University of Manchester

School of Computer Science

Degree Program of Advanced Computer Science

THE MANCHESTER SUSHI FINDER

Hani Al Abbas

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

Master’s Thesis

2014

Table of Contents

Word Count: 20531

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Programming Interface
DL	Description Logics
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JVM	Java Virtual Machine
KR	Knowledge Representation
OWL	Web Ontology Language
POM	Project Object Model
RDF	Resource Description Framework
UI	User Interface
UML	Unified Modelling Language
W3C	World Wide Web Consortium
XML	Extensible Markup Language

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This project is designed to show the benefits of using ontology-driven applications in terms of browsing and querying for information. It demonstrates the use of the represented knowledge between machines instead of simply sharing raw data. Since the web is full of raw information that may or may not be relevant to each other, Web Ontology Language (OWL) came to represent the knowledge of domains instead of raw data. In doing so, relations between different objects within a domain are represented as well. This project makes use of ontology that represents knowledge in a taxonomical hierarchy. The keywords do not need to be remembered as they are presented in the hierarchy. The project is about developing a tool that adopts an ontology-based method and a faceted search mechanism using OWL annotations to store the configurations. The representation of knowledge with ontologies will drive this tool using some standard annotations, which are considered to be metadata for the ontology. This tool can reduce issues of recall and ambiguity; to improve recall, the keywords will be displayed and listed for the user in a hierarchy to choose from. Another benefit is that the tool will guide the user toward building only valid search queries. As a result, the user does not have to remember keywords, and all relevant query elements are derived automatically from the ontology. As for the ambiguity problem, the ontology-based tool and the faceted-based search both contribute to reducing the ambiguity. An ontology-based approach helps in reducing the ambiguity since the meaning of the displayed keyword is clear and there is no other meaning of a keyword, so the user will know what he/she is looking for. As for the ambiguity between the concepts within the displayed hierarchy, a faceted-based search is introduced to narrow down and personalize the search results and suite for the user.

This project is based on an existing application (the Manchester Pizza Finder) that has an ontology-driven interface. The application was built using the code of the Manchester Pizza Finder and adding some new modifications and functionalities. The Manchester Pizza Finder is a tool that displays a list of toppings based on pizza ontology. The user query for different pizzas depends on the included and excluded chosen toppings. This project takes this tool further by adding more functionalities and a number of enhancements, such as making it dynamically configured based on the ontology used, and implementing filters to be applied on the constructed query

and on the search results. The application has the same basic functionalities as the Manchester Pizza finder; it is called the Manchester Sushi Finder and is a tool to query for sushi based on included and excluded ingredients. Although the main ontology used is drawn from a sushi menu restaurant, does not mean only sushi ontology will work. On the contrary, a part of making the tool flexible is to allow it to work with different ontologies and domains.

DECLARATION

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

INTELLECTUAL PROPERTY STATEMENT

ACKNOWLEDGMENT

I would like to acknowledge all the people who helped me either directly or indirectly finish this dissertation. I would like to thank them for their patience, guidance, and support to complete what I consider to be my most important academic achievement. I would like to thank the following people:

- My supervisor, Sean Bechhofer, for his guidance and assistance during this project. I am sincerely thankful for all his invaluable support and contributions.

- My Company, Saudi Aramco, for giving me this opportunity by sponsoring my academic studies at one of the best universities in the world.

- My friends and colleagues Abdulrahman Hussein, Phalecs Jagboro, Ruvin Yusubov, Elchin Ismayilov, and Atheer Algherairy for their encouragement and help in evaluating my project.

- My parents for supporting me and believing in me during my studies.

- My wife, Duaa, for her continuous support and patience, and my two sons, Ahmed and Houd, for they are the light that kept me going forward.

1 INTRODUCTION

1.1 Motivation

As the trend nowadays is to try to make machines more intelligent [1], sharing knowledge of information instead of sharing raw data on the web is becoming more desirable. Computers that have the ability to think, to understand, to share knowledge, and to simulate the reasoning process of humans seem to be an idea from an artificial intelligence (AI) fiction movie. The semantic web is helping to convert the current web of information into a web of knowledge. It is all about sharing knowledge, which is understandable for machines, on the web. Knowledge of a concept domain is being captured and represented according to our understanding within a file called ontology. Ontologies are considered the main pillar of the semantic web. Computers do not understand information stored on the web, such as Extensible Markup Language (XML) and Hypertext Markup Language (HTML). These are just codes to the machines, and they display it to users regardless of what knowledge needed. As a result, the sematic web emerged as a place for machines to reason over retrieved information that is knowledge associated. Ontologies are used to represent knowledge and make inferences from that knowledge using a machine's computational capabilities and some reasoning techniques, such as description logics.

An intelligent way of browsing and retrieving represented knowledge eases the process of retrieving knowledge to the user. There are a lot of intelligent browsers with ontology-driven user interfaces, such as Transparent Access to Multiple Bioinformatics Information Sources (TAMBIS) [2], Semantic Webs and AgentS in Integrated Economies (SEWASIE) [3], and the Manchester Pizza Finder [4]. Browsing and constructing queries through such user interfaces (UIs) is easy and saves time because the UI acts as an interactive manual. It eases the process of constructing the intended query since the process itself is guided be the UI. In addition, it saves the user time by displaying only what is the system intended to do. The user does not need to have previous knowledge about the domain due to the explicit display of the options needed to construct a query. Ideas like manuals and the help menu in the menu bar of a UI would seem absolute when compared to the self-guided UI. An additional technique to make the UI smart is to use faceted browsing, which allows the search to be personalized and returns more specific results by suggesting filters. Faceted browsing is intricately related to an ontology-driven UI since both provide some information about the query while it is being constructed [5].

Ontology-driven UI systems include TAMBIS and SEWASIE. TAMBIS is a system that gathers and analyses bioinformatics information from different sources using one interactive UI. SEWASIE is meant to access multiple sources of data and help the user throughout the construction of the precise query needed.

The idea of an ontology-driven UI is not new. This project will try to build an application on top of an existing tool (the Manchester Pizza Finder) with new functionalities and enhancements. The new tool is called the Manchester Sushi Finder, and it was built to use the sushi ontology that was previously developed by the Ontology Engineering course unit. However, the tool will also be able to use ontologies with a similar structure, concept domain, and configuration.

1.2 Aims

The aim of this project is to investigate and demonstrate the benefits of using OWL ontologies and the OWL API within an ontology-driven UI application as shown in the project page [6]. A configurable and flexible UI will be implemented to make the process of checking and testing ontology easier for students who have uploaded their ontologies. Most of the configurations will be stored in the ontology file, and the UI can browse other ontologies that contain some specific configurations, such as annotations. The application is called the Manchester Sushi Finder, and it allows a user to construct queries to search for sushi based on both included and excluded ingredients defined in the conceptual model represented in the ontology file. This tool is based on existing the Manchester Pizza Finder that is described in section 2.7.

1.3 Objectives

The aim of this project is divided into several objectives:

- Gather project requirements.
- Increase the reusability of the UI by making it configurable to suit the content of other conceptual models.

- Increase the usability of the UI by showing the languages available in the ontology with their percentages, and switch between them.
- Decrease the time needed to get the intended result by applying filters to the content of the conceptual model or/and to the results of the search query by introducing the notion of filters and facet searches to access more specific information.
- Increase the accuracy of the system so users can only construct valid queries to receive the intended results; the ontology-driven UI and use of faceted browsing will increase the accuracy of what needs to be queried.
- Provide a more flexible system by saving most of the configurations as annotations within the ontology itself.
- Represent the constant of the conceptual model with different views, such as a tree and lists. Provide users with more than one option to view the content of the model.

1.4 Contributions of this Project

This project was undertaken to enhance and add more functionalities to an exiting tool (the Manchester Pizza Finder), which allows users to browse pizza toppings and construct queries to obtain a certain type of pizza. While constructing a query, the user chooses pizza toppings to be included and excluded. As a result, pizza that matches the user's specified criteria will be shown in the resulting window. This tool can use only one static pizza ontology at a time.

The new tool (the Manchester Sushi Finder) can use ontologies with specific annotations in them as configurations. The new tool can browse the ingredients of any food domain associated with certain annotation properties. The tool offers the ability to upload different ontologies during the runtime. Filters are included in the ontology file as annotations and displayed in the tool if they exist. The tool has the ability to show facets if they were specified in the ontology to be applied on the result. Furthermore, it provides different views (tree view and list view) of the ingredient to ease the process of browsing. It can show languages if the ontology is labelled with different languages and would also show the percentage of the languages according to the ontology.

As part of an ontology engineering course unit to develop food domain ontology to demonstrate the use and benefit of OWL to students, this tool would ease the process of check ontologies and might help them to understand the concept faster. Students can see their ontologies running when using the tool and can clearly see where they are going. The tool uses annotations heavily so that the mentioned functionalities work. Finding some limitation in OWL annotation techniques may contribute to considering a fix in newer versions of OWL in the future.

1.5 Structure of the Dissertation

There are seven chapters in this dissertation. The first chapter contains an introduction regarding the project as well as the motivation for this project. Then the aim and the objectives are detailed. It ends with the contributions of the project.

Chapter two gives a brief background about the main key parts of this project. It discusses OWL, OWL API, and the disadvantages of a traditional way of searching and retrieving information. Then, it presents the technologies that will help overcome these disadvantages, such as an ontology-based user interface, faceted-based searches, and ontology visual querying, and finally, it gives a brief overview of an earlier version of the project.

In chapter three, the research methodology is discussed, and how the project will be executed is detailed at length. Then it states the project deliverables. Finally, the evaluation plan is presented along with the tools used for this project.

Chapter four discusses in detail the design process for the system. Then, it describes the user stories and scenarios. At the end, the design process of an ontology used with the tool is discussed.

In chapter five, the implementation process of the system is explained. It starts out by discussing some of the most important technologies used in the project. Then, it mentions which software model has been used. Next, it demonstrates the major functionalities of the project. Finally, it states some of the limitations that have been faced during the implementations phase.

Chapter six details the evaluation process along with a critical analysis. It discusses the online survey that was created to evaluate the project. It starts with the survey questions, and then the types of evaluators are discussed. At the end, a critical analysis has been preformed on the results of the survey.

Chapter seven concludes the dissertation by giving an overall overview of the project.

