>
<td>33</td>
</tr>
<tr>
<td>3.1. Chapter Introduction</td>
<td>33</td>
</tr>
<tr>
<td>3.2. Requirements Gathering</td>
<td>33</td>
</tr>
<tr>
<td>3.3. Overall System Architecture</td>
<td>34</td>
</tr>
<tr>
<td>3.4. The Query Interface</td>
<td>35</td>
</tr>
<tr>
<td>3.5. The Configuration Tool</td>
<td>38</td>
</tr>
<tr>
<td>3.6. The Configuration File</td>
<td>39</td>
</tr>
<tr>
<td>3.7. Chapter Summary</td>
<td>42</td>
</tr>
<tr>
<td>Chapter 4 System Implementation</td>
<td>43</td>
</tr>
<tr>
<td>4.1. Chapter Introduction</td>
<td>43</td>
</tr>
<tr>
<td>4.2. Implementation Platforms and Programming Languages</td>
<td>43</td>
</tr>
<tr>
<td>4.3. Query Template Engine</td>
<td>45</td>
</tr>
<tr>
<td>4.3.1. The Mapping Algorithm</td>
<td>45</td>
</tr>
<tr>
<td>4.4. The Query Interface</td>
<td>49</td>
</tr>
<tr>
<td>4.4.1. Browsing the Ingredients</td>
<td>50</td>
</tr>
<tr>
<td>4.4.2. Including and Excluding Ingredients</td>
<td>54</td>
</tr>
<tr>
<td>4.4.3. Switching Between Base Things and Languages</td>
<td>61</td>
</tr>
<tr>
<td>4.5. The Configuration Tool</td>
<td>67</td>
</tr>
<tr>
<td>4.6. Issue Faced During Development</td>
<td>70</td>
</tr>
<tr>
<td>4.7. Chapter Summary</td>
<td>71</td>
</tr>
<tr>
<td>Chapter 5 Testing and Evaluation</td>
<td>72</td>
</tr>
<tr>
<td>5.1. Chapter Introduction</td>
<td>72</td>
</tr>
<tr>
<td>5.2. Testing</td>
<td>72</td>
</tr>
<tr>
<td>5.3. Evaluation</td>
<td>72</td>
</tr>
<tr>
<td>5.4. Comparison between the Pizza Finder and the Sushi Finder</td>
<td>76</td>
</tr>
</tbody>
</table>
5.5. Chapter Summary ................................................................................................................. 78

References .................................................................................................................................. 81

Appendix A: Configuration File ............................................................................................... 84

Appendix B: Questionnaires ...................................................................................................... 87

Final Word Count: 16,625
List of Figures

Figure 2-1: Illustration of classes, properties, and individuals .................................. 16
Figure 2-2: Classes and properties in the sushi ontology ............................................. 17
Figure 2-3: The relationships between base things and their ingredients ..................... 18
Figure 2-4: Query composition in SQoogle, part of the SEWASIE Project .................... 21
Figure 2-5: The Manchester Pizza Finder query interface ........................................... 24
Figure 2-6: The Manchester Pizza Finder results interface ......................................... 25
Figure 2-7: Ontology management in OWL API ......................................................... 27
Figure 2-8: Example 1 - DL Query ............................................................................... 29
Figure 2-9: Example 1 – SPARQL ............................................................................. 30
Figure 2-10: Example 2 - DL Query ........................................................................... 30
Figure 2-11: Example 2 - SPARQL .......................................................................... 31
Figure 3-1: The Manchester Sushi Finder Architecture ............................................. 35
Figure 3-2: The Query Interface ................................................................................. 36
Figure 3-3: The Configuration Tool ............................................................................ 38
Figure 3-4: Label Annotations for Chicken Class ...................................................... 39
Figure 4-1: The Query Interface ................................................................................ 50
Figure 4-2: Different View Tabs in the Query Interface ............................................. 51
Figure 4-3: Extensional Feedback Mechanism in the Query Interface ....................... 55
Figure 4-4: Implementation of Sanctioning-like Mechanism ..................................... 60
Figure 4-5: Included Expression in the Query Interface ............................................ 60
Figure 4-6: Selecting Base Thing (Sushi) .................................................................. 62
Figure 4-7: The Results (Sushi) ................................................................................ 63
Figure 4-8: Selecting Base Thing (Platter of Sushi) ................................................... 64
Figure 4-9: The Results (Platter of Sushi).................................................................................65
Figure 4-10: Switching to Arabic Language..............................................................................66
Figure 4-11: Switching to Spanish Language..............................................................................66
Figure 4-12: The Configuration Tool Options..............................................................................67
Figure 4-13: Select Configuration File (A).................................................................................68
Figure 4-14: Select Configuration File (B).................................................................................68
Figure 4-15: New Configuration File – Different Views Tab.........................................................69
Figure 4-16: New Configuration File – Results Tab....................................................................69
List of Tables

Table 1: A Summary of Not Supported Features in the Pizza Finder..............................................26
Table 2: Examples of the Short forms of Classes’ IRIs .................................................................40
Table 3: A Summary of the Application’s Functions .................................................................71
Table 4: Times Spent in Answering Question One.................................................................74
Table 5: Times Spent in Answering Question Two .................................................................74
Table 6: Times Spent in Answering Question Three ...............................................................75
Table 7: Times Spent in Answering Question Four .................................................................75
Abstract

The Manchester Pizza Finder is an existing Java application that demonstrates the benefits of using the Web Ontology Language (OWL) ontology within an application. The finder application gives a user the ability to find the pizza that meets his/her preferences of what toppings are included and what toppings are excluded. The Manchester Pizza Finder has been developed by Matthew Horridge at the University Of Manchester, and it uses Pizza Ontology and description logic (DL) reasoner to generate the available pizza toppings and query the ontology in order to retrieve a list of pizzas based on user’s choices of included and excluded toppings.

The current version of the pizza finder application works with ontologies of simple structures; that represent only one base thing (e.g., pizza) that is directly related to its components (e.g., ingredients, toppings, etc.) through one object property.

The aim of the project is to develop an intelligent finder tool, named The Manchester Sushi Finder. It could be considered as an extending to The Manchester Pizza Finder with new enhancements in order to make it more flexible to work with ontologies that are developed based on different restaurants’ menus and configurable in many areas such as: alternative languages and labels appear in the interface. These ontologies, consumed by the application, can represent a number of base things that are related to their components directly through one object property or indirectly through a number of object properties.

The main focus of this project is to develop a Java application that not only allows a user to select a dish based on various criteria taken from OWL ontology but also tries to investigate the ability of consuming ontologies from different places/users, browse them, allow end users to easily construct proper queries against them and obtain reasonable results without having to deal with the underlying representations or the concrete syntax of the ontology.
Declaration

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

i. The author of this dissertation (including any appendices and/or schedules to this dissertation) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this dissertation, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has entered into. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the dissertation, for example graphs and tables ("Reproductions"), which may be described in this dissertation, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this dissertation, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://documents.manchester.ac.uk/display.aspx?DocID=487), in any relevant Dissertation restriction declarations deposited in the University Library, The University Library's regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University’s Guidance for the Presentation of Dissertations.
Acknowledgments

I would like to express sincere gratitude to my supervisor, Dr. Sean Bechhofer, for his valuable guidance and advice, which has helped me in this project.

I would also like to extend my warmest thanks to my dear husband, Sulaiman, for his unconditional moral support, encouragement and patience during this period.

I also thank my lovely son Mohammad for his patience during the busy times I have encountered throughout the duration of my studies.

Finally, I wish to express my love and gratitude to my beloved family—my mother, father, brother and sisters—for their understanding, encouragement and endless love throughout the duration of my studies.
Chapter 1  Introduction

1.1. Objectives

In recent years, there has been a growing interest in developing and using domain-specific ontologies which represent the knowledge about things and relations between things in a domain of interest by providing formal vocabularies and their intended meanings [1]. A Sushi Ontology that describes sushi, platters of sushi, and their ingredients - based on YO! Sushi¹ menu - has been developed by students who took Ontology Engineering for Semantic Web course², therefore the idea of The Manchester Sushi Finder came across.

The objectives of this project are as follows:

1- To develop an intelligent finder tool that uses an ontology describing dishes in a restaurant menu.

2- To give customers the ability to find the dish that meets their preferences. This is done by allowing them to construct proper queries against the ontology and obtain reasonable results without having to deal with the underlying representations or the concrete syntax of the ontology.

3- To provide the students of the Ontology course with an experimental platform from which they could browse and query their ontologies, and see how their models could be consumed within an application.

In order to achieve these objectives, the following tasks need to be accomplished:

1- Examine a similar system and research ontology querying.

2- Gather the system requirements for the proposed application.

3- Design the system architecture.

4- Implement the system.

5- Evaluate the developed system.

¹ YO! Sushi is a Japanese restaurant that offers various sushi dishes and other Japanese food.
² http://www.cs.manchester.ac.uk/study/postgraduate-taught/courses/courseunitsyllabus/?courseunitcode=COMP60421
1.2. Dissertation outline

The dissertation is structured as follows: Chapter Two: includes a literature review for the relevant aspects to this project along with an analysis of a similar existing system. Chapter Three: discusses the requirements and design of the system. Chapter Four: presents the implementation of the system. Chapter Five: describes the evaluation of the system along with a comparison between the existing and new systems. Finally, the dissertation is concluded by providing a summary of the project and a suggestion for future work.
Chapter 2  Background Research

2.1. Chapter Introduction

This chapter provides the reader with a general literature review for the relevant aspects to this project. It introduces OWL Ontologies and its elements with particular reference to the Sushi Ontology. This is followed by a specific background research on ontology visual querying and ontology driven interface. Then, an analysis of a similar existing system is presented. At the end of this chapter, a discussion of some available tools to query OWL Ontologies is provided to evaluate them to show the most suitable tool for this project.

2.2. Introduction to Ontologies

Numerous data models have been defined to represent information in computer systems. One popular model is the Database model, which has increasingly been used in different applications. However, Ontologies have been introduced to this area, along with a growing interest in using them rather than databases in applications that need a richer meaning [2].

Ontology is commonly defined as “a formal, explicit specification of a shared conceptualization” [3]. It is used to represent knowledge about things and relationships between things in a domain of interest by providing formal vocabularies and their intended meanings [1]. Ontology consists of classes, instances of classes and properties between these instances. Moreover, it can be seen as a set of axioms or constrains. These axioms allow us to state additional information about classes, properties, and instances [4]. Ontologies use logic languages such as: Description Logic (DL) to represent these axioms and express more semantics. Using these semantics, reasoning can be performed and inferences can be drawn about ontologies [2].

2.3. OWL Ontologies

A recent ontology language is the Web Ontology Language (OWL) from the World Wide Web Consortium (W3C). The ontology and its elements are identified using Internationalized Resource Identifiers (IRIs). The main components of OWL ontology are classes, individuals, and properties. A brief description of each is given below [1].
1- Classes: Classes can be seen as “sets that contain individuals” [1]. A class may have a number of subclasses. Therefore, classes in the ontology could be organised into a hierarchy, called a taxonomy, in which subclasses are subsumed by their super classes. For example, if we have class A, which is subclass of class B, then all members of class A are members of class B. Moreover, ontologies have a general class called Thing, which contains all individuals in the ontology. Therefore, all classes are subclasses of Thing. In OWL ontologies, formal descriptions are used to describe classes and determine the conditions that need to be satisfied by an individual in order to be considered a member of the class. Furthermore, an individual can be a member of more than one class if it satisfies each class’ conditions.