2 BACKGROUND

2.1 OWL

OWL is a semantic web language that represents things about the world, groups of things, and the relations between them [7]. It is a way to represent knowledge such that it is a representation of the world and our knowledge of it and it is accessible to programs and can be used by them [8]. It makes implicit knowledge explicit [7]. The intention behind creating OWL is for it to be used not only by humans but also by applications [9]. OWL can be accessed from machines because it is based on computational logic, which means that a machine using a special type of software can reason over OWL documents [7]. There are two versions of OWL: OWL and OWL 2 [7]. OWL has been a World Wide Web Consortium (W3C) recommendation since 2004, and then OWL 2 was published in 2009, followed by a second edition in 2012 [7, 10]. The OWL 2 is just an extension and revision of the original OWL that was published in 2004 [7]. The OWL has several defined syntaxes, including Functional Syntax, RDF/XML, OWL/XML, and the Manchester OWL Syntax [7, 10].

The information on the web was described by the OWL working group [9] as scattered; it might be meaningful for humans but not for machines. Therefore, the semantic web gives explicit meaning for this information. As a result, integrating and processing the information would be easier for machines.

In 2012, the OWL 2 was introduced by the OWL working group [11]. It is not substantially different than the OWL but instead could be seen as an extension of the OWL with some additional features. The OWL 2 has several syntaxes, including Manchester, functional, RDF/XML, OWL/XML, and Turtle syntaxes; most developers only need one syntax to use OWL. Figure 1 shows the structure of OWL 2 and syntaxes used with OWL.

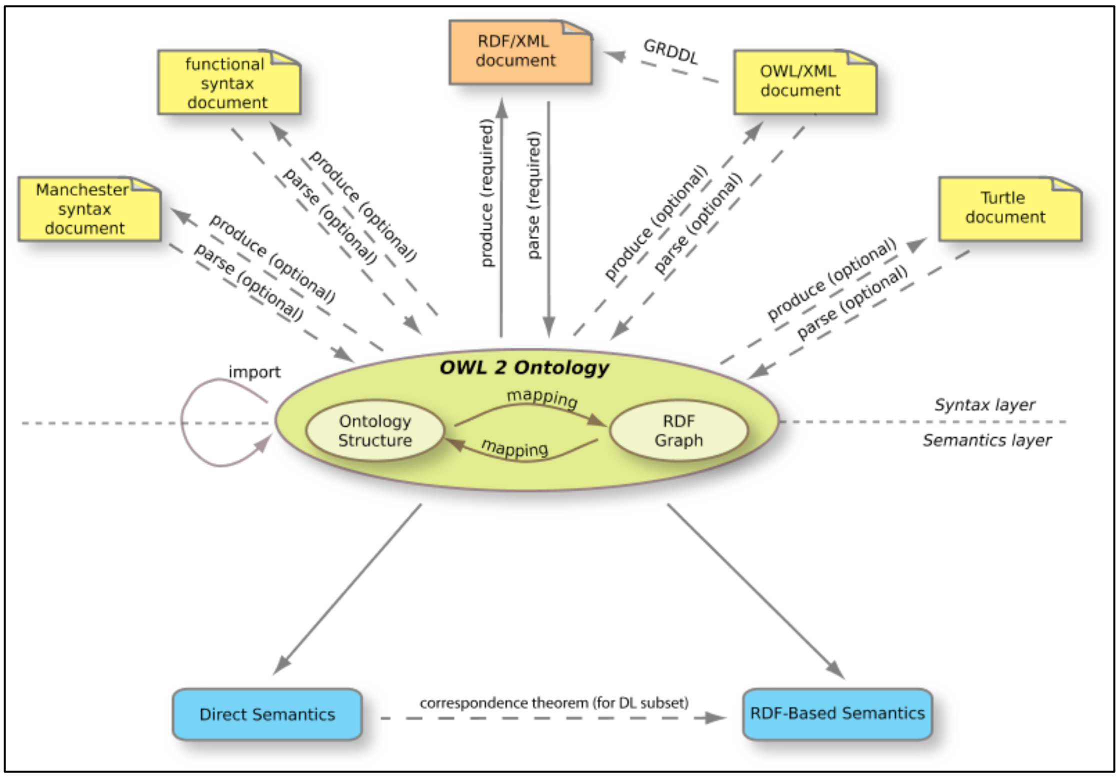


Figure 1: The structure of OWL 2 [11]

OWL files or documents are called ontologies, as described in [7]. The purpose of these ontologies is to make it easier for the machine to access information on the web and perform reasoning on them. These ontologies can be posted on the web or uploaded to a local computer depending on the need. One of the advantages of ontologies on the web is that they can be referenced from or have a reference to other ontologies, whereas those that are stored on a computer can only be used locally.

2.2 OWL API

Wikipedia [12] describes an Application Programming Interface (API) as a set of protocols that ensure the software components interact with each other in the right way, and it can take many forms in different areas. It is used on the web as a set of Hypertext Transfer Protocol (HTTP). APIs also see heavy use as libraries of programming languages, such as Java APIs. In object-oriented languages like Java, the API is a set of classes and methods that can be accessed and used. Basic examples would include using the inputting and outputting classes (e.g., the BufferedReader and BufferedWriter classes in Java). Since this project will be built using the Java programming language, the API used is a Java API called the OWL API. The OWL API is a set of classes and methods that facilitate access to Web Ontology Language

(OWL) ontologies. It creates objects that represent ontology objects and manages the interactivity between them and any other program.

The OWL API is described as an API with the purpose of specifying how to interact with OWL ontologies [10]. OWL ontologies can be created, manipulated, and reasoned over using OWL API. It has been available almost as long as OWL. The OWL API went through several revisions following its development, and it has the ability to parse and serialize OWL ontologies to different syntaxes, such as functional Syntax, RDF/XML, OWL/XML, and the Manchester OWL Syntax.

The OWL API takes the burden of parsing and serializing OWL ontologies off of the developer since it has already been taken care of during its implementation using Java [10]. The OWL API also comes with some capabilities, such as loading and saving ontologies.

OWL ontologies can also be accessed using the OWL API through the OntologyManager interface alone [10]. This interface manages all changes in ontology; Figure 2 shows a UML diagram of how ontologies would be managed using the OWL API. It makes use of Model–view–controller (MVC) software architectural pattern since all changes go through the OWL Ontology Manager.

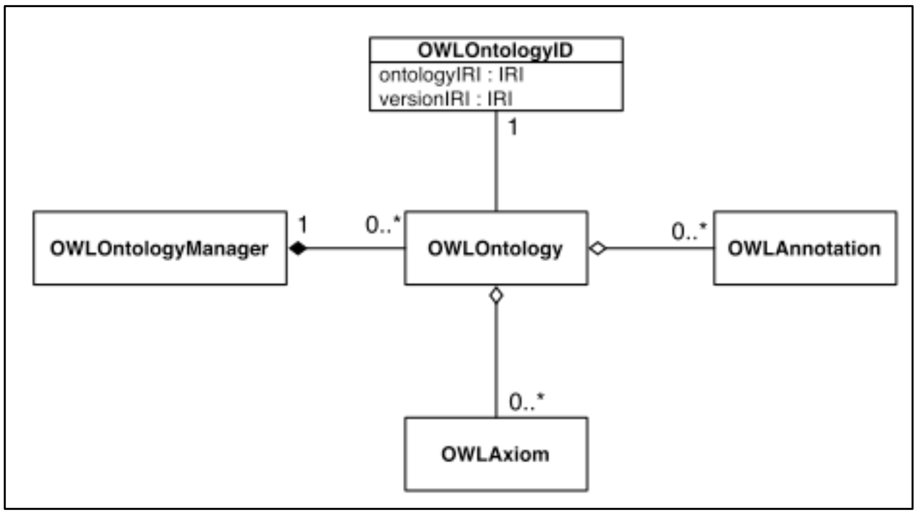


Figure 2: UML diagram showing the management of ontologies using the OWL API [10]

Inference is applied on the OWL ontologies using the OWLReasoner interface [10]. This interface provides some useful checks for consistency and verifying computation of class and axiom entailments. Since the reasoning functionality is separate, developers can either use the available reasoners or can provide their own implementation. There are some already existing implementations of reasoners, such as FaCT++, HermiT, and Pellet.

The OWL API does not offer much of a query mechanism [10]. However, it provides some sort of basic querying, which is based on entailment checking functionality.

2.3 Conventional Information Retrieval Method

Information retrieval is the method in which some information is collected from a source or multiple sources that contain the needed data. Every retrieval system requires some mechanisms to obtain the relevant and needed information. Nowadays, more information probably resides in the cloud than it has in recent years [13]. As the amount of data on the web continues to grow dramatically, the need to find and retrieve relevant information is becoming more important. Getting the desired results is becoming more problematic because of the amount of the data on the web and the techniques that are used to retrieve the information. Usually, in a search engine the results range from relevant and irrelevant. It would be nice for a user to search for something and have the most relevant query result for his/her needs to be returned.

There are many different methods of retrieving data from the web. A keyword-based search could be the oldest one, although other methods are also used.

The keyword-based search method uses specific words called “keywords” that are linked with database records [14]. This procedure is likely the default and the usual choice to use in searches since it has been used over a long period of time. This method seems easy to use, as it resembles natural language, which can be understood by humans but not by machines. Because of the human factor that exists in writing the search query, things inevitably could go wrong. Simplicity and ease come with a cost, and keyword-based search methods suffer from some serious issues. (1) One of these issues is the lack of accuracy and recall because of all of the synonyms and the homonyms, which are based on memorizing terms rather than concepts. (2) Another major issue is that using a keyword-based search can add more ambiguity, when the user just wants to browse around to find out what is there or if the user does not know the right term used in specific content.

According to [14], there are solutions for both issues. The lack of precision and recall can be treated using an ontology-based information-retrieval method. The growth in the ambiguity issue would be solved by using a multi-faceted search method, which would guide the user while constructing the search query. Therefore, the use of a knowledge base and a concept base would be more desirable than just providing arbitrary information. In addition, a sense of AI is also felt since machines can make inferences based on some rules.

In this project, the keyword-based method will not be used since it has limitations like recall and ambiguity. As I am trying to overcome these limitations, an ontology-driven user interface will be used as an alternative method.

2.4 Ontology-based User Interface

In general [15], developing user interfaces can be hindered by the knowledge of the user. Letting the user know what can be asked for and constructing a meaningful search query using the user interface is the major problem. Some solutions to remedy this issue could be making every option in the user interface available to reduce the recall issue. Another solution could be to write manuals for the user to follow; however, these solutions might provide more complexity and other problems. The former solution could overwhelm the user with all of the options available. The latter solution could increase the load on the user to study and spend time on something that needed to be recalled eventually. An ontology-driven user interface would be the most suited solution since ontologies are based on conceptual models rather than just terms [3, 15]. This conceptual model gives a map for the user to follow when constructing queries [15].

An ontology-based UI is defined [3, 15] as a user interface that allows the user to construct and manipulate queries based on a domain concept stored in the ontology. This domain concept drives the user interface, so there is no need for manuals or the storage of all available options in the UI since the ontology-based UI should act as a guide for the user. The process depends on recognizing knowledge instead of memorizing keywords. It allows the user to build complex and meaningful queries and return the needed results. In addition, it offers the user the option of browsing around to find what he/she needs. The user does not have to have any prior experiences with the underlying conceptual knowledge. TAMBS gives the illusion of retrieving from a single source while it reads from multiple sources and converts the selected options to appropriate query languages that match sources.

The user interface offers choices and some scenarios for the user to guide the user toward constructing meaningful queries that return the intended results [3, 15]. Users would not face a no-result status after running queries. Query expressions are Description Logics (DLs) expressions, and they are incremental and compositional [3, 15]. DL is a method for knowledge representation that is used by the conceptual model [16]. It provides a hierarchal model based on a conceptual model that represents classes of specific domains and the relationships between the instances of these classes [3, 15]. The DL model is not easy because of the need for knowledge about the DL syntax along with understanding it, so a friendly user interface needs to be included to prevent the user from dealing with DL [15].

According to [15], there are two kinds of concepts that the DL model supports: the concepts definitions and the assertions made on the concepts definitions, like the subsumption relationship between two classes. In a way, assertions on the original concepts are considered as defining new concepts definitions. Compositional concepts can be formed using some services provided by DL. Reasoning about the concept definitions is done through the following services provided by DL:

- **Satisfiability**: Make sure that the concepts are consistent.
- **Subsumption**: Create a composite concepts definition from assertions made on the original concepts definition.
- **Classification**: Develop a new classification hierarchy based on the subsumption relationship.
- **Retrieval**: Retrieve any individual who is part of the concept definition whether it is original or generated from the subsumption relationships.

Ontologies support creating annotation properties within themselves and associate them with entities [17]. These annotation properties consist of the name and the value. Annotations within ontologies play a major role in driving the user interface. Annotations would form some set of rules for the user interface to follow and base interactions upon. A tool was developed by [17] using animal ontology and annotations, which provide a good example of what annotations can do in terms of user interface interactivity. This tool allows the user to brows for ape photos based on some annotated criteria, such as the quality of the image and the environment where it was taken.

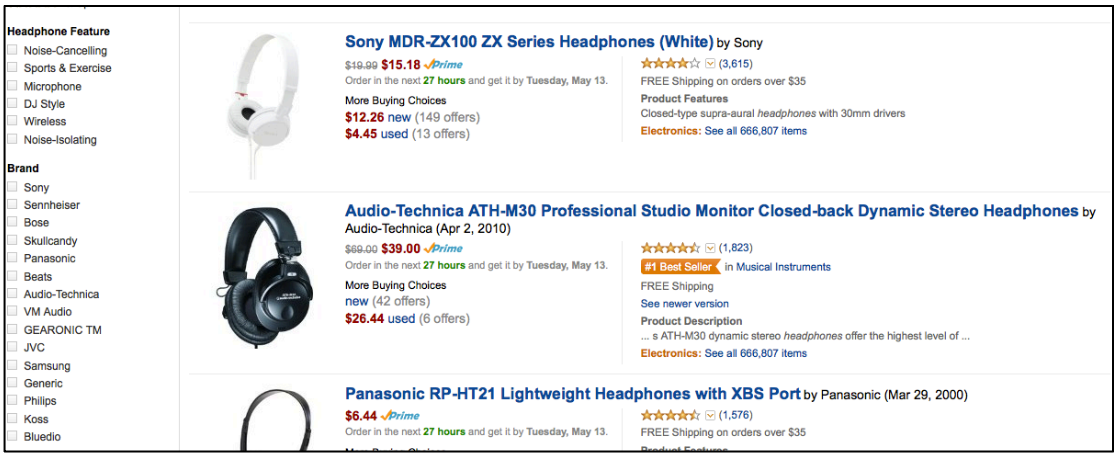
In this project, I am trying to develop an application to find out different information about sushi. The application is ontology-based. Annotations play a major role in driving the user interface to make it more flexible. This application is based on the Manchester Pizza Finder application, which will be elaborated on in a separate section, although this one also contains some enhancements.

2.5 Faceted Based Search

An ontology-based user interface only provides the taxonomy, conceptual hierarchy, and broad search capabilities. With the ontology conceptual hierarchy, users still can get broad search results. The transition from general to more specific results needs some kind of smart retrieval mechanism. Facet-based searching along with an ontology-based user interface would help the user construct valid search queries and personalize the search queries to suit the user's needs. Just as hiw ontology in user interface development to eliminate the recall element, using a faceted-based search eliminates the ambiguity constructing the query and produces the intended results. Therefore, ontology helps in returning relevant results but is also a faceted-based search assistant who takes the most relevant results and returns the most precise ones. Ontogator is a system that combines the two methods: the ontology-driven interface and a faceted based search [14]. The intent of Ontogator is to search for a particular image with specific annotations [14].

The ontology support faceted classification system provides a taxonomic order. Taxonomic order allows multiple ways of viewing results rather than pre-determined ones [14, 18]. Faceted search is based on the faceted classification system where information elements are dimensions called facets [18]. Faceted search is an intelligent and efficient retrieval mechanism that allows the users to filter collections of information based on some facets [18, 19]. Faceted search is also known as “faceted navigation” or “faceted browsing” [18]. Users can get more accurate and relevant results by applying filters (Facets) [18]. Facets here are derived from the ontology itself based on some annotations as metadata [14].

There is both faceted browsing and faceted searching. Faceted browsing is constructing search queries by selecting some provided filters (Facets) [14]. In faceted browsing, the user is given choices to select from to form the valid search query [14]. The query language is hidden from the user, so the burden of knowing the syntax is lifted. On other hand, faceted search is more about personalizing the search result to suit a specific need of the user. Faceted searches are heavily used on e-commerce websites like Amazon[1] or eBay[2]. Figure 3 shows the use of a faceted search on Amazon.com.



Headphone Feature
- ☐ Noise-Cancelling
- ☐ Sports & Exercise
- ☐ Microphone
- ☐ DJ Style
- ☐ Wireless
- ☐ Noise-Isolating

Brand
- ☐ Sony
- ☐ Sennheiser
- ☐ Bose
- ☐ Skullcandy
- ☐ Panasonic
- ☐ Beats
- ☐ Audio-Technica
- ☐ VM Audio
- ☐ GEARONIC TM
- ☐ JVC
- ☐ Samsung
- ☐ Generic
- ☐ Philips
- ☐ Koss
- ☐ Bluedio

Sony MDR-ZX100 ZX Series Headphones (White) by Sony
~~$19.99~~ $15.18 Prime
Order in the next 27 hours and get it by Tuesday, May 13.
More Buying Choices
$12.26 new (149 offers)
$4.45 used (13 offers)
★★★★☆ (3,615)
FREE Shipping on orders over $35
Product Features
Closed-type supra-aural *headphones* with 30mm drivers
Electronics: See all 666,807 items

Audio-Technica ATH-M30 Professional Studio Monitor Closed-back Dynamic Stereo Headphones by Audio-Technica (Apr 2, 2010)
~~$69.00~~ $39.00 Prime
Order in the next 27 hours and get it by Tuesday, May 13.
More Buying Choices
new (42 offers)
$26.44 used (6 offers)
★★★★☆ (1,823)
#1 Best Seller in Musical Instruments
FREE Shipping
See newer version
Product Description
... s ATH-M30 dynamic stereo *headphones* offer the highest level of ...
Electronics: See all 666,807 items

Panasonic RP-HT21 Lightweight Headphones with XBS Port by Panasonic (Mar 29, 2000)
$6.44 Prime
Order in the next 27 hours and get it by Tuesday, May 13.
★★★★☆ (1,576)
FREE Shipping on orders over $35

Figure 3: The use of a faceted search on an e-commerce website (Amazon)

Faceted searching could be applied in two ways: unidirectional or bidirectional. In a unidirectional way, it is either applied before constructing the search query on a collection of selection that is browsed to construct the query or to the results of a query so they can be further refined. In the bidirectional way, it is applied both before

[1] http://*www.amazon.com*
[2] http://www.ebay.com

and after running the search query. Both serve the same purpose, which is to personalize the search and make it easy to suit the user's needs.

Facets bring benefits to applications, including the following [14]:

- **Guidance**: Facets guide the user toward constructing valid search queries.
- **Transparency**: Facets provide the user idea with data regarding what is available and help users to browse the content.
- **Lucidity**: Facets help removing the ambiguity caused by synonymous and homonymous query terms.
- **Relevance**: Facets help with pre-computed partial results calculated to select choices.

2.6 Ontology Visual Querying

The idea of visual querying involves visually constructing a search query using a drag and drop method instead of the traditional way. The same idea can be applied to ontology-based interfaces and is more powerful because of the benefits of the ontology-based applications. The interface would guide the user to build interactive and meaningful queries using ontologies [3, 5]. In addition, another advantage derived from the benefits of ontology-based applications is constructing only exact queries [3, 5].

According to [5], visual querying is not new. It has been around almost since the beginning of textual query languages. Nearly all visual querying languages have two features in common: (1) a model to represent the stated query and (2) a way of constructing the query. Since visual querying languages were invented to query from a data structure, it is only natural for their evolution to follow the development of data structure [5, 20]. A simple example of visual querying would be in Microsoft Access where it uses visual querying in tables to get the result of the query from the database.

A major benefit from ontology visual querying is the ease of querying, since the user only must drag and drop what needs to be queried. The user does not have to remember or know the vocabulary since user can survey the domain [3, 5]. As a result, forming queries for naïve users becomes easier [5]. In ontologies, new concepts can be defined either directly, like defining class, or indirectly such as making an inference of something. Therefore, creating a query is the same as creating a new concept, such as the TAMBIS system and SEWASIE system [2, 3, 5]. Another advantage would be helping users not experienced with the system to create satisfiable queries according to the constraints [5].