2. Individuals: Individuals are instances of a class, and represent objects in the domain of interest.

3. Properties: Properties in ontologies represent relationships. In OWL, we have three types of properties:

   - **Object properties**, which link two individuals. The domain and range of an object property can be specified. This means that the property links individuals from the specified domain to individuals from the specified range.

   - **Data type properties**, which link individuals to a data value.

   - **Annotation properties**, which are used to annotate (add description data to) other components in the ontology. This type of property could be used to add comments, labels with alternative languages, and authorship information. Moreover, annotation properties do not have formal semantic or logical meanings [5].
Figure 2-1 illustrates an example of the main components mentioned above. There are three classes: **Person**, **Pet**, and **Country**. Each class contains a number of individuals, represented as small diamonds. Moreover, there are three object properties: **livesInCountry**, **hasPet**, and **hasSibling**. These properties link two individuals together; for example, the **livesInCountry** property links together **Matthew** and **England**.

### 2.4. Sushi Ontology

As mentioned earlier, the purpose of the project is to use an ontology that describes dishes in a restaurant menu and develop an intelligent finder tool that gives a customer the ability to find a dish that meets his/her preferences. My goal is to make the tool work with ontologies that are developed based on different restaurants’ menus and they do not need to be about sushi. These ontologies should have a particular structure. In other words, the ontology should represent base things and components (e.g., ingredients, toppings, etc.). These base things are related directly or indirectly to their components.

An example of the required structure can be illustrated using a sushi ontology that has been developed by a student who took the Ontology Engineering for Semantic Web course. This sushi ontology describes sushi, platters of sushi, and their ingredients based on the **YO! Sushi** menu. They are modelled as classes in the ontology, as they are considered to be descriptions of kinds of things. The ontology has been developed using Protégé; Figure 2-2 shows the classes and properties in the ontology.
Different areas have been modelled in the ontology. However, I will focus only on the following classes and properties:

- The `NamedSushi` class is used to represent different kinds of sushi pieces.
- The `SushiPlatter` class is used to represent different kinds of sushi platters.
- The `SushiIngredient` class is used to represent different kinds of ingredients from which a piece of sushi is formed.
- The `hasIngredient` object property links a piece of sushi with its ingredients. It has the domain `NamedSushi` and the range `SushiIngredient` (see Figure 2-3).
- The `hasContent` object property links a platter of sushi with its contents that are pieces of sushi. It has the domain `SushiPlatter` and the range `NamedSushi` (see Figure 2-3).

To clarify the required structure, consider two pieces of sushi that are represented in the ontology: **Prawn Nigiri** and **Avocado Maki**. They are subclasses of the `NamedSushi` class. Prawn Nigiri has the following ingredients: rice, prawn, and wasabi, and is thus directly related to its ingredients, using the `hasIngredient` object property. Similarly, Avocado Maki has the following ingredients: rice, nori, avocado, and Japanese mayonnaise, and is thus directly related to its ingredients, using the `hasIngredient` object property.

Now, assume that the ontology represents a platter of sushi, called **Assorted Nigiri and Maki Platter**. It is subclass of the `SushiPlatter` class, and its contents are two pieces of Prawn Nigiri and three pieces of Avocado Maki. The `hasContent` object property relates
Assorted Nigiri and Maki Platter to its contents, Prawn Nigiri and Avocado Maki, and the hasIngredient object property relates each piece of sushi to its ingredients.

Therefore, this platter has pieces of sushi and these pieces of sushi have ingredients. Consequently, we can say that this platter is indirectly related to its ingredients, which are the ingredients of its contents.

As a result, this sushi ontology has two base things: NamedSushi and SushiPlatter. NamedSushi is directly linked to its ingredients while SushiPlatter is indirectly linked to its ingredients.

The aim of the project is to develop a tool that can work with ontologies of this targeted structure with any depth. This means that we may have a base thing that is indirectly related to its ingredients through any number of object properties.

Suppose we have a Sushi Tray that has a number of platters, and that each platter has a number of pieces of sushi, and that each piece of sushi has a number of ingredients. The Sushi Tray is indirectly related to its ingredients within a deeper nested structure than that of the Sushi Platter. Figure 2-3 shows an abstract overview of the three base things described here and how they are related to their ingredients.

![Diagram showing the relationships between base things and their ingredients]

**Figure 2-3**: The relationships between base things and their ingredients

The aim of The Manchester Sushi Finder is to develop a query interface that works with ontologies of this particular shape and allows end users to construct and execute queries. Those queries will retrieve base things that satisfy their selections of what
ingredients are included and what ingredients are excluded. Students who are taking the *Ontology Engineering for Semantic Web* course will build ontologies of that shape. Therefore, one of the motivations for developing such a query interface is to provide them with an experimental platform from which they can browse and query their ontologies, and to see how their models could be consumed within an application.

2.5. Previous Research on Ontology Querying

2.5.1. Ontology Visual Querying

Ontology visual querying is “the use of an ontology to direct interactive query construction” [6]. Questions that can be asked over an ontology are restricted by the classes and properties defined in the ontology. Therefore, it is useful to have an interface that helps the user to construct queries that correspond to meaningful questions. Those questions can be asked over ontology, and they are expected to return answers.

Ontology visual querying provides a visual representation of the model and allows users to navigate and recognize the domain of interest over which queries will be constructed. Moreover, users are not required to remember the ontology and its concepts in order to construct queries over it [6].

A number of ontology visual query systems has been deployed such as SEWASIE³. According to Bechhofer and Paton (2009) [6], such systems can have a number of characteristics, some of which are:

- *Identification of starting points*: A place in the ontology from which the query construction can be started.

- *Query language used in the system*: Query languages differ in their expressivity and thus differ in the precision of the questions that can be asked.

- *Query modification operations*: Query expressions can be modified during query construction, such as by adding/removing relationships in the expression or replacing a named concept with a more specialised one.

- *The ontology definition language*: Ontology languages differ in both their

³ [http://www.dbgroup.unimo.it/Sewasie/](http://www.dbgroup.unimo.it/Sewasie/)
expressivity and the level of complexity in displaying and exploring them.

- **Query presentation:** There are different ways of presenting queries; they can be presented visually or in a natural language.

- **Feedback mechanisms:** Feedback may be **intentional** or **extensional**. The intentional mechanism depends on the knowledge of the definitions in the ontology, while the extensional mechanism is about the returned results.

SEWASIE was a three-year research project started in 2002 [7]. The project aimed to develop an advanced search engine by providing access to heterogeneous data on the Internet through an integrated ontology. SEWASIE consists of different systems. In particular, I will consider the query tool system, which has a user-friendly interface and allows the user to construct and execute queries. Figure 2-4 shows this interface. It contains five tabs to carry out the required processes. The user starts with the first tab, **Information Domain**, to select the domain of interest from the domains available.

Then, in the **Query Start** tab, the user selects a concept from the domain of interest to be the **starting point** of the query construction. After this, the query modification operations will be performed in the **Compose** tab, where the user will see the starting point chosen from the previous tab. From this starting point, the user can build the query, which will be represented as a tree. The user can perform a number of query modification operations, such as adding or removing properties and specialising or generalising concepts. At the end of the query construction, the user clicks the **Done** button and moves to the **Results** tab. The results are produced in a table that contains the query expression and the URI of the data sources [8].
2.5.2. An Ontology Driven Interface (Sanctioning Approach)

Description Logics (DLs) are knowledge representation languages that can be used to construct and represent conceptual models. A DL model consists of three things: concepts, roles and individuals. It is based upon a group of primitive concept definitions and assertions about the subsumption relations between these concepts, in addition to concept-forming operators, which can be used to construct the descriptions of composite concepts [9].

A DL model of a domain can be used to drive an interface which allows a user to construct or manipulate expressions. However, constructing or manipulating DL expressions or queries against a DL model is not always easy, especially for non-experienced users who are new to the knowledge representation field [9].

In the past, textual interfaces have been used in systems for interacting with DL models. Such interfaces require users to understand and deal with the underlying representations and the concrete syntax of the model [9].

For the purposes of facilitating interactions with DL expressions or queries and isolating users from the underlying representations, Bechhofer et al. (1999) in [9] develop a user tool that is based on a mechanism named Sanctioning. This mechanism constrains the construct of composite concepts, and ensures that only meaningful compositions can be built. These constraints (or Sanctions) are added to the DL model [9].

Figure 2-4: Query composition in SQoogle, part of the SEWASIE Project [8]
A sanction can be seen as a “constraint that says that a concept can be combined with another in the context of a particular role to form a new description. Without a sanction, the composition cannot be formed.” [9]

Sanctioning mechanisms have been used in TAMBIS and STRACH projects [9]. Both projects used GRAIL [10] description logic which is a knowledge representation language for medical terminology that was developed by the Medical Informatics group at the University of Manchester. GRAIL has a set of concept forming operators, in addition to a two-level sanctioning mechanism: so-called grammatical sanctions and sensible sanctions. The former represents the high-level, or general, relationships between concepts, while the latter represents the relationships that can be really formed.

Sanctioning mechanisms can be used to control the construction of compositions and expressions. Moreover, the use of sanctioning within an ontology plays an important role in the process of developing query interfaces that guide the user in constructing and manipulating appropriate query expressions while disallowing them from constructing queries that will never have answers [9].

Bechhofer and Horrocks (2000) in [11], describe one sanctioning-like mechanism: so-called reasonableness. This mechanism can be used with more expressive logic compared to that of GRAIL. A main difference is that the sanctions are not added to the DL model, but are instead added in a simple layer placed on top of the DL, using the reasoning services from the underlying DL [11].

Reasonableness controls the interface behaviour to ensure that only reasonable concepts can be constructed and only a reasonable number of specialisation options are presented to the user at any point. This mechanism allows us to answer the question, “What can I say about this concept”? This can be done by finding, for each concept $C$ in the Knowledge Base (KB), a list of concepts that can be reasonably conjoined with $C$ [11].

Assuming that $C$ and $D$ are names of concepts in the KB, and $R$ is a name of a role in the KB. The reasonableness information is a set ($R$) of assertions of the form reasonable ($C$, $D$) or reasonable ($C$, $\exists R.D$).

The reasonableness mechanism restricts the specialisation options for a concept $C$ by calculating the minimal non-redundant set of concepts ($R_C$) that can be reasonably conjoined with $C$, such that [11]:

22
In The Manchester Sushi Finder, when a restaurant customer uses the finder application to ask for sushi that contains specific ingredients, there are some further restrictions that the customer may want to specify, such as the cooking style or cutting style of the selected ingredients. However, these restrictions are applicable to some ingredients but not all of them; for example, the user can ask for a seared beef or fried chicken but searing and frying are not applicable to other ingredients such as mayonnaise. This brought up the need for the notion of Sanctions described above.