2.7 The Manchester Pizza Finder

The Manchester Pizza Finder is an application that finds a specific type of pizza based on the user's topping choices. Users can include and exclude any topping, and based on their input, the results will satisfy the query. A DL reasoner is present in this application since it generates the filtering criteria (pizza topping) and their categories in the runtime. It also ensures that the constructed queries and results are consistent. Based on the choices made, the DL reasoner retrieves the results that fulfil the input query. This application shows the use of ontologies, the OWL API, and the power of building ontology-based interfaces, and faceted browsing.

The Pizza Finder is considered an ontology-based application since the ontology drives the interface and provides a conceptual hierarchy of a pizza domain. In this type of application, the user does not need to recall keywords, know a query language, on query for a specific pizza. The application itself guides the user toward building only valid queries with the ability of making complex meaningful ones. It has the ability to incrementally compose queries. Users can browse around to figure out what specific toppings are needed. The queries are based on the knowledge of the pizza domain, not on keywords.

Pizza Finder personalizes the query construction process by providing some filters (facets). Therefore, the results would suit the user's needs. The Pizza Finder uses a faceted-base in query building, so the user will be guided to construct only valid queries and does not have to recall what keyword to choose to search for something in the domain. Users have the option to query for a broad or specific pizza in the domain based on chosen facets. Figure 4 shows the use of facets in the Pizza Finder; user can query for vegetable topping pizzas or for more specific items in the vegetable topping category, such as tomatoes.

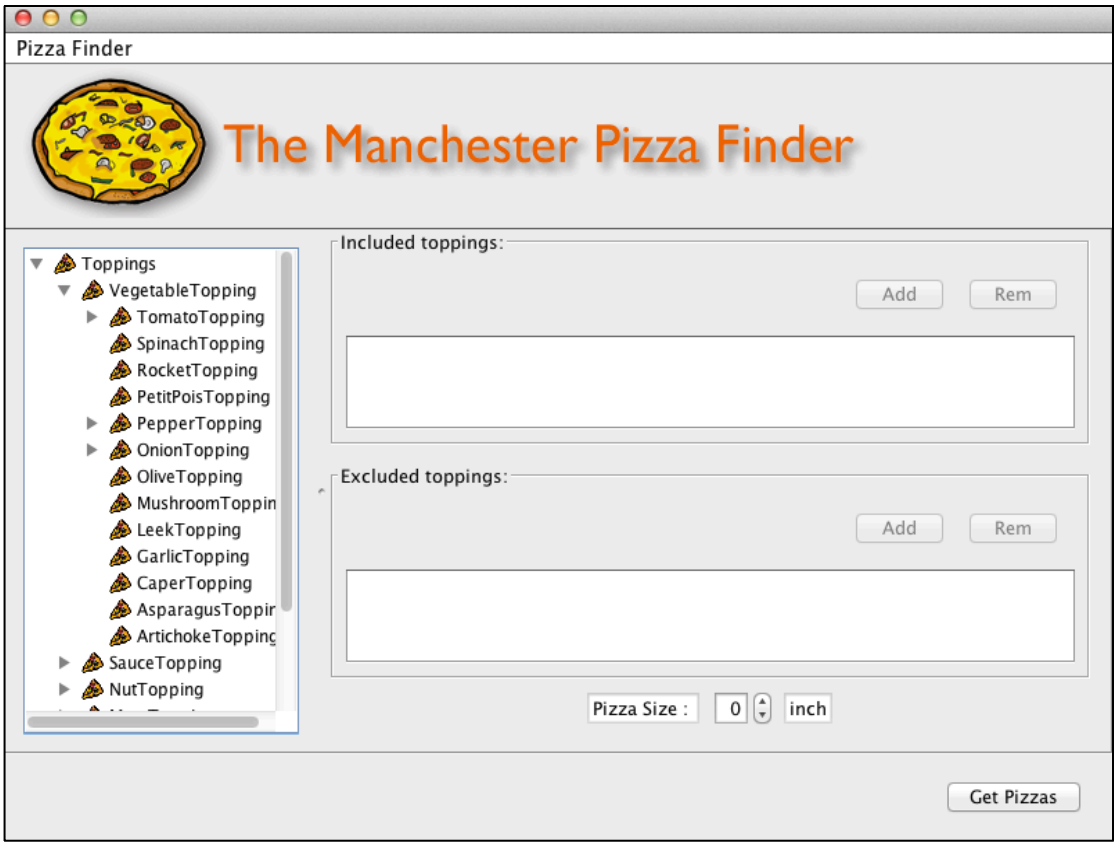


Figure 4: Facets in the left column are used to specify exact toppings [4].

The Sushi Finder is considered an extension of the Pizza Finder. Some enhancements of the Pizza Finder will be introduced. The Sushi Finder is more flexible than the pizza one. The application should be able to search for any given ontology regardless of its domain, but it should follow some standard annotations. Another enrichment will be the use of annotations to drive the user interface dynamically. Keeping the configurations within the ontology itself makes it easier for the user interface to be flexible. The entire application would be configurable in terms of the labels and languages being used. It is also configurable within itself, so there is no need for changing the configuration files. Once a facet search is introduced, the user can filter the options based on the configurations created in the ontology as annotations. In addition, users can apply filters to the query results.

2.8 Conclusion

In [4], the Manchester Pizza Finder was described as a user interface application that makes use of OWL ontology. It uses pre-defined pizza ontology that represents a domain concept of a pizza restaurant menu. For the application to be able to communicate with the pizza ontology, an API needs to be used. The OWL API is an important component as any other part of the application, if not more important. This OWL API manages all the communications between the application and the pizza ontology and is implemented using Java. Therefore, the Pizza Finder was developed using Java, which makes the communication between the application and the ontology easier. The OWL API has full access to the pizza ontology; it can perform operations on the ontology, such as making sure it is consistent.

3 RESEARCH METHODS

3.1 Research Methodology

As mentioned in the objectives section, the purpose of this study is to develop a system that will do the following:

1. Find specific sushi based on certain ingredient choices. Include and exclude criteria for the ingredients that are being used.
2. Allow students to run the system using an ontology they have developed for their coursework as long as they annotate their ontology in some way. The user interface is flexible since all labels can be configured.
3. Permit users to view the hierarchy of the ingredients in different views, like trees and lists.
4. Filter the hierarchy of the ingredients based on some facets as well as the results of the search query. The faceted search will allow the search feature to be more personalized instead of broad and general.

To achieve its objectives, this project has been divided into five stages.

3.1.1 Requirements Gathering Stage

Since the project is to develop a system that behaves in a certain way, I have started with the first and most important role in the software development process, which is requirements gathering. The project has three stakeholders; the end user who will use the system, the system provider who will provide the tool to the end user and probably apply configurations to the tool, and the ontology engineer. Since it was difficult to meet with all these stakeholders, I had to put myself in their shoes. Some of those stakeholders were involved in the requirements gathering process. The main stakeholder was my supervisor, as he requested for this system to be developed. Meetings were set up to discuss the requirements (what exactly should be done?). Some conversations were held with my fellow students, who also attended the ontology engineering for semantic web course, and we discussed whether they had used this system before, how it would help them, and what functionalities would be needed. All of those meetings and discussions provided more details and facilitated our understanding of some of the requirements. Others were grasped much later on in the process.

3.1.2 Background Study Stage

In the second stage, background research and a survey of the relevant literature was carried out along with an investigation of techniques to be used in the project. The review included journals, articles, publications, and existing systems with similar functionalities. A fair amount of time was spent reviewing different literature in an attempt to understand the varying aspects of the project's requirements. Reviewing the relevant literature helped by not only providing a wider knowledge of the problem domain but also in understanding the requirements of the project. As a result, I have a solid understanding of the project and the approaches that can be taken to handle such systems, and it helped me in dividing the project up into small tasks.

At this stage, a background investigation was conducted on OWL ontology, OWL API, ontology-based systems, faceted-based search systems, ontology visual querying, and the Manchester Pizza Finder, a system that I will build my project on.

3.1.3 Development Stage

In the development stage, I first decided to use Java to create my project. I began by reacquainting myself with Java, particularly its swing components, and the OWL API. Implementation started at this early stage. The strategy was to divide the development of the system into separate main functionalities and then combine them. Although some functionalities were developed separately from the application itself, this stage will officially begin after the second semester's exams. After the exams, I will check that the functionalities are working properly and then combine them.

The developments are considered enhancements of the Manchester Pizza Finder. They involve reading the configurations from the ontology file and acting accordingly for more flexibility. In addition, different views of the content of the conceptual model and the results of the search query will be considered. Adequate functionality to filter the content of the model based on some criteria saved in the ontology will also be added to customise the search even further for the user.

3.1.4 Testing Stage

At this stage, testing will be conducted on the application according to some scenarios that have been predefined. These scenarios are called users' stories, which will be further explained later on in the report. Since the strategy of the project involves developing functionalities alone then combining them, testing will be carried out during the development stage on the two functionalities separately and then on the final product after combining them.

3.1.5 Review and Submission Stage

After success in evaluating the product, the review and submission stage will start. An instruction file will be provided to guide the ontology developer in how to make the ontology work with the application. The application will act as a manual, so there will be no need for a guide for the system. The next step will be finishing the application, finalizing the dissertation, and then submitting them.

There are five milestones within this project that will guide me through the progress of the project. The milestones are as follows:

1. Initial report
2. Progress report
3. Application prototype
4. Application final product
5. Dissertation submission

The first milestone has already been completed; the initial report was submitted successfully in March. However, the second milestone was not turned in on time, and the original plan was altered. The new deadline was in June 6. The last three milestones will be worked on in parallel due to time restrictions. The final product is expected to be finished at the beginning of August. Finally, the dissertation will be submitted the first week of September.

3.2 Project Deliverables

At the end of this project, the benefits of using the semantic web within applications will be shown by building an application that demonstrates these benefits. The application will be ontology driven UI, being more specific certain annotation within the ontology will drive the UI. The tool will be flexible, as it will use ontology with specific annotations. In addition, it will be configurable, allowing users to query for specific sushi based on certain ingredients. There are three main deliverables of this project:

- User stories (scenarios) and acceptance tests
- Final version of the application
- Evaluation of the project

3.3 Project Evaluation Plan

In order to be able to assess the whole project, I need some qualities to evaluate the project against. Some qualities have been recognized to evaluate how good the system is and, more importantly, to measure whether the project can be considered a success by examining the success of the objectives. These qualities are as follows:

- **Were the deliverables met?**

  A most likely way to measure if the deliverables were met or not according to a timeframe is to check if a deliverable was completed within its allocated time.
- **Is the system accessible?**

  We can think of the system in two ways: as a project for an MSc program that should be accessible for students and lecturers who teach ontology engineering for semantic web, and as a real-world application (restaurant menu). Based on this distinction, users who will access the system will differ. In the first case, the application is a desktop application and is implemented using Java, which will assure that the students and lecturers can access the system easily. In the second case, end users can access the system, but a web application would be more reasonable. For now, it is only a desktop application.
- **Is the system (re)usable?**

  Regardless of whether they are students or real-world users, the end users can determine the usability of the system. There are two concepts here that determine the usability and reusability of the system. Therefore, both of them need to be checked as part of the project evaluation.
  - **Usability**: As an end user, one can ask several questions that will assure the system is usable. Some of these questions are as follows:

  1. How easy is it to use the system?

     The idea behind using an ontology-based user interface is to guide the user in how to use it and ease that process. Therefore, the system should be easy to use.

2. How much time must be spent to figure out the system?
   This question can be answered after submitting the project.
3. Are experts required to use the system?
   The system is designed for students who enrol in ontology engineering for the semantic web. Therefore, some degree of expertise is needed to configure the ontology to work with the application.
4. How easy is it to administer the system?
   Since most of the configurations are saved within the ontology, administering the system is already has been taking care of.

- **Reusability**: The system is reusable in that it can use different ontologies. This is another goal of the project: making it more flexible. The important question here is how easy it is to reuse the system (using different ontologies)?

- **Is the system easily configurable?**
  As most of the configurations are saved in the ontology file, there nothing to configure in the UI except labels and buttons. An ontology developer is the only stakeholder who will have to deal with the configurations, which are annotations in the ontology file. They are easy to write, as the ontology developer needs to follow the instructions that are provided with the application. The user interface should be configured automatically using annotations in the OWL file.
- **Is the result of the search query specific to user?**
  Stakeholders, such as students or real-world users (like restaurant customers), will have direct contact with the feature. This question can be answered after submitting and using the application. Nevertheless, applying a set of filters on the search result will narrow the search for the user since the system uses a faceted search method.

These are some questions that will help to assess the project. Most of them cannot be answered until the stakeholders have used the application. Therefore, their evaluation will help in a second version of the application or in the final product if it is considered a prototype.

3.4 Project Tools

Since the main part of the project is development, a programming language needs to be chosen. The OWL API, which was implemented in Java, is used in the project to manage the interactivity between OWL ontology and the application. Therefore, choosing Java as programming language makes sense. The project is being developed using NetBeans 7.4.

3.5 Conclusion

In Chapter Three, the details of the research methods are demonstrated. It begins with the methods used to plan the project. Then, it states the project deliverables. Next, it details the evaluation plan of the project. Finally, it shows the tools used to complete this project.

4 SYSTEM DESIGN

This chapter provides details about the design process of the tool (the user interface) as well as the user stories mentioned in 3.2, and the design process of an example ontology that has been used to test the tool. It starts with a detailed design of the user interface. Then, it moves to the design process of the user stories and how they have been used to specify system functionalities. Finally, it discusses the design of the sushi ontology, which is based on a sushi menu.

4.1 User Interface Design

This section details the design process of the user interface (UI) tool. It also shows the interaction design with the ontology like the one mentioned in 4.3. The main ideas behind the design are developing a UI that is driven by ontology, has the ability to use different ontologies, uses different languages in the ontology, employs filters on the browsed content, and includes facets in the search results. To provide further details, this section starts with the idea of building an ontology-driven UI. Then it moves to the second point: the flexibility of using different ontologies with the UI. After that, the use of different languages available in the ontology realm is discussed. In addition, this section elaborates on applying filters to the displayed content and applying facets to the search results. Finally, it sheds light on some UI enhancements like importing ontologies during runtime or having different views of the browsed contents.

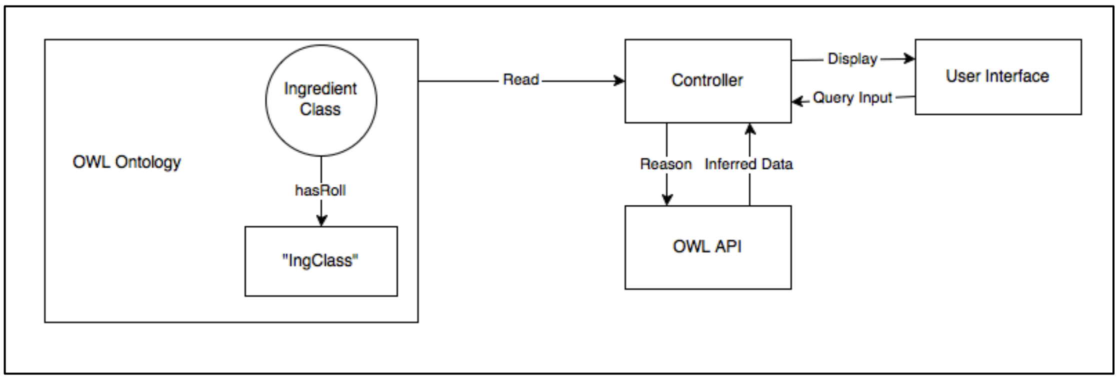


Figure 5: Ontology-driven UI diagram

Figure 5 demonstrates how the UI is driven by ontology. It shows three important parts: the ontology where it represents a food domain concept, the OWL API that links the ontology with the UI and preform operations on the ontology programmatically, and the tool (UI) that displays content, results, and control functionalities. The content (knowledge of the concept domain) is preserved in the ontology file. Ontology could have many classes, like those mentioned in 4.3. The ingredient class is an important one since it is displayed for the user and it constructs the search query. To open the ontology programmatically, its location is needed like any other file that is opened via code. The user is asked for the ontology location as mentioned in Table 18 in the Appendix. After arriving at the ontology location and opening it, the Ingredient class is retrieved by using a combination of two actions: annotating the Ingredient class using the hasRole annotation property with the value "IngClass," as mentioned in Table 19, and using the OWL API to retrieve the class with the annotation specified. An algorithm is implemented to search for the class that uses the "hasRole" annotation property and the value "IngClass" by using the OWL API to access the ontology, ensure it is consistent, and get the Ingredient class. The Ingredient class gets rendered using a tree and a list OWL API and is displayed so that the user can browse it and select from it to construct a search query. As a result, the user will be guided through the construction process of the search query and experience a reduced amount of recall for the keywords as well as the clarity for constructing a meaningful search query.

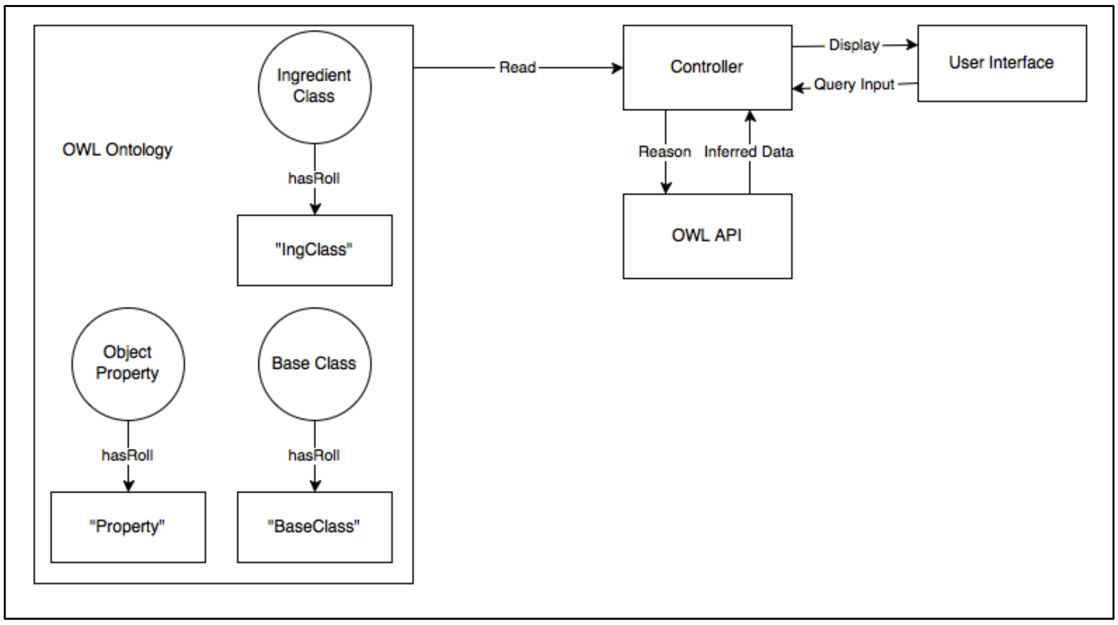


Figure 6: Three main methods to use different ontologies

While Figure 5 shows how to display the ingredient class to the user by annotating it using the hasRole annotation property, Figure 6 illustrates the idea of using different ontologies using the UI. The UI needs more information than just an ingredient class to perform queries and get results. As mentioned in Table 19 and as shown in Figure 6, three components are required to execute queries: the base class, the ingredient class, and the object property that links the previous two classes. An example of a DL search query would be as follows: [BaseClass] and ([Property] some [IngClass]), where BaseClass is the named entity that we want to query for, IngClass is the ingredients of BaseClass, and Property is the like between the two. To make ontologies work with the UI, the base class, ingredient class, and the property will need to be annotated with the hasRole annotation property. The value of hasRole for the base class is “BaseClass,” ingredient class is “IngClass,” and property is “Property.” The tool searches for the classes and object properties, which are annotated using hasRole and have the values “BaseClass,” “IngClass,” and “Property,” using the OWL API. Then, the UI displays the ingredient class to the user to select from the included and excluded ingredients. Then, the tool form different queries depend upon the selected included and excluded ingredients. These queries would be as follows:

Included Ingredients:

[BaseClass] and ([Property] some [selected Ing_1])

[BaseClass] and ([Property] some [selected Ing_2])

…

[BaseClass] and ([Property] some [selected Ing_n])

Excluded Ingredients:

[BaseClass] and not ([Property] some [selected Ing_1])

[BaseClass] and not ([Property] some [selected Ing_2])

…

[BaseClass] and not ([Property] some [selected Ing_n])

As the last part of this process, the intersection of the results of the included and excluded queries is taken. Finally, the result is displayed to the user.