### 2.6. Analysis of Existing Systems

#### 2.6.1. The Manchester Pizza Finder

As mentioned previously, the idea of the project is inspired by the Manchester Pizza Finder. A Java application developed by Matthew Horridge at the University of Manchester, it uses Pizza Ontology OWL API⁴ and Description Logic (DL) reasoner to generate the available pizza toppings and query the ontology in order to retrieve a list of pizzas based on the user’s choices of what toppings should be included and what toppings should be excluded. Figure 2-5 shows the query interface, which contains the available toppings in the ontology represented in a tree, and allows the user to select and add toppings to the included or excluded lists. The user then clicks the Get Pizzas button to execute the query and get its results [12].

---

The pizza ontology used in this Pizza Finder has a NamedPizza class, a hasTopping property, and a PizzaTopping class. The hasTopping object property relates a NamedPizza directly to its toppings. In this application, the query is performed by creating a class expression that is an intersection over:

- Base class: NamedPizza;
- Expressions of the form hasTopping some X for each included topping X; and
- Expressions of the form not hasTopping some X for each excluded topping X.

Consequently, the resulting class expression can be used to obtain pizzas with the desired toppings.

The Pizza Finder has a graphical interface that is simple and easy to use. In addition to an XML file to store the ontology location and the names of classes and properties used by the application, if a user wants to load his own ontology, he can modify this file. The file looks like the one given below [13]:

![The Manchester Pizza Finder query interface](image)
Moreover, the Pizza Finder displays the results along with small icons representing vegetarian and spicy pizzas. Figure 2-6 shows how the results are displayed.

![Pizza Finder](image)

**Figure 2-6:** The Manchester Pizza Finder results interface.

However, the Pizza Finder works with ontologies of a simple structure; it uses only one base thing that is directly related to its components (e.g., ingredients, toppings, etc.) through one object property. Therefore, loading the sushi ontology into this application will only allow us to query about *NamedSushi*. Because it is the only base thing in the
sushi ontology that is related directly to its components, we cannot query about the other base things that are related to their components through a number of object properties, such as platters of sushi (see Figure 2-3).

Additionally, if we have more than one base thing modelled in the ontology and all of them are directly related to their components, the Pizza Finder will not allow us to have all these base things as options in the query interface, so that the end user can choose only one of them. Moreover, it does not support switching between alternative languages available in the ontology, nor does it support changing the appearance of the query interface, such as logos and labels. This could make the tool more flexible to suit the contents of different ontologies loaded to it.

Furthermore, there is no configuration interface that allows the user to configure the tool. If users want to load their own ontologies, they have to modify the XML file. Additionally, the Pizza Finder only shows icons for vegetarian and spicy results. However, the results may have other characteristics, such as Halal, nut-free dietary requirements, or low-calorie options. The following table summarizes the features that are not supported in the Pizza Finder.

<table>
<thead>
<tr>
<th>A summary of the features that are not supported in the Pizza Finder:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1- It does not support querying about more than one base thing.</td>
</tr>
<tr>
<td>2- It does not support querying about a base thing that is related to its components through a number of object properties.</td>
</tr>
<tr>
<td>3- It does not support switching between alternative languages available in the ontology.</td>
</tr>
<tr>
<td>4- It does not support changing the appearance of the query user interface (e.g. logo, labels).</td>
</tr>
<tr>
<td>5- There is no configuration user interface.</td>
</tr>
<tr>
<td>6- It only shows icons for spicy and vegetarian results. It does not support other characteristics of the results such as: nut-free or low-calorie options.</td>
</tr>
</tbody>
</table>

Table 1: A Summary of Not Supported Features in the Pizza Finder

26
2.7. Tools for Querying OWL Ontologies

There are many query languages that can be used to query OWL Ontologies such as: the RDF-based query language named SPARQL [14]. However, OWL API is used in this project. It is an application programming interface (API) for working with OWL ontologies in a Java application [15]. It allows creation, manipulation and reasoning over OWL ontologies. In OWL API, there is no specific query answering component to work with ontology query languages such as: SPARQL [15]. However, the reasoner interface in OWL API can provide fundamental query support that fits the needs of The Manchester Sushi Finder application. In this section, a brief discussion about OWL API and SPARQL is provided. In addition, the reason behind the choice of using a reasoner interface in OWL API, rather than using the SPARQL query language, is provided.

2.7.1. OWL API

OWL API is implemented in Java, and it is available as open source. Its initial development occurred in 2003. OWL API allows developers to use ontologies within applications at a suitable level of abstraction. The ontology can be seen as a group of axioms and annotations. Figure 2-7 shows this view. The key principle here is that OWL API isolates developers from other issues regarding how data structures are serialised or parsed. In particular, it provides “an axiom-centric abstraction” of the ontology that is independent from the underlying RDF or triple based representation [15].

Figure 2-7: Ontology management in OWL API [15]
As illustrated in Figure 2-7, the OWLOntology interface provides access to the axioms and annotations contained in an ontology. Each ontology instance is unique to a manager. Creating, loading, changing and saving an ontology are performed using its OWLOntologyManager. The ontology IRI is used to identify the ontology and the version IRI is used to identify the version of an ontology [15].

Moreover, the OWLReasoner interface is used to perform reasoning over an ontology. It supports a number of inference services such as: class satisfiability checking, ontology consistency checking, hierarchy information and reasoning about individuals [15]. Furthermore, there are number of implementations of the OWLReasoner interface for existing reasoners such as: HermiT [16], FaCT++ [17] and Pellet [18] [15].

In The Manchester Sushi Finder, the implementation of the HermiT reasoner is used to query the computed class hierarchy. This is done by retrieving the sub-classes of a complex class expression.

2.7.2. SPARQL

SPARQL is a query language that was standardized by the World Wide Web Consortium (W3C) in 2008. It is mainly designed to query RDF triples. The resource description framework (RDF) is a directed graph used to represent data in the web. Moreover, SPARQL can be used to query the underlying RDF serialisations of OWL ontologies [19]. The following is a SPARQL query example that returns subclasses of the class SushiPlatter:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX test: <http://www.ontologies.org/Sushi.owl#>