As stated in Table 21, displaying languages depend on the ontology itself and whether it is labelled with more than language or not. As an assumption, the default language is English. The tool looks for labels with a language associated with them and calculates the percentage of the language to the whole ontology. The UI displays available languages in a dropdown menu to the user. After choosing a language, the ingredients and the results get rendered with the new language and displayed to the user. In case there is no language associated with the class labels, the name of the ontology class is displayed.

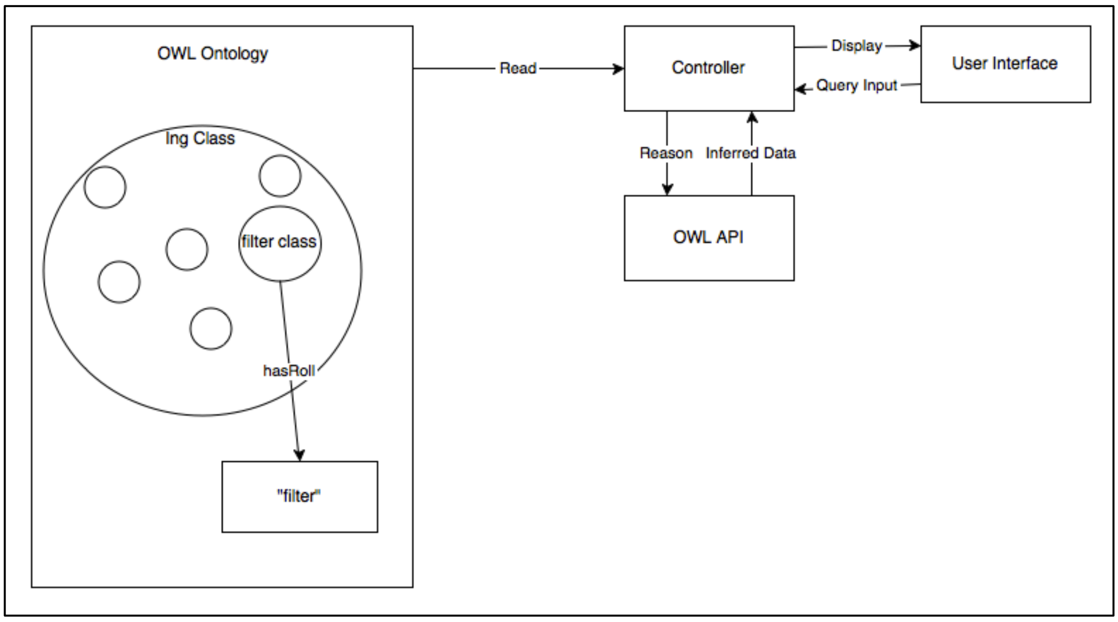


Figure 7: Filter annotation

Figure 7 shows the design of adding filters to the tree view within the tool. As stated in Table 22, if the ontology contains filter annotations associated with ingredient class, then the filter will appear in the UI; otherwise, it will not be present. Filtering using the tree's view depends on the ontology annotations. The hasRole annotation is used here as well, where an ingredient class is annotated with the value "filter." Filters usually are class expressions like any class with spicy ingredients. After the annotated ingredient classes, the tool looks for classes with the hasRole annotation property and the value of "filter." Then, the tool shows the specified filters to the users. Finally, upon choosing a filter, the ingredients are rendered. The ingredient classes that match the filtering criteria are highlighted with a yellow colour, and while the rest are disabled and cannot be selected. As mentioned in Table 23, filtering in the list view is completely different than it is in the tree list. Filtering in the list view is text-based filtering, where the user inputs text in the input text field. Then, the tool filters the list according to the input text. Any class that contains the input text is displayed to the user, whereas the others are hidden from the user.

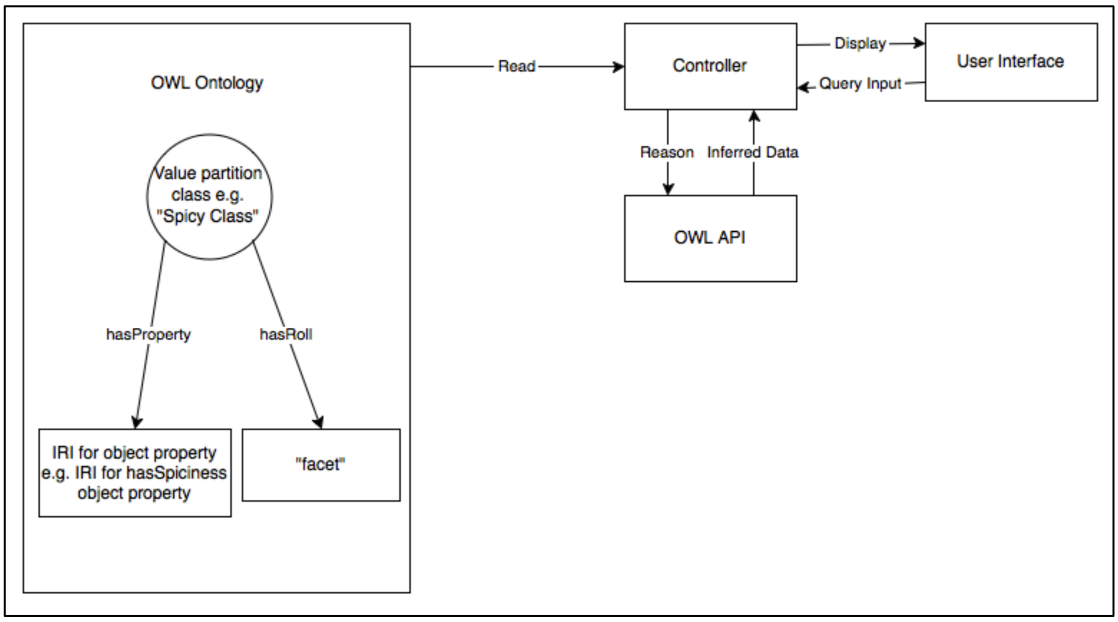


Figure 8: Facet design in the ontology

Facets require more work in the annotations than filters, since the property that link the ingredient class and the facet class needs to be identified. As Figure 8 shows, the value partition class, such as the Spicy class needs to be annotated using two annotation properties. Firstly, we use the hasRole annotation property with the constant value “facet” to identify that this class is being used as a facet. Secondly, we use the hasProperty annotation property with the IRI value of the object property to link the value partition class and an ingredient class. The tool then searches for classes with the hasRole annotation property with a constant value of “facet.” After finding the facet class, the tool gets the object property used to link ingredients classes with the facet. Finally, when the user queries for something and obtains the result, the UI shows facets that are specified in the ontology; the result changes based on the selected facet.

4.2 User Stories

This section demonstrates the functionalities of the tool in detail. The user story approach is used to determine the functionalities. User stories act as a guide to specify needed functionalities in a clear way. All the user stories follow the idea of the “3 C’s”: the card, the conversation, and the confirmation. The card is the general idea of the functionality; the conversation is the role of different stakeholders and what is require from them, and the confirmation is the details of the functionality. The user stories include the details regarding the following:

- Accessing the tool

- Running the tool for the first time
- Later runs of the tool
- Having a different view of the tool
- Having different languages in the tool
- Filters in the tree view
- Filter in the list view
- Querying in the tool
- Having facets to narrow down the search result even further

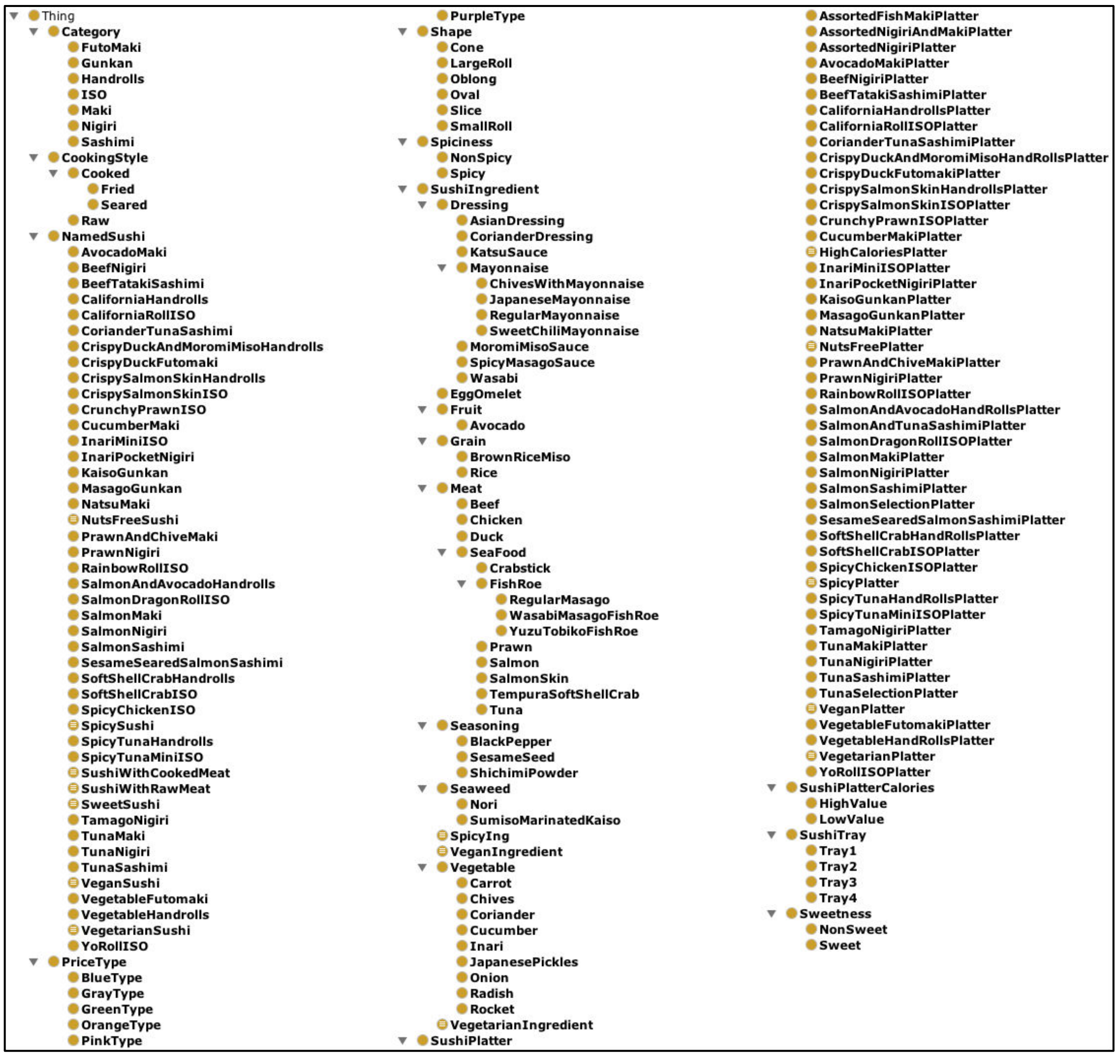
These user stories are demonstrated in the APPENDIX.

4.3 Sushi Ontology Design

In this section, an example of the design of the ontology to make it work with the tool is given. This section shows the class hierarchy, object properties, and annotation properties. Then, it explains the class definitions used. Finally, it provides some examples of DL queries that are applied to the ontology. I have chosen the sushi ontology, since it was the main ontology used to test the tool during implementation. The following sections detail an example of ontology. That does not mean the tool use only one ontology.

This ontology was developed using Protégé.

4.3.1 Class Hierarchy



```
Thing
  Category
    FutoMaki
    Gunkan
    Handrolls
    ISO
    Maki
    Nigiri
    Sashimi
  CookingStyle
    Cooked
      Fried
      Seared
    Raw
  NamedSushi
    AvocadoMaki
    BeefNigiri
    BeefTatakiSashimi
    CaliforniaHandrolls
    CaliforniaRollISO
    CorianderTunaSashimi
    CrispyDuckAndMoromiMisoHandrolls
    CrispyDuckFutomaki
    CrispySalmonSkinHandrolls
    CrispySalmonSkinISO
    CrunchyPrawnISO
    CucumberMaki
    InariMiniISO
    InariPocketNigiri
    KaisoGunkan
    MasagoGunkan
    NatsuMaki
    NutsFreeSushi
    PrawnAndChiveMaki
    PrawnNigiri
    RainbowRollISO
    SalmonAndAvocadoHandrolls
    SalmonDragonRollISO
    SalmonMaki
    SalmonNigiri
    SalmonSashimi
    SesameSearedSalmonSashimi
    SoftShellCrabHandrolls
    SoftShellCrabISO
    SpicyChickenISO
    SpicySushi
    SpicyTunaHandrolls
    SpicyTunaMiniISO
    SushiWithCookedMeat
    SushiWithRawMeat
    SweetSushi
    TamagoNigiri
    TunaMaki
    TunaNigiri
    TunaSashimi
    VeganSushi
    VegetableFutomaki
    VegetableHandrolls
    VegetarianSushi
    YoRollISO
  PriceType
    BlueType
    GrayType
    GreenType
    OrangeType
    PinkType
    PurpleType
  Shape
    Cone
    LargeRoll
    Oblong
    Oval
    Slice
    SmallRoll
  Spiciness
    NonSpicy
    Spicy
  SushiIngredient
    Dressing
      AsianDressing
      CorianderDressing
      KatsuSauce
      Mayonnaise
        ChivesWithMayonnaise
        JapaneseMayonnaise
        RegularMayonnaise
        SweetChiliMayonnaise
      MoromiMisoSauce
      SpicyMasagoSauce
      Wasabi
    EggOmelet
    Fruit
      Avocado
    Grain
      BrownRiceMiso
      Rice
    Meat
      Beef
      Chicken
      Duck
      SeaFood
        Crabstick
        FishRoe
          RegularMasago
          WasabiMasagoFishRoe
          YuzuTobikoFishRoe
        Prawn
        Salmon
        SalmonSkin
        TempuraSoftShellCrab
        Tuna
    Seasoning
      BlackPepper
      SesameSeed
      ShichimiPowder
    Seaweed
      Nori
      SumisoMarinatedKaiso
    SpicyIng
    VeganIngredient
    Vegetable
      Carrot
      Chives
      Coriander
      Cucumber
      Inari
      JapanesePickles
      Onion
      Radish
      Rocket
    VegetarianIngredient
  SushiPlatter
    AssortedFishMakiPlatter
    AssortedNigiriAndMakiPlatter
    AssortedNigiriPlatter
    AvocadoMakiPlatter
    BeefNigiriPlatter
    BeefTatakiSashimiPlatter
    CaliforniaHandrollsPlatter
    CaliforniaRollISOPlatter
    CorianderTunaSashimiPlatter
    CrispyDuckAndMoromiMisoHandRollsPlatter
    CrispyDuckFutomakiPlatter
    CrispySalmonSkinHandrollsPlatter
    CrispySalmonSkinISOPlatter
    CrunchyPrawnISOPlatter
    CucumberMakiPlatter
    HighCaloriesPlatter
    InariMiniISOPlatter
    InariPocketNigiriPlatter
    KaisoGunkanPlatter
    MasagoGunkanPlatter
    NatsuMakiPlatter
    NutsFreePlatter
    PrawnAndChiveMakiPlatter
    PrawnNigiriPlatter
    RainbowRollISOPlatter
    SalmonAndAvocadoHandRollsPlatter
    SalmonAndTunaSashimiPlatter
    SalmonDragonRollISOPlatter
    SalmonMakiPlatter
    SalmonNigiriPlatter
    SalmonSashimiPlatter
    SalmonSelectionPlatter
    SesameSearedSalmonSashimiPlatter
    SoftShellCrabHandRollsPlatter
    SoftShellCrabISOPlatter
    SpicyChickenISOPlatter
    SpicyPlatter
    SpicyTunaHandRollsPlatter
    SpicyTunaMiniISOPlatter
    TamagoNigiriPlatter
    TunaMakiPlatter
    TunaNigiriPlatter
    TunaSashimiPlatter
    TunaSelectionPlatter
    VeganPlatter
    VegetableFutomakiPlatter
    VegetableHandRollsPlatter
    VegetarianPlatter
    YoRollISOPlatter
  SushiPlatterCalories
    HighValue
    LowValue
  SushiTray
    Tray1
    Tray2
    Tray3
    Tray4
  Sweetness
    NonSweet
    Sweet
```

Figure 9: Class hierarchy of the sushi ontology

Figure 9 demonstrates the classes' hierarchy of the sushi ontology. The Thing class is the default super class of all the other classes. The subclasses of the Thing class describe the sushi concept domain. Notice that there are two main classes, NamedSushi and SushiIngredient, under the generic class Thing. The NamedSushi class describes different names of sushi, such as AvocadoMaki and BeefNigiri, while the SushiIngredient class describes the different ingredients of sushi.

The SushiIngredient class categorises the ingredients of sushi from general to specific. Notice the Meat class is part of the General class, while the Beef and Duck classes demonstrate specific concepts. The Seafood class further specifies the ingredients to describe the seafood concepts domain. Notice also that the equivalent ingredients classes are equivalent to some class expressions, like vegetarianIngredients, vegenIngredients, and SpicyIngredients. Any class under sushiIngredient can be used as a filter.

Moreover, this figure also shows the use of value partition patterns such as Spiciness, CookingStyle, Shape, and Sweetness classes. Spiciness class partition the spiciness into spicy and nonSpicy, and Sweetness into sweet and nonSweet. These value partitions can be used to specify facets in order to narrow down the result of a search query. For example, the spicy class could be used to specify that I want only spicy sushi.

4.3.2 Properties

There are three types of OWL properties that are being used here. These properties are object properties, data properties, and annotation properties.

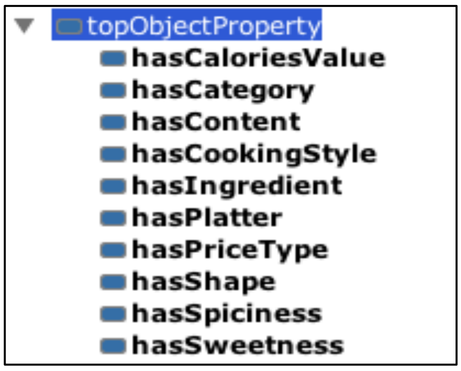

topObjectProperty
hasCaloriesValue
hasCategory
hasContent
hasCookingStyle
hasIngredient
hasPlatter
hasPriceType
hasShape
hasSpiciness
hasSweetness

Figure 10: Object properties of the sushi ontology

Figure 10 illustrates the object properties used within the ontology. Notice that a default generic object property exists here also. These object properties represent the relations between the classes. The hasIngredient object property plays an important role since it connects the NamedSushi class with the SushiIngredients class. The domain of hasIngredients NamedSushi and the range is SushiIngredients. Any specified sushi query should use the hasIngredients property in constructing it.

The properties hasSpicness and hasSweetness present a test of sushi ingredients. They also represent the relations of SushiIngredients and whether they are spicy, nonSpicy, sweet, or nonSweet. While these object properties show the relations between the sushi ingredients and some their characteristics, other properties represent different thing like hasShape, which illustrates the relation between a shape and a NamedSushi.

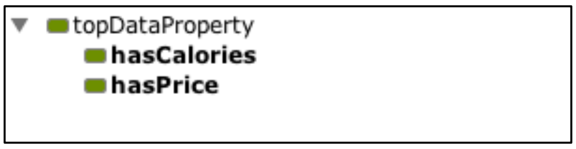


Figure 11: Data properties of the sushi ontology

Figure 11 illustrates the data properties used in the ontology. A handful of properties were selected: hasCalories, which links a specific sushi platter and its caloric value. On the other hand, hasPrice links the specific sushi platter with its price value. The caloric value and price value both are static value entered by the ontology developer.