SELECT ?subject
WHERE { ?subject rdfs:subClassOf test:SushiPlatter}
```

The first lines in a SPARQL query define the namespaces used in the query. The WHERE clause contains the triple pattern. The query is performed by matching this triple pattern against the triples in the RDF graph. A SELECT clause is used to express
the solution by listing the variables to be returned, and those variables are bound in the query pattern [20].

2.7.3. Comparison between OWL API and SPARQL in Querying OWL Ontologies

This section clarifies the reason for the choice to use OWL API to construct class expressions and then obtain their subclasses, rather than using the SPARQL query language. Two examples of queries over the same Sushi Ontology are illustrated here. These are a simple version of queries that may be needed in The Manchester Sushi Finder.

Example 2:

A query asks for something that has avocado as an ingredient. The same query is written in two ways: the first one is a description logic (DL) query written in Manchester Syntax, and the other is a SPARQL query. Both queries are executed directly through the SPARQL query panel and DL query panel in Protégé. Figure 2-8 and Figure 2-9 show that they return the same results.

![Figure 2-8: Example 1 - DL Query](image-url)

Figure 2-8: Example 1 - DL Query
Example 2:

A query asks for something that has beef as an ingredient. Similarly, the same query is written in two ways. However, in this example, they do not return the same results; the SPARQL query returns no results. See Figure 2-10 and Figure 2-11.
Looking at the two resulted classes from the DL query, and how they are modelled in the ontology; the beef ingredient in both classes has a cooking style property indicating that the beef is seared in this two sushis:

Beef and hasCookingStyle some Seared

On the other hand, the underlying RDF/XML representations of these resulted classes have some triples that are semantic in nature, while others are syntactic. The assertion about the cooking style of the beef is combined with the language syntax (i.e. intersection). Therefore, in order to construct a proper RDF query and obtain the correct results, we may need to know how the ontology is serialized into RDF.

Moreover, according to Sirin and Parsia (2007), it is not easy to obtain semantics for such RDF-based query languages under OWL-DL semantics, which is due to the fact that the RDF representation combines the language syntax with its assertions [14].
However, The Manchester Sushi Finder is intended to deal with a number of ontologies. Therefore, looking at their underlying RDF triples is not the best choice. Alternatively, the use of OWL API gives the following advantages:

1- The level of abstraction provided by OWL API.
2- The ability to use the reasoner to obtain the inferences.

2.8. Chapter Summary

In this chapter a literature review about OWL Ontologies and the characteristics of ontology visual querying systems were discussed in details. This is followed by an analysis of an existing system and an overview about the tools that can be used in querying ontologies in The Manchester Sushi Finder.
Chapter 3 System Design

3.1. Chapter Introduction

This chapter provides the reader with an overview of the way in which The Manchester Sushi Finder works. First, the targeted users and their requirements are identified. Then, the system architecture and its components are explained in details.

3.2. Requirements Gathering

The first phase in developing The Manchester Sushi Finder was identifying the functional requirements of the application. This phase was conducted through discussions with the supervisor in order to specify the targeted users. In addition to analysis of the existing system (i.e., the Pizza Finder), the potential features the new system were investigated. Moreover, the previous researches on querying ontologies in terms of what the characteristics of a visual ontology query system are and how the sanctioning-like mechanism can be used to guide the construction of reasonable queries have been taken into consideration while developing the sushi finder. As a result of this phase, a list of user scenarios has been identified.

For The Manchester Sushi Finder, there are two targeted users: a tool provider and an end user. The tool provider knows the underlying representations of the ontology and will be able to configure the tool to work with the ontology. The tool provider could be a student of the Ontology Engineering for Semantic Web course. On the other hand, the end user is a restaurant customer who used the tool to find dishes that meet his/her preferences and he/she has no experience in the knowledge representation field.

Consequently, the user scenarios are defined as follows:

For the tool provider:

1- As a tool provider, I want to upload the ontology to be consumed by the application.
2- As a tool provider, I want to define query templates for the ontology.
3- As a tool provider, I want to determine the alternative languages that are available in the ontology so that the end user can choose one of them.
4- As a tool provider, I want to determine the components class (ingredients, toppings, etc.), facets properties and classifications so that the user can have different views of the components class.

5- As a tool provider, I want to determine some sanctions’ assertions so that the end user can make further restrictions on the selected component and still have reasonable query expressions.

6- As a tool provider, I want to determine a number of characteristics that the results might have (e.g. spicy) so that these characteristics appear to the end user.

7- As a tool provider, I want to write my own labels, which are to appear on the query interface.

8- As a tool provider, I want to upload the image of the restaurant’s logo in order for it to appear on the query interface.

9- As a tool provider, I want to store all of my configuration information (listed above from 1-8) in a file so that I can use it later.

10- As a tool provider, I want to be able to select a configuration file to be used by the application.

11- As a tool provider, I want to be able to modify my configuration file if needed.

For the end user:

12- As an end user, I want to select one of the available languages so I can view the components (ingredients, toppings, etc.) and results in that selected language.

13- As an end user, I want to easily find and select the component to be included or excluded.

14- As an end user, I want to select one of the available base things (such as: sushi, platter, etc).

15- As an end user, I want to obtain the correct results that satisfy my previous selections of base thing and included and excluded components.

3.3. Overall System Architecture

Based on the project objectives and the gathered requirements, an overall application design was needed to envisage the complete extent of the system and the effort needed to develop it. This has helped in dividing it into smaller components and prioritising
their implementation. Figure 3-1 shows a general overview of the system architecture to clarify the main components in the system and how do they communicate.

![Diagram](image)

**Figure 3-1: The Manchester Sushi Finder Architecture**

The system is divided into two main components: the query interface for the end user and the configuration tool for the tool provider. The query interface uses the information in the configuration file while the configuration tool generates this configuration file. A detailed description of the query interface, configuration tool and configuration file are provided in the following sections.

### 3.4. The Query Interface

The major component in the system is the query interface. This interface is for the end user, who is a restaurant customer. The interface is created based on information contained in the configuration file (i.e., config.xml). Figure 3-2 illustrates the architecture of this component.
Figure 3-2: The Query Interface

As seen in Figure 3-2, the query interface communicates with other components in the system to perform a number of functions. These functions are as follows:
1- It allows the user to browse the ingredient class using the main view: the tree view. It also provides three more views - as tabs- so the end user can choose one of them:

- **Search view** which is produced by the *Search Component*. The query interface passes the ontology object along with the ingredient class. The component then communicates with Hermit reasoner to generate a list of all ingredients. This view contains a search box that performs an incremental search on the ingredients list.

- **Classes Filtering view** which is produced by the *Classes Filtering Component*. The query interface passes the ontology object along with the ingredient class and a list of classes specified in the configuration file and then the component generates a list of checkboxes correspond to these classes and a list of all ingredients. When the user selects a checkbox, the component communicates with Hermit reasoner to do filtering on the list of ingredients based on what the selected classes are.

- **Facets view** which is produced by the *Facets Component*. The query interface passes the ontology object along with the ingredient class and a list of facets properties specified in the configuration file and then the component generates the fillers options to each property as checkboxes and a list of all ingredients. When the user selects a checkbox, the component communicates with Hermit reasoner to do filtering on the list of ingredients based on the selected property and filler.

2- It allows end users to select one of the available bases (such as sushi, platter, etc.) and to select included and excluded ingredients. When the user selects an ingredient, the sanctioning-like mechanism is performed using the *Sanctions Component* to give the user the ability to say right things about the selected ingredient which are further characteristics the ingredient might have (such as: fried, raw, sliced, etc.).

3- It passes the selected information (from step 2) to the *Query Template Engine*. This engine is responsible for constructing a proper class expression for the query. In order to perform this task, the engine applies the *Mapping Algorithm* to
the templates stored in the configuration file. The templates and the algorithm are explained later in 4.3. in Chapter Four. Thereafter, the engine communicates with Hermit reasoner to obtain the subclasses of the constructed class expression and passes these subclasses back to the query interface. These subclasses are the results that match the end user’s choices.

4- It allows the user to switch between alternative languages available and all the classes, annotations and object properties are rendered in the selected language.

### 3.5. The Configuration Tool

The other component in the system is the configuration interface, illustrated in Figure 3-3. This interface is for the tool provider, who can configure the tool to work with a specific ontology. Using this tool, the tool provider can: create a new configuration file, modify an existing configuration file and select a configuration file to be used by the application.

The main purpose of this interface is to allow the tool provider to configure the tool via a graphical interface, rather than modifying the configuration file (i.e., config.xml) manually. Moreover, the tool uses an XML Schema (i.e., XSD file) to ensure that the created, modified or selected configuration file is valid with respect to this schema.

This is done by allowing the tool provider to perform various tasks via this interface, such as uploading an ontology, constructing query templates, determining the alternative languages available in the ontology, and specifying other configuration information that guide the use of the ontology. In fact, the output of this interface is the configuration file, and this file is then used by the query interface.

![Figure 3-3: The Configuration Tool](image-url)

38
3.6. The Configuration File

The configuration file is an XML file that conforms to a specific XML Schema and contains all the information needed by the application to work with an ontology. When the application runs, it starts with a default configuration file. This file can be changed. The default configuration file is in Appendix A.

Each configuration file must have some necessary information to perform the basic functions of the query interface: browse the ingredient class and query the ontology. This information is as follows:

- Ontology Uniform Resource Location (URL).
- The IRI of the ingredient class in the ontology.
- Query Templates. (Described in 4.3. in Chapter Four)

There is other optional information that ensures a better use of the ontology:

- **Available languages in the ontology**: These languages appear to the end user as choices in the query interface. The ingredient class in all the different view tabs and the results classes will be rendered using their label annotations that are tagged with a language equals to the one that the end user chooses. Figure 3-4 is a protégé screenshot shows all the label annotations that (Chicken) class has.

![Figure 3-4: Label Annotations for Chicken Class](image)

- **The default class rendering**: This determines how the class should be rendered if it doesn’t have a label annotation with a language tag equals to the one chosen
by the end user. There are two options: either using the short form of the class IRI or a label annotation without a language tag.

A short form of an IRI is the string after the (#) symbol in the IRI. It is usually used for display purpose. The following table shows examples of the short forms of a number of classes’ IRIs.

<table>
<thead>
<tr>
<th>Class IRI</th>
<th>Short Form</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.sushi.com/ontologies/sushi.owl#Rice">http://www.sushi.com/ontologies/sushi.owl#Rice</a></td>
<td>Rice</td>
</tr>
<tr>
<td><a href="http://www.sushi.com/ontologies/sushi.owl#Beef">http://www.sushi.com/ontologies/sushi.owl#Beef</a></td>
<td>Beef</td>
</tr>
<tr>
<td><a href="http://www.sushi.com/ontologies/sushi.owl#111">http://www.sushi.com/ontologies/sushi.owl#111</a></td>
<td>111</td>
</tr>
</tbody>
</table>

**Table 2: Examples of the Short forms of Classes’ IRIs**

As seen in the last example in the previous table, the short form is opaque. Therefore, we cannot assume that the short form of an IRI is always meaningful.

In Figure 3-4, the Chicken class has a label annotation with no language tag: Amount of Chicken

And the Chicken class IRI is:

http://www.ontologies.org/Sushi.owl#Chicken

The short form of this IRI is: Chicken.

The choice between using the short form of a class IRI and an untagged label annotation is up to the tool provider who knows the ontology.

- **Ingredient Classifications:** A list of classes IRIs that are used by the Classes Filtering Component to generate the (Classes) view tab in the query interface.

- **Ingredient Facets Properties:** A list of object properties IRIs that are used by the Facets Component to generate the (Facets) view tab in the query interface.
- **Result Annotation Property:** An annotation property IRI that the tool provider uses when he/she wants its value to appear as a description for each result.

- **Results Characteristics:** A list of classes IRIs and their images. If the result is a subclass of one of these classes in the list, then the result will have the image corresponding to the class, and it could have multiple images. This shows what the characteristics of each result are.

- **Don’t Show Result:**

  The need for this information in the configuration file is due to the fact that you might have classes that help you while organising your model within your ontology. For example, in the Sushi Ontology there are Spicy Sushi, Vegetarian Sushi, Spicy Platter and so on. Such classes are things that you would expect to be used in queries but would not expect to have them as results of something. These classes are somehow abstractions; they are not real sushi or platters that a customer can buy. Therefore, we need to represent the fact that some of the classes in the ontology are interesting to the application and some of them are not. There are a number of ways to reconcile this issue. One possible way is to annotate the ontology to have an indication that some of the classes in the ontology are abstractions and some of them are not, but these annotations may just be about this particular usage of the ontology. Another way is to list all of the abstract classes in the configuration file, but we may end up with a very long list. In fact, the first way is used here, although it is not clear which one is the ideal solution. So this configuration information contains the IRI of an annotation property and it is value. This annotation in the ontology indicates that this class shouldn’t appear in the user interface. When the finder application shows the results, any class that is annotated with the specified value will not appear.

- **Sanctions Assertions:** A list of assertions of the following form: the IRI of class C, the IRI of property R, the IRI of class D. These assertions are used by the Sanctions Component to perform the sanctioning-like mechanism.

Besides these, the file might contain the following optional appearance information:
- Logo URL.
- Labels’ texts for the restaurant’s title, included label and excluded label.

### 3.7. Chapter Summary

This chapter presents the system design for The Manchester Sushi Finder, starting with the system design goals and requirements. It also includes a detailed description about the system architecture and its components.
Chapter 4  System Implementation

4.1. Chapter Introduction

This chapter focuses on implementation of The Manchester Sushi Finder, based on the design described in Chapter 3. This chapter describes the implementation platform and tools besides the mechanisms used in implementing some significant parts of the system. General ideas and screenshots of some functionalities of the system can also be found in this chapter.

4.2. Implementation Platforms and Programming Languages

NetBeans Integrated Development Environment (IDE)

The Manchester Sushi Finder is a Java application which was developed in NetBeans IDE 7.3.1. NetBeans is a free, open-source IDE. It is compatible with all operating systems that support Java, such as: Mac OS, Windows and Linux. It has many features that ensure efficient project development and management. It has an easy graphical user interface design which uses drag and drop tools. In addition to a feature that has been used within this project which is versioning control; NetBeans allows one to work with well-known versioning control systems from one’s project within the IDE [21].

Versioning Control Systems - Git:

An important aspect of software development is adopting a backup strategy for the source code that stores different versions of the software.

“A tool that manages and tracks different versions of software or other content is referred to generically as a version control system (VCS)” [22].

Such systems allow users to create and maintain a repository for their software’s contents and provide a log that stores all of the changes and historical editions for each file.

Git is one of those version control tools. It was invented by Linus Torvalds and supports a distributed development model where a number of developers can contribute. Git ensures the integrity of its data repositories by using a cryptographic hash function. It can be installed on a number of operating systems such as Windows and Linux [22].
The Git support feature within Netbeans has been used while developing the finder application.

The Git repository is a database that stores all of the information required to keep and control the project’s history and revisions. This information is stored in a subdirectory (named `.git`) at the root of the project’s working directory. This is called the current or local repository [22].

Using Git, a file in one’s working directory can be in one of the following three states:

- Committed: it is stored in the local repository.
- Modified: one made changes to the file but didn’t commit it to the local repository.
- Staged: one marked a modified file so that it will go into one’s next commit action [23].

Moreover, Git allows one to share repositories in order to support the distributed development model. A remote repository can be established and Git can transmit data from the local repository to the remote repository. To establish a Git repository, one can either create a new empty repository or clone or copy an existing one [22].

GitHub is a web-based Git hosting service established in 2008 for software projects that use Git version control. The Manchester Sushi Finder repository is hosted in GitHub and can be accessed from the following link:

https://github.com/AtheerAlgherairy/TheManchesterSushiFinder

**Maven**

Maven is a tool which is based upon the project object model (POM) concept and has been used to manage the project’s build and dependencies. Maven uses a pom.xml file to store project configuration information, including the list of dependencies and plugins that the build process required. Maven can be installed and integrated with Netbeans IDE [24].
**XMLBeans**

XMLBeans is an open-source tool which allows developers to easily create and manipulate XML documents in Java. Basically, it is a schema compiler which parses an XML schema (stored in a XSD file) and generates a Java Archive file (JAR) which contains a number of Java classes that correspond to the types defined within the schema. Then, developers can use these generated classes to create or manipulate XML documents that conform to that schema [25].

**OWL API**

Described previously in 2.7.1. in Chapter Two.

### 4.3. Query Template Engine

The *Query Template Engine* is a command line Java program that does the main function in the application. As described earlier, the query interface passes the following information to the *Query Template Engine*:

- List of included OWL class expressions.
- List of excluded OWL class expressions.
- Selected template.

Then, the engine in turn applies the *Mapping Algorithm* to the templates stored in the configuration file in order to construct a proper class expression for the query.

#### 4.3.1. The Mapping Algorithm

There are two kinds of query templates: simple and complex. A simple template contains: base class (B), property (P) and component class (C), which corresponds to a base thing related directly to its components through one object property. On the other hand, a complex template contains: base class (B), property (P) and other template (T), which corresponds to a base thing related to its components through a number of object properties.

Simple template: $T_S = \{B_S, P_S, C_S\}$.

Complex template: $T_X = \{B_X, P_X, T\}$.
The Mapping Algorithm takes a template $T$, a list of included expressions $\{I_i\}$ and a list of excluded expressions $\{E_i\}$. Then, it generates the expression as follows:

$$\text{Expression } (T, \{I_i\}, \{E_i\}) =$$

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $T$ is simple:</td>
<td>$B_S \land \exists P_S . I_i$</td>
</tr>
<tr>
<td></td>
<td>$\land \neg \exists P_S . E_i$</td>
</tr>
<tr>
<td>Else if $T$ is complex:</td>
<td>$B_X \land \exists P_X . (\text{GetInclusionExpr}(T_X, I_i))$</td>
</tr>
<tr>
<td></td>
<td>$\land \neg \exists P_X . (\text{GetInclusionExpr}(T_X, E_i))$</td>
</tr>
</tbody>
</table>

$$\text{GetInclusionExpr } (T, I_i) =$$

<table>
<thead>
<tr>
<th>Condition</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>If $T$ is simple:</td>
<td>$B_S \land \exists P_S . I_i$</td>
</tr>
<tr>
<td>Else if $T$ is complex:</td>
<td>$B_X \land \exists P_X . (\text{GetInclusionExpr}(T_X, I_i))$</td>
</tr>
</tbody>
</table>

In The Manchester Sushi Finder, these query templates are represented in XML and stored in the configuration XML file. Each simple template is represented as an element called $\text{SimpleTemplate}$, and this element has the ID and name attributes, and it also has three child elements that correspond to base class, property and the component class. On the other hand, each complex template is represented as an element called $\text{ComplexTemplate}$, and this element has the ID and name attributes, and it also has three child elements that correspond to base class, property and reference to other template.

These query templates depend on the ontology. For the Sushi Ontology, explained previously in 2.4. in Chapter Two, there are three base things: sushi, platters of sushi and trays of sushi. Therefore, there is one $\text{SimpleTemplate}$ element and two $\text{ComplexTemplate}$ elements under the $\text{QueryTemplates}$ element. They are as follows:
Only for the purpose of simplification are the short forms of the classes and properties hereby used instead of their IRIs.

The query template engine applies the mapping algorithm to these stored templates. The inputs to the algorithm are a list of included expressions \{I_i\}, a list of excluded expressions \{E_i\}, and an XML element that correspond to a selected template. The output is a class expression to be used for the query.

The following two examples show two executions of the algorithm with two different inputs to demonstrate that the algorithm generates proper expressions where the negation occurs in the correct place.

**Example 1- simple template:**

Inputs:

- List of included expressions: \{Salmon, Rice\}
- List of excluded expressions: \{Chicken, Onion\}
- Selected template ID = 1

Applying the algorithm:
Expression is the intersection over:

- NamedSushi.
- hasIngredient some Salmon
- hasIngredient some Rice
- not(hasIngredient some Chicken)
- not(hasIngredient some Onion)

The output:

Expression= NamedSushi and hasIngredient some Salmon and hasIngredient some Rice and not(hasIngredient some Chicken) and not(hasIngredient some Onion)

Example 2- complex template:

Inputs:

- List of included things: {Duck, Rice}
- List of excluded things: {Chives}
- Selected template ID = 2

Applying the algorithm:

Expression is an intersection over:

- SushiPlatter
- hasContent some (apply getInclusionExpr (template ID=1, Duck))
- hasContent some (apply getInclusionExpr (template ID=1, Rice))
- not (Property some (apply getInclusionExpr (template ID=1 , Chives)))

After applying getInclusionExpr method:

Expression is an intersection over:

- SushiPlatter
- hasContent some(NamedSushi and hasIngredient some Duck)
- hasContent some(NamedSushi and hasIngredient some Rice)
- not(hasContent some(NamedSushi and hasIngredient some Chives))

The output:

Expression= SushiPlatter and hasContent some(NamedSushi and hasIngredient some Duck) and hasContent some(NamedSushi and hasIngredient some Rice) and not(hasContent some(NamedSushi and hasIngredient some Chives))

In order to avoid errors that might occur when applying the mapping algorithm, we need to ensure that the templates are stored properly in the XML. This is done using XMLBeans. The XMLBeans uses an XML schema and gives the ability to validate an XML file against this schema.

4.4. The Query Interface

The following figure (Figure 4-1) shows the query interface that is generated based on the information in a sample configuration file (config.xml). The default configuration file is illustrated in Appendix A.

The query interface consists of three main parts:

- The left side of the interface is for browsing the ingredients.
- The right side of the interface is for including and excluding ingredients.
- The bottom side of the interface is for selecting one of the base things to query about as well as selecting one of the available languages.
4.4.1. Browsing the Ingredients

By default, the query interface has Tree and Search view tabs to browse the ingredient, and it could have two more views based on the information in the used configuration file. In particular, the Classes view tab is constructed if there are classes specified under IngredientsClassifications element in the configuration file. The facets view tab is constructed if there are object properties specified under IngredientsFacets element in the configuration file.

The contents of these tabs differ according to the information in the used configuration file. Based on a sample configuration file (i.e. config.xml), the interface in Figure 4-1 has all the four tabs, and the contents of these tabs are illustrated in Figure 4-2.
Figure 4-2: Different View Tabs in the Query Interface
- **Tree:**
  This is a default view. It is generated using the reasoner to obtain the subclasses hierarchy of the ingredient class specified in the configuration file.

- **Search:**
  This is a default view. It is generated by the *Search Component*. The component uses the reasoner to obtain the subclasses hierarchy of the ingredient class and then flatten them in a list. As seen in the *Search view* tab in Figure 4-2, it has a search box that performs an incremental search on the ingredients list. As the user types text in the search box, the list shrinks to show only the possible matches for the text.

- **Classes:**
  This view is generated using the *Classes Filtering Component* and presents some predetermined selections that could help the end user to easily find the ingredient.

The sample configuration file (i.e. config.xml) has the following four classifications under *IngredientsClassifications* element:

```xml
<IngredientsClassifications>
  <Classification class="http://www.ontologies.org/Sushi.owl#VegetarianIngredient" />
  <Classification class="http://www.ontologies.org/Sushi.owl#SpicyIngredient" />
  <Classification class="http://www.ontologies.org/Sushi.owl#Meat"/>
  <Classification class="http://www.ontologies.org/Sushi.owl#SeaFood"/>
</IngredientsClassifications>
```
Therefore, the Classes view tab in Figure 4-2 has four checkboxes and a list of all ingredients. When any of the checkboxes are checked or unchecked, then the component uses the reasoner to obtain all of the ingredients that are subclasses of the intersection of the selected classes and then rebuild the list of ingredients again.

- Facets:
The Facets Component uses a list of properties. It gives the user the ability to find an ingredient based on its characteristics.

The sample configuration file (i.e. config.xml) has the following four properties under IngredientsFacets element:

```xml
<IngredientsFacets>
  <Facet property="http://www.ontologies.org/Sushi.owl#hasSpiciness"/>
  <Facet property="http://www.ontologies.org/Sushi.owl#hasSweetness"/>
  <Facet property="http://www.ontologies.org/Sushi.owl#hasTexture"/>
  <Facet property="http://www.ontologies.org/Sushi.owl#hasCost"/>
</IngredientsFacets>
```

For each property, the Facets Component follows the following mechanism:

a. Create an expression, which is the intersection of the ranges of the property.

b. Get the subclasses of the created expression.

c. These subclasses will be rendered as checkboxes under the property name.

For example, the object property hasTexture in Manchester Syntax is as follow:
ObjectProperty: hasTexture
Domain: SushiIngredient
Range: Texture

This property has only one range class so no need to get the intersection of the ranges. The mechanism will generate all subclasses of Texture class and produce them as checkboxes under hasTexture property, they are: Chewy, Creamy, Cereal, Liquid and Crunchy. See the Facets view tab in Figure 4-2.

When any of the checkboxes are checked or unchecked, then the component will produce a class expression which is based on the selected checkbox and its property. Then, the component uses the reasoner to obtain all of the ingredients that are subclasses of the produced expression. Finally, the component rebuilds the list of ingredients again.

In Facets view tab in Figure 4-2, Creamy is selected. The produced expression is:

SushiIngredient and hasTexture some Creamy

When the user selects a number of checkboxes, the component will get the intersection of all produced expressions.

4.4.2. Including and Excluding Ingredients

As seen in Figure 4-1, the right side of the interface has two boxes: the first one is for the ingredients that the user wants to include in the base thing that he/she is querying about, while the second one is for the ingredients that the user doesn’t want to be included. The user can drag and drop ingredients to the corresponding boxes.

Moreover, The Manchester Sushi Finder has two significant characteristics:

1- Feedback mechanism:

The query interface provides an extensional feedback mechanism, which is one of the characteristics of ontology visual query systems. As the user adds ingredients
into, or removes ingredients from, the included/excluded boxes, the interface performs an on-the-fly calculation to show the number of results to be returned. The following figure shows the number of the results, based on some user selections.

![Sushi Box Interface](image)

**Figure 4-3**: Extensional Feedback Mechanism in the Query Interface

2- Sanctioning-like mechanism:

As described in 2.5.2 in Chapter Two, the need for such a mechanism is to allow the user to specify further qualities from the selected ingredients. These qualities cannot be determined before selecting the ingredient – unlike what the case is for the Facets view tab – because of two reasons:

- They differ from one base thing to another. For example, salmon could be fried in some platters and raw in others.
They are not applicable for all ingredients. For example, one can specify the cutting style of a meat ingredient but it is not reasonable to specify the cutting style of a sauce ingredient.

Implementing this mechanism enhances the richness of the Sushi Finder. It allows the user to include/exclude expressions not just named classes. As a result, it gives the user the ability to ask some questions that could not be asked before, such as the following:

- Find all of base things (e.g., sushi, pizza, etc.) that have spicy vegetarian ingredients in them.

- Find all of base things (e.g., sushi, pizza, etc.) that have fried chicken, seared beef, sliced salmon, etc. in them.

In The Manchester Sushi Finder, the method that is taken from [11] is implemented. The sanction assertions could be annotations associated with the ontology or in a separate layer. Since we have a configuration file which has some information about how to expect the ontology to be used, these sanction assertions are added in the configuration file.

The set of assertions is specified under the Sanctions element in the configuration file. They are of the form (C, R, D), where C and D are IRIs of classes within the ontology and R is an IRI of an object property within the ontology.

For example, the sample configuration file (i.e. config.xml) has the following two assertions under Sanctions element:

```xml
<Sanctions>
  <Sanction
    C="http://www.ontologies.org/Sushi.owl#Meat"
    R="http://www.ontologies.org/Sushi.owl#hasCookingStyle"
    D="http://www.ontologies.org/Sushi.owl#CookingStyle"/>
  <Sanction
    C="http://www.ontologies.org/Sushi.owl#Meat"
    R="http://www.ontologies.org/Sushi.owl#hasCuttingStyle"
    D="http://www.ontologies.org/Sushi.owl#CuttingStyle"/>
</Sanctions>
```
The sanctioning-like mechanism is performed using the *Sanctions Component* as follows:

When the user drops an ingredient into included/excluded box, the query interface passes this ingredient, the ontology and the list of assertions founded in the configuration file to the *Sanctions Component*.

The *sanctions component* uses *(getSetOfReasonableClasses)* function to find the minimal non-redundant set of class expressions that might reasonably be conjoined with the selected ingredient class. These class expressions are of the form \( \exists R.D \) where \( R \) is a property in the ontology and \( D \) is a class in the ontology.

The pseudo codes of the used function along with two helper functions are as follows:
Function getSetOfReasonableClasses
Pass In: Ingredient, Ontology, Assertions
Initialise setOfExpressions
Initialise newSetOfExpressions

For each class X in Ontology
   Initialise expr
   Set expr to Call Reasonable (Ingredient, X, Assertions)
   IF expr is not empty
   IF (Ingredient \x22\x22 X) is satisfiable AND Ingredient is not subclass of (Ingredient \x22\x22 X)
   Then Add expr to setOfExpressions
   EndIf
   EndIf
EndFor
Set newSetOfExpressions = Call fixTautological (setOfExpressions)
Return newSetOfExpressions
EndFunction

Function Reasonable
Pass In: A, B, Assertions
Initialise Expr to empty
   If the list of assertions is not empty
   For each entry in assertions on the form (C, R, D):
      If A isSubClassOf C AND B isSubClassOf D
      Then Set Expr to \exists R . B and Stop for loop
      EndIf
   EndFor
   EndIf
Return Expr
EndFunction
**Function** fixTautological

**Pass In:** setOfExpressions

**Initialise** newSetOfExpressions

**For** i=1 to the size of setOfExpressions

**Set** flag to True

**Set** Expr1= (∃R.D)_i (the i^{th} expression in the set)

**For** k=1 to the size of setOfExpressions

**Set** Expr2= (∃R.D)_k (the k^{th} expression in the set)

**IF** D in Expr2 isSubClassOf D in Expr1

**AND** they are not equal

**Then Set** flag to False

**EndIF**

**EndFor**

**IF** Flag is True

**Then Add** Expr1 to newSetOfExpressions

**EndIF**

**EndFor**

**Return** newSetOfExpressions

**EndFunction**

Figure 4-4 and Figure 4-5 show screenshots of how the described mechanism works in the query interface. These screenshots are based on the information and the assertions in the sample configuration file (i.e. config.xml). Figure 4-5 illustrates how the query interface takes expressions not only named classes to be included or excluded.
Figure 4-4: Implementation of Sanctioning-like Mechanism

Figure 4-5: Included Expression in the Query Interface
4.4.3. Switching Between Base Things and Languages

The bottom side of the query interface is for selecting one of the base things to query about as well as selecting one of the available languages. The available base things are taken from the query templates listed in the configuration file (See 4.3. in Chapter Four) and the available languages are also specified in the configuration file.

Referring back to the query interface illustrated in Figure 4-1, which is generated based on the sample configuration file (i.e. config.xml), there are:

1- Three base things: Sushi, Platter of sushi and Tray of platters. This is based on the templates specified in config.xml.

After selecting ingredients and choosing a base thing, the Query Template Engine is used to construct the query expression and obtain the right results.

Figure 4-6, Figure 4-7, Figure 4-8 and Figure 4-9 illustrate the ability of switching between Sushi and Platter of Sushi, and obtaining their corresponding results using the Query Template Engine.

When the query interface shows the results of any query, then it potentially shows some icons and descriptions along with each result (see Figure 4-7 and Figure 4-9). As mentioned earlier in 3.6. in Chapter Three, these icons and descriptions are based on some information in the configuration file, namely, ResultAnnotationProperty and ResultsCharacteristics elements.

2- Two languages: Arabic and Spanish. In addition to the default option.

When the user selects a language, then all classes and properties appear in the interface will be rendered in the selected language.

The default option means no language; this renders classes/properties using their short form of IRIs or a label with no language tag.

The mechanism that is followed by The Manchester Sushi Finder to decide how to render classes/properties is as follows:
- If the class/property has a label annotation with the selected language, then the application uses this label. Otherwise, based on the information in the configuration file (i.e. default class rendering), the priority is to use the short form of the IRI or use a label annotation with no language tag.

- If the class/property has no label annotations at all, then the short form of its IRI will be used.

Figure 4-10 and Figure 4-11 illustrate the ability of switching between available languages.

![Figure 4-6: Selecting Base Thing (Sushi)](image-url)
### Sushi Box

<table>
<thead>
<tr>
<th>Item</th>
<th>Ingredients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inari Pocket Nigiri</td>
<td>Cucumber, Egg, Inari, Radish and Rice</td>
</tr>
<tr>
<td>Soft Shell Crab Handrolls</td>
<td>Nori, Radish, Rocket, Sweet Chilli Mayo, Fish Roe and Fried Soft shell Crab.</td>
</tr>
<tr>
<td>Vegetable Futomaki</td>
<td>Cucumber, Carrot, Egg, Nori, Radish and Rice</td>
</tr>
</tbody>
</table>

**Figure 4-7: The Results (Sushi)**
Figure 4-8: Selecting Base Thing (Platter of Sushi)
Inari Pocket Nigiri Platter

Two of Inari Pocket Nigiri: A pocket made from inari (fried soya bean), filled with rice, tamago, cucumber and pickled radish. 184 kcal.

Soft Shell Crab Hand Rolls Platter

A seaweed (nori) cone filled with rice and Tempura fried soft shell crab, rocket, pickled radish, yuzu tobiko and sweet chilli mayonnaise. 211 kcal.

Vegetable Futomaki Platter

2 pieces of vegetable futomaki: large rice filled rolls wrapped in seaweed (nori) and filled with cucumber, pickled radish, tamago, avocado and carrot. 188 kcal.

Figure 4-9: The Results (Platter of Sushi)
Figure 4-10: Switching to Arabic Language

Figure 4-11: Switching to Spanish Language
4.5. The Configuration Tool

The implementation of this tool depends highly upon XMLBeans, which facilitates the creation and manipulation of XML documents. The configuration tool provides the tool provider with a graphical interface for creating, modifying or selecting the configuration file to be used by the application. Moreover, in all of these options, the tool ensures that the created, modified or selected configuration file conforms to a designated schema. These options can be accessed from the (Sushi Finder) menu in the query interface as per the following figure:

![Figure 4-12: The Configuration Tool Options](image)

The following screenshots show the execution of the (Select Configuration File) option provided by the configuration tool. The selected file in Figure 4-13 is for the Pizza ontology; which is used in the Pizza Finder, and this file is created using the (New Configuration File) option provided by the tool.
Figure 4-13: Select Configuration File (A)

Figure 4-14: Select Configuration File (B)
The (New Configuration File) interface has six tabs. Each tab is for a specific part of the new configuration file. The following screen shots show two tabs of the interface:

Figure 4-15: New Configuration File – Different Views Tab

Figure 4-16: New Configuration File – Results Tab
4.6. Issue Faced During Development

During the development of The Manchester Sushi Finder, one issue to consider is whether some configuration information should be treated as part of the ontology or as an application specific behaviour. This decision is used to determine where to store some configuration information: whether to store them as annotations embedded in the ontology or in a separate layer (in a configuration file). Although it is difficult to judge which method is superior, some tradeoffs are discussed here to justify the design decisions in The Manchester Sushi Finder.

For instance, the sanctions assertions could be stored in the configuration file or as annotations in the ontology.

Storing them in the configuration file provides some flexibility; it allows a tool provider to apply a different set of sanctions for the same ontology, so the application will have different scenarios or behaviours depending on the person who configures the tool. For example, if a restaurant would like to give customers only the meat cooking style options and not the cutting style options, then the tool provider could do this by specifying only the cooking style assertion in the configuration file. However, more work must be done to determine how to manage and find these assertions, especially if a third party uses the ontology since he/she needs to find which part of the ontology is interesting for application purposes.

On the other hand, having some configuration information as annotations in the ontology has some value. This means fewer management issues and makes it easier to find the interesting parts of the ontology for application purposes, but this also means that this information becomes increasingly associated with the ontology, and thus it becomes harder to have different information for different application scenarios.

The same issue exists for the facets properties and classifications, which are stored in the configuration file and used in generating the facets and classes view tabs. The flexibility to accommodate different application scenarios is more important here.

In contrast, for don’t show results, this information is more closely related to the ontology rather than the application. Ontology developers may include some abstract classes to organise their hierarchies, but these are not conceptually important because
they are not real concepts in the domain. Therefore, annotating such classes to indicate that they are not real concepts in the domain is acceptable in different application scenarios.

4.7. Chapter Summary

In this chapter, the platform and tools used for implementing The Manchester Sushi Finder have been explained. Additionally, the mechanisms used in implementing the main functions of the application have been explained. Screenshots of the user interface that show the execution of the explained mechanisms are also shown in this chapter. The following table summarises the functionalities of the application.

| Query Interface | • Provide four ways to view the components (ingredients) class: facets, search, class-filtering and tree.  
| | • Allow end user to construct queries by:  
| | - Selecting a base thing.  
| | - Including and excluding named classes and/or class expressions (using the sanctions).  
| | • view the results  
| | • Switch the language.  
| Configuration Tool | • Create new configuration file.  
| | • Modify configuration file.  
| | • Select configuration file to be used by the application. |

Table 3: A Summary of the Application’s Functions
Chapter 5 Testing and Evaluation

5.1. Chapter Introduction

This chapter covers the tests conducted during the development of the application. These tests aimed to ensure that the application performed as expected. In addition, the evaluation carried out after the development phase was completed. This was to determine the usefulness of the application. It examined the flexibility and usability of the Sushi Finder, and made a comparison between the existing Pizza Finder and the Sushi Finder.

5.2. Testing

To ensure that the system fulfils the identified requirements and behaves as expected, the following tests were performed after implementing each part of the system:

Query Template Engine Test: This test is done by using different inputs (i.e., base things and ingredients) and comparing the results of query expressions produced by the engine with the results of manually written query expressions in the DL Query tab in Protégé. This is to ensure that the engine constructs the right expressions that return the right query results.

Query Interface Test: A number of manually written configuration files that have different information are used to check that the interface behaves as expected according to the information contained in each configuration files. For instance, these files differ in the number of facets’ properties, ingredient classifications, sanction assertions, logos, and so on. Accordingly, the interface differs in the number of shown facets, classifications, logos, and so on.

Configuration Tool Test: The tool has been tested by uploading ontologies, ensuring that all the combo lists are filled with the correct information taken from the ontologies, and examining the created XML file to ensure that it conforms to the required schema. Moreover, the ability to read and modify valid XML files has also been tested.

5.3. Evaluation
The project aimed to provide the students of the Ontology course with an experimental platform from which they could browse and query their ontologies. Therefore, the flexibility of the application needed to be evaluated. The project also aimed to provide non-experienced users (i.e., restaurant customers) with a tool they could use to ask questions to find dishes that meet their preferences without having to deal with the underlying representations or the concrete syntax of the ontology. Therefore, the usability of the application was evaluated.

- **Flexibility of The Manchester Sushi Finder:**

  The flexibility of the finder application was evaluated by sending it to four Ontology course students. They were asked to create configuration files to their ontologies and configure the tool to work with their files. All were able to upload their ontologies, define query templates and configure the tool to work with their ontologies regardless of the ease or difficulty of the usage, which will be discussed in the usability evaluation.

  Moreover, a configuration file for the pizza ontology that is used by the Pizza Finder was created and used by the application (see Figure 4-14).

- **Usability of The Manchester Sushi Finder:**

  Because there are two targeted users of The Manchester Sushi Finder, the usability of the finder was evaluated in two aspects:

  1- **Experiment for YO! Sushi customers.**

     This experiment evaluated whether the application allows non-experienced users to ask questions against the ontology, and also whether the use of such finders within restaurants would be useful and time efficient for customers.

     Ethical approval was obtained for this experiment. The participants were a total of seven men and women aged from 18 to 30 who had been to the *YO! Sushi* restaurant before.

     The participants were provided with the *YO! Sushi* menu (a paper menu) and the finder application. Then, they were given four questions to answer.

     The questions were of form ‘Find all sushi platters that have X in them’ or ‘Find all sushi platters that have X and do not have Y in them’.

73
The participants were asked to answer these questions by writing down the number of results they got, first using the menu and then using the finder application. The evaluator calculated the time the participants spent using the menu and the finder to answer each question. The questions and the times spent by all seven participants in using the menu and the finder to answer these questions are as follows:

**Question 1:** Find all sushi platters that have fried chicken in them.

There was only one platter that had fried chicken, and all of the participants answered the question correctly using both the finder and the menu. The following table shows the times (in seconds):

<table>
<thead>
<tr>
<th>Q1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>AVG</th>
<th>AVG (Minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu</td>
<td>75</td>
<td>96</td>
<td>100</td>
<td>60</td>
<td>80</td>
<td>86</td>
<td>83</td>
<td>83</td>
<td>1.39</td>
</tr>
<tr>
<td>Finder</td>
<td>27</td>
<td>19</td>
<td>73</td>
<td>13</td>
<td>29</td>
<td>32</td>
<td>28</td>
<td>32</td>
<td>0.53</td>
</tr>
</tbody>
</table>

*Table 4: Times Spent in Answering Question One*

**Question 2:** Find all sushi platters that have prawn in them and do not have wasabi in them.

The answer is three platters. Five of the participants answered the question correctly using both the finder and the menu, whereas two of them did not get the correct answer using the menu. The following table shows the times (in seconds):

<table>
<thead>
<tr>
<th>Q2</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>AVG</th>
<th>AVG (Minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu</td>
<td>77</td>
<td>107</td>
<td>135</td>
<td>74</td>
<td>149</td>
<td>122</td>
<td>143</td>
<td>115</td>
<td>1.92</td>
</tr>
<tr>
<td>Finder</td>
<td>37</td>
<td>18</td>
<td>27</td>
<td>39</td>
<td>39</td>
<td>22</td>
<td>41</td>
<td>32</td>
<td>0.53</td>
</tr>
</tbody>
</table>

*Table 5: Times Spent in Answering Question Two*

**Question 3:** Find all sushi platters that have tuna in them but do not have seared tuna in them.

The answer is eleven platters. However, only two participants answered the question correctly using both the finder and the menu, whereas the majority (five
of them) did not get the correct answer using the menu. The following table shows the times (in seconds):

<table>
<thead>
<tr>
<th>Q3</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>AVG (Minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu</td>
<td>143</td>
<td>137</td>
<td>150</td>
<td>101</td>
<td>120</td>
<td>124</td>
<td>154</td>
<td>133</td>
</tr>
<tr>
<td>Finder</td>
<td>53</td>
<td>33</td>
<td>32</td>
<td>40</td>
<td>30</td>
<td>15</td>
<td>20</td>
<td>32</td>
</tr>
</tbody>
</table>

**Table 6:** Times Spent in Answering Question Three

**Question 4:** Find all sushi platters that have salmon in them and do not have avocado in them.

The answer is eight platters. Four of the participants answered the question correctly using both the finder and the menu, whereas three of them did not get the correct answer using the menu. The following table shows the times (in seconds):

<table>
<thead>
<tr>
<th>Q4</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>AVG (Minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu</td>
<td>135</td>
<td>119</td>
<td>131</td>
<td>160</td>
<td>170</td>
<td>127</td>
<td>152</td>
<td>142</td>
</tr>
<tr>
<td>Finder</td>
<td>30</td>
<td>38</td>
<td>27</td>
<td>60</td>
<td>60</td>
<td>29</td>
<td>60</td>
<td>43</td>
</tr>
</tbody>
</table>

**Table 7:** Times Spent in Answering Question Four

Notably, all of the participants were able to answer questions using the finder. Moreover, all of the participants always obtained the right answer using the finder, but they would miscount some platters using the menu. Furthermore, for all the participants in all the questions, the time spent in answering the questions using the menu was more than the time spent using the finder. Therefore, we can say that the finder improves the accuracy and reduces the time.

After the experiment was conducted, some feedback about the finder application was taken from the participants by giving them a simple questionnaire to fill out. The experiment, questionnaire and detailed results analysis of the questionnaire can be reviewed in Appendix B.

After analysing the questionnaire’s results, it can be noticed that the participants preferred to use different view tabs in finding the required ingredients: Three of
them preferred to use the Search view, whereas four of them preferred to use the Tree view.

Moreover, some of the participants expressed their admiration of the finder application, with the majority of them confirming that the usage level of the sushi finder is easy and they took around five minutes to learn how to use it. Furthermore, all of them agreed that using such finders within restaurants would be helpful for customers.

2- Questionnaire for Ontology Course Students.

As mentioned earlier, four Ontology course students were provided with the finder application and a configuration manual. They were asked to follow the instructions in the manual to create configuration files to their ontologies and configure the tool to work with their files. Thereafter, they were asked to fill out a questionnaire to evaluate the usability of the configuration tool. Ethical approval was obtained for this questionnaire. The questionnaire and its detailed results analysis can be reviewed in Appendix B.

After analysing the questionnaire’s results, it can be seen that the majority of the students found the configuration interface to be moderately friendly, and they thought that it was easy to configure the tool to work with their ontologies. Moreover, the majority of them took from ten to twenty minutes to accomplish this configuration task. Furthermore, the majority agreed that Oncology course students could use The Manchester Sushi Finder as an experimental platform from which they can browse and query their ontologies.

5.4. Comparison between the Pizza Finder and the Sushi Finder

The characteristics of the ontology visual query systems that are identified by Bechhofer and Paton (2009) [6] are used to compare the existing Pizza Finder with the new Sushi Finder. They are as follows:

1- Identification of Starting Points

In the Sushi Finder, the tool provider configures the application by adding a number of query templates that are mostly based on existing base things
modelled in the ontology. Then, the end user can choose one of those base things, so the user selects one of the available starting points from which the query can be constructed. However, in the Pizza Finder, there is always only one starting point from which the query can be constructed.

2- Query Modification Operations

In the Sushi Finder, the end user can do some query modification operations:

**Filler Specialisation:** By selecting a number of included and/or excluded ingredients, these ingredients are used as fillers that specialise a named concept in the generated query expression.

**Property Addition:** Using a sanction-like mechanism, the user can modify the query expression and add relationship(s) to the expression in order to determine any further restrictions on the selected ingredients.

However, in the Pizza Finder, the end user can only do the filler specialisation by selecting the number of included and/or excluded toppings.

3- Feedback Mechanism

In terms of the feedback mechanism, both of the finders have an intentional feedback mechanism that prevents the user from constructing an unsatisfiable query expression that returns no results, such as preventing the user from selecting the same ingredient/topping as both an included and excluded thing.

On the other hand, the Sushi Finder has also an extensional feedback mechanism, which is not in the Pizza Finder. The query interface in the Sushi Finder shows the number of results to be returned based on the selections made by the end user.

Some additional features in the Sushi Finder are the Search view and the Facets view that help the user find the ingredients faster. Furthermore, the Sushi Finder provides a configuration tool to create and modify the configuration files using a graphical interface and ensuring that the created/modified files conform to the schema rather than working with the XML files manually, which could be error prone. Additionally, the dynamic configuration provided by the Sushi Finder makes the configuration process
easier; the user can select another configuration file while the application is running.

5.5. Chapter Summary

The tests were performed during the implementation to ensure that each part of the system behaved as expected. Additionally, the flexibility and usability of The Manchester Sushi Finder were evaluated. Moreover, using the criteria discussed in 2.5.1 in Chapter Two, this chapter concludes with a comparison between the Sushi Finder and the Pizza Finder. This comparison demonstrates positive results in terms of the functionalities of the Sushi Finder.
Chapter 6 Conclusion and Future Work

The main aim of the project is to develop a Java application, *The Manchester Sushi Finder*. It extends *The Manchester Pizza Finder* by adding new features and more functionality.

The *Manchester Sushi Finder* is now flexible to work with ontologies that have been developed based on different restaurants’ menus and also configurable in many areas. These ontologies, consumed by the application, can represent a number of base things that are related to their components directly through one object property or indirectly through a number of object properties. Therefore, *The Manchester Sushi Finder* can be used by students of the Ontology course as an experimental platform to see how their ontologies could be consumed within an application.

Moreover, the experiments done in the evaluation shows that *The Manchester Sushi Finder* allows non-experienced users to easily construct proper queries against the ontologies and obtain reasonable results without having to deal with the underlying representations or the concrete syntax of the ontology.

The dissertation covers the entire development process of *The Manchester Sushi Finder*: the requirements, design, implementation and evaluation. Moreover, the main objectives of the project have been achieved, and the evaluations were satisfactory.

*The Manchester Sushi Finder* application (v1.0) can be downloaded from:

https://github.com/AtheerAlgherairy/TheManchesterSushiFinder/tree/v1.0

In the future, the application can be improved in terms of:

- **The appearance of the query results:**

  After constructing the query expression and retrieving the right objects, the finder uses a specified annotation property to provide a short description of the retrieved objects. This could be enhanced by developing a mechanism that breaks down each object into its constituents in the results. For instance, each platter in the results should be broken down into the number of sushi pieces it contains, and each piece on the platter should be broken down into its ingredients. However, this poses a new
challenge which is how to produce natural language descriptions of retrieved objects in the results.

- The query templates mechanism:

  The Manchester Sushi Finder uses templates that allow us to have different base things, but for each base thing, we have only one single property to perform the inclusion and exclusion. The query templates mechanism could be improved to include multiple properties for each base thing. For instance, if a base thing has other constituents that are not ingredients, this means that further properties are required for these constituents. This improvement would allow customers to query about base thing by specifying a number of different constituents that the base thing might have, such as ingredients and the cost of a platter.
References


Appendix A: Configuration File

1- Default Configuration File (config.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>

<FinderConfiguration>
  <OntologyLocation url="sushi_9223429_atheer.owl"/>
  <IngredientClass class="http://www.ontologies.org/Sushi.owl#SushiIngredient"/>
  <Logo URL="/SushiLogo.png"/>
  <TitleLabel text="Sushi Box"/>
  <IncludedLabel/>
  <ExcludedLabel/>
  <AvailableLanguages>
    <language name="default"/>
    <language name="ar"/>
    <language name="es"/>
  </AvailableLanguages>
  <QueryTemplates>
    <SimpleTemplate ID="1" name="sushi">
      <BaseClass name="http://www.ontologies.org/Sushi.owl#NamedSushi"/>
      <Property name="http://www.ontologies.org/Sushi.owl#hasIngredient"/>
      <ComponentClass name="http://www.ontologies.org/Sushi.owl#SushiIngredient"/>
    </SimpleTemplate>
    <ComplexTemplate ID="2" name="platter of sushi">
      <BaseClass name="http://www.ontologies.org/Sushi.owl#SushiPlatter"/>
      <Property name="http://www.ontologies.org/Sushi.owl#hasContent"/>
      <Template ref="1"/>
    </ComplexTemplate>
    <ComplexTemplate ID="3" name="Tray of platters">
      <BaseClass name="http://www.ontologies.org/Sushi.owl#SushiTray"/>
      <Property name="http://www.ontologies.org/Sushi.owl#hasPlatter"/>
      <Template ref="2"/>
    </ComplexTemplate>
  </QueryTemplates>
</FinderConfiguration>
```
<ClassRendering use="label"/>

<IngredientsFacets>
  <Facet property="http://www.ontologies.org/Sushi.owl#hasSpiciness"/>
  <Facet property="http://www.ontologies.org/Sushi.owl#hasSweetness"/>
  <Facet property="http://www.ontologies.org/Sushi.owl#hasTexture"/>
  <Facet property="http://www.ontologies.org/Sushi.owl#hasCost"/>
</IngredientsFacets>

<IngredientsClassifications>
  <Classification class="http://www.ontologies.org/Sushi.owl#VegetarianIngredient"/>
  <Classification class="http://www.ontologies.org/Sushi.owl#SpicyIngredient"/>
  <Classification class="http://www.ontologies.org/Sushi.owl#Meat"/>
  <Classification class="http://www.ontologies.org/Sushi.owl#SeaFood"/>
</IngredientsClassifications>

<Sanctions>
  <Sanction
    C="http://www.ontologies.org/Sushi.owl#Meat"
    R="http://www.ontologies.org/Sushi.owl#hasCookingStyle"
    D="http://www.ontologies.org/Sushi.owl#CookingStyle"/>
  <Sanction
    C="http://www.ontologies.org/Sushi.owl#Meat"
    R="http://www.ontologies.org/Sushi.owl#hasCuttingStyle"
    D="http://www.ontologies.org/Sushi.owl#CuttingStyle"/>
</Sanctions>

<ResultsAnnotationProperty property="http://www.ontologies.org/Sushi.owl#description"/>

<DontShow annotationIRI="http://www.ontologies.org/Sushi.owl#dontShow" value="true"/>
<ResultsCharacteristics>
    <ResultsCharacteristic class="http://www.ontologies.org/Sushi.owl#SpicySushi"
        url="/spicyIcon.jpg"/>
    <ResultsCharacteristic class="http://www.ontologies.org/Sushi.owl#SushiWithNuts"
        url="/nutsIcon.jpg"/>
    <ResultsCharacteristic class="http://www.ontologies.org/Sushi.owl#VegetarianSushi"
        url="/vegIcon.jpg"/>
    <ResultsCharacteristic class="http://www.ontologies.org/Sushi.owl#SpicyPlatter"
        url="/spicyIcon.jpg"/>
    <ResultsCharacteristic class="http://www.ontologies.org/Sushi.owl#VegetarianPlatter"
        url="/vegIcon.jpg"/>
    <ResultsCharacteristic class="http://www.ontologies.org/Sushi.owl#LowCaloriesPlatter"
        url="/lowCalIcon.jpg"/>
</ResultsCharacteristics>
</FinderConfiguration>
Appendix B: Questionnaires

1- Experiment for YO! Sushi Customers:

Please use the provided YO! Sushi menu and the finder application to answer the following questions:

1- Find all sushi platters that have fried chicken in them?

<table>
<thead>
<tr>
<th>Number of platters:</th>
<th>Using the YO! Sushi Menu</th>
<th>Using the Sushi finder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which one did you find easier?

<table>
<thead>
<tr>
<th>Have you got the same answer?</th>
<th>• yes</th>
<th>• No</th>
</tr>
</thead>
</table>

2- Find all sushi platters that have prawn in them and do not have wasabi in them?

<table>
<thead>
<tr>
<th>Number of platters:</th>
<th>Using the YO! Sushi Menu</th>
<th>Using the Sushi finder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which one did you find easier?

<table>
<thead>
<tr>
<th>Have you got the same answer?</th>
<th>• yes</th>
<th>• No</th>
</tr>
</thead>
</table>

3- Find all sushi platters that have tuna in them but do not have seared tuna in them?

<table>
<thead>
<tr>
<th>Number of platters:</th>
<th>Using the YO! Sushi Menu</th>
<th>Using the Sushi finder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which one did you find easier?

<table>
<thead>
<tr>
<th>Have you got the same answer?</th>
<th>• yes</th>
<th>• No</th>
</tr>
</thead>
</table>

4- Find all sushi platters that have salmon in them and do not have avocado in them?

<table>
<thead>
<tr>
<th>Number of platters:</th>
<th>Using the YO! Sushi Menu</th>
<th>Using the Sushi finder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which one did you find easier?

<table>
<thead>
<tr>
<th>Have you got the same answer?</th>
<th>• yes</th>
<th>• No</th>
</tr>
</thead>
</table>
2- Questionnaire for YO! Sushi Customers:

1- Which kind of ingredient browsing in the sushi finder do you prefer to use in answering the previous questions?
   - Tree
   - Facets browsing
   - Classes filtering
   - Search box

2- Did you like the sushi finder user interface?
   - Yes.
   - No.
   - To some extent.

3- How would you describe the usage level of the sushi finder?
   - Easy
   - Acceptable
   - Hard

4- How long it takes you to learn how to use the sushi finder (query Interface)?
   - Less than 5 minutes
   - From 5 to 10 minutes
   - More than 10 minutes

5- Do you think using such finders within restaurants would be helpful for customers?
   - Yes
   - No
   - I don’t Know

6- What suggestions do you have for improving the sushi finder application?

Thank you; I really appreciate your help.
Kind regards,
Atheer Algherairy
3- The Detailed Results Analysis of Customers Questionnaire:

1- Which kind of ingredient browsing in the sushi finder do you prefer to use in answering the previous questions?

![Pie chart showing preferences for ingredient browsing methods.](chart1)

- Tree: 57%
- Facets Browsing: 0%
- Classes Filtering: 0%
- Search Box: 43%

2- Did you like the sushi finder user interface?

![Pie chart showing user interface preferences.](chart2)

- Yes: 71%
- No: 0%
- To some extent: 29%

3- How would you describe the usage level of the sushi finder?

![Pie chart showing usage level preferences.](chart3)

- Easy: 86%
- Acceptable: 14%
- Hard: 0%
4- How long it takes you to learn how to use the sushi finder (query Interface)?

- Less than 5 minutes: 71%
- From 5 to 10 minutes: 29%
- More than 10 minutes: 0%

5- Do you think using such finders within restaurants would be helpful for customers?

- Yes: 100%
- No: 0%
- To some extent: 0%
4- Questionnaire for Ontology Course Students:

Please use the finder application and then follow the instructions provided in the configuration manual to configure the finder to work with your own ontology.

1- Which kind of ingredient browsing in the sushi finder do you prefer?
   - Tree
   - Facets browsing
   - Classes filtering
   - Search box

2- How user-friendly is the sushi finder interface?
   - Extremely user-friendly
   - Moderately user-friendly
   - Slightly user-friendly
   - Not at all user-friendly

3- How long it takes you to learn how to use the query interface in the sushi finder?
   - Less than 5 minutes
   - From 5 to 10 minutes
   - More than 10 minutes

4- Was it easy to configure the finder application to make it work with your ontology?
   - Yes
   - No
   - To some extent.

5- How long it took you to learn how to configure the finder application to make it work with your ontology?
   - Less than 10 minutes
   - From 10 to 20 minutes
   - More than 20 minutes

6- Do you think that the given instructions in the configuration manual were easy to follow?
   - Yes
   - No
   - To some extent.

7- In general, how would you describe the usage level of the sushi finder?
   - Easy
   - Acceptable
   - Hard
8- Overall, do you think that The Manchester Sushi Finder could be used as an experimental platform for ontology course students from which they can browse and query their ontologies?
   - Yes
   - No
   - To some extent

9- What suggestions do you have for improving the sushi finder application?

Thank you; I really appreciate your help by answering this questionnaire
Kind regards,
Atheer Algherairy
5- The Detailed Results Analysis of Ontology Course Questionnaire:

1- Which kind of ingredient browsing in the sushi finder do you prefer?

- Tree: 75%
- Facets Browsing: 0%
- Classes Filtering: 25%
- Search Box: 0%

2- How user-friendly is the sushi finder interface?

- Extremely user-friendly: 0%
- Moderately user-friendly: 25%
- Slightly user-friendly: 0%
- Not at all user-friendly: 75%

3- How long it takes you to learn how to use the query interface in the sushi finder?

- Less than 5 minutes: 100%
- From 5 to 10 minutes: 0%
- More than 10 minutes: 0%
4- Was it easy to configure the finder application to make it work with your ontology?

5- How long it took you to learn how to configure the finder application to make it work with your ontology?

6- Do you think that the given instructions in the configuration manual were easy to follow?
7- In general, how would you describe the usage level of the sushi finder?

8- Overall, do you think that The Manchester Sushi Finder could be used as an experimental platform for ontology course students from which they can browse and query their ontologies?