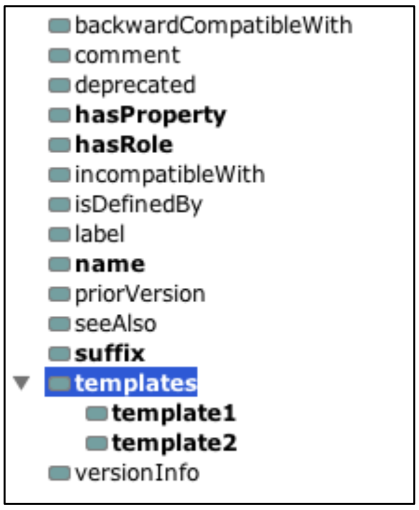
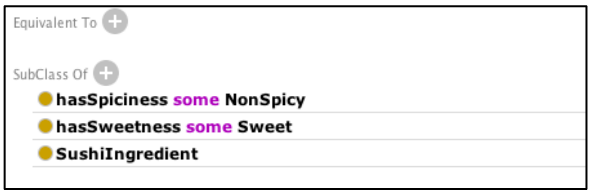


Figure 12: Annotation properties of the sushi ontology

Figure 12 shows the annotation properties used to attached metadata to different part of the ontology. Annotation properties are important to this project, since they are being used in specifying filters, facets, and even in driving the UI. The user can define new annotation properties, such as hasRole or hasProperty. The hasRole property is used to identify the role played by a certain class, such as a role of a filter, facet, or ingredient class. The suffix property is used to state the suffix used in the class hierarchy if any.

4.3.3 Expressing Class Definitions

This subsection illustrates the semantics used within the sushi ontology. It starts with normal class definitions used to represent the relations between classes. Then it moves to demonstrating the semantics used to drive the UI, to specify filter, and to install facet. The class definitions of class hierarchy are similar to each other, and the space here is limited. Therefore, the most important class definitions are illustrated.



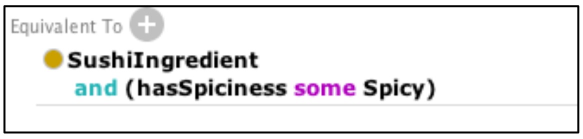
Equivalent To

SubClass Of

```
hasSpiciness some NonSpicy
hasSweetness some Sweet
SushiIngredient
```

Figure 13: Class expression of the EggOmlete class

Figure 13 shows the semantics of EggOmlete under the SushiIngredient class. This semantics reports the taste of EggOmlete is NonSpicy and it is Sweet by using the object property hasSpiciness and hasSweetness.

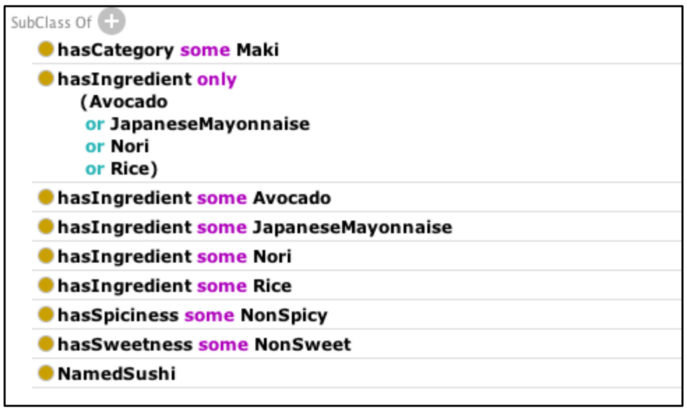


Equivalent To

```
SushiIngredient
 and (hasSpiciness some Spicy)
```

Figure 14: Class expression of the SpicyIng class

Figure 14 states the semantics of the SpicyIng class. The meaning of these semantics is that when you have any SushiIngredinet that is Spicy, the sweet characteristic could also be added using the hasSweetness property and the Sweet class to indicate Sweet class.



SubClass Of

```
hasCategory some Maki
hasIngredient only
    (Avocado
     or JapaneseMayonnaise
     or Nori
     or Rice)
hasIngredient some Avocado
hasIngredient some JapaneseMayonnaise
hasIngredient some Nori
hasIngredient some Rice
hasSpiciness some NonSpicy
hasSweetness some NonSweet
NamedSushi
```

Figure 15: Class expression of AvocadoMaki class

Figure 15 illustrates the semantics of a NamedSushi class AvocadoMaki. The first one specifies a suitable category for AvocadoMaki: Maki Category. The hasIngredient property is used to specify the ingredients of AvocadoMaki. Notice it uses some substances to include the ingredients and simply states that only these ingredients and nothing else is included. The rest of SushiIngredients's subclasses are similar in this way.

The most important part of semantics is using annotation properties to drive the UI as well as specifying filters, facets, and the object properties used with them. According to Figure 16, three things need to be considered for the tool to using different ontologies: BaseClass, IngredientClass, and the Property were used to represent the relation between BaseClass and IngredientClass. Figures Figure 17, Figure 18, and Figure 19 illustrate the semantics in detail.

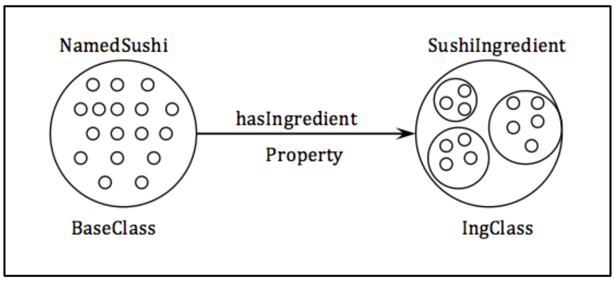
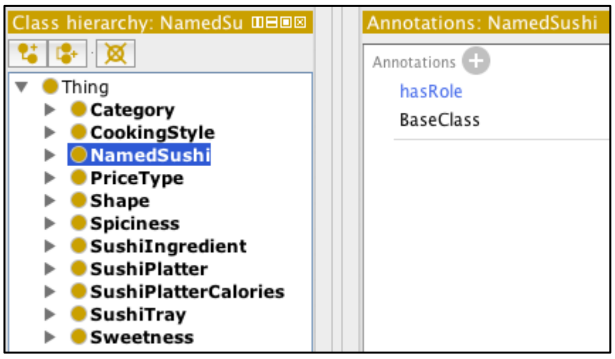


Figure 16: Basic annotations diagram of sushi ontology

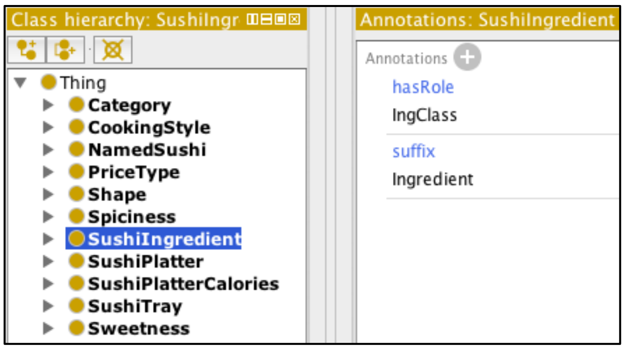


Thing
- Category
- CookingStyle
- NamedSushi
- PriceType
- Shape
- Spiciness
- SushiIngredient
- SushiPlatter
- SushiPlatterCalories
- SushiTray
- Sweetness

hasRole
BaseClass

Figure 17: The hasRole annotation used in the NamedSushi class

Figure 17 shows that the NamedSushi class uses the hasRole annotation property to indicate this is the BaseClass by using the constant value “BaseClass.”



Thing
- Category
- CookingStyle
- NamedSushi
- PriceType
- Shape
- Spiciness
- SushiIngredient
- SushiPlatter
- SushiPlatterCalories
- SushiTray
- Sweetness

hasRole
IngClass

suffix
Ingredient

Figure 18: The hasRole annotation used in the SushiIngredient class

The hasRole annotation property is being used in different locations within the ontology. As seen in Figure 18, the SushiIngredient Class is annotated using the hasRole annotation property with the constant value "IngClass" to indicate that the SushiIngredient class has the role of the Ingredient class.

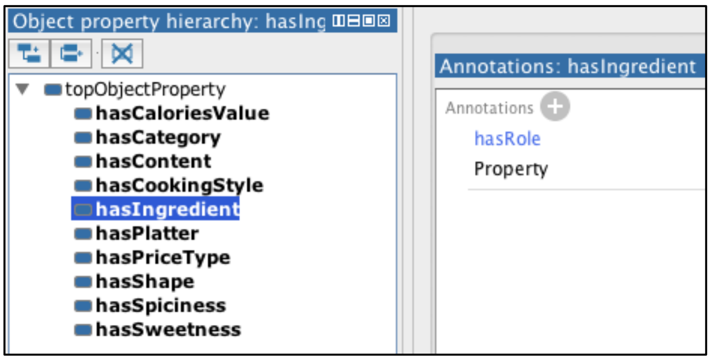


Figure 19: The use of a hasRole annotation property in a hasIngredient object property

Figure 19 shows the use of the hasRole annotation property in the hasIngredient object property. It means that the hasIngredient object property plays the role of the property that describes the relation the ingredient class and the base class.

hasRole is also used in determining filters and facets. Filters need only to be indicated by the hasRole annotations property with a constant value of "filter" for the tool to know it is a filter. Figure 20 shows that the VeganIngredient class is a filter:

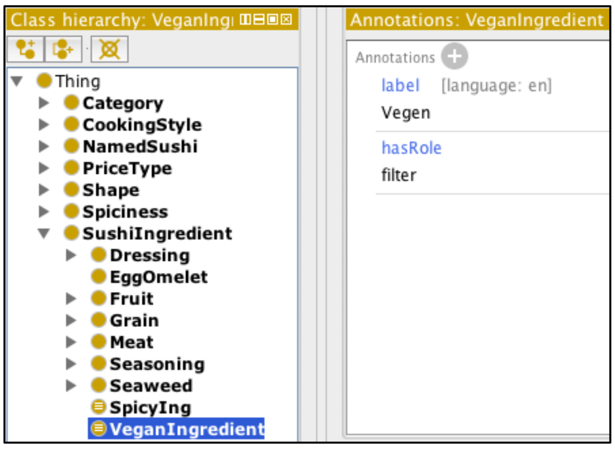
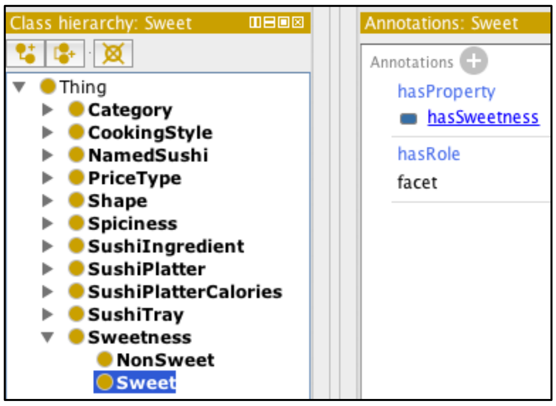


Figure 20: hasRole annotation property used to determine VeganIngredient as a filter

Defining facets requires more than using the hasRole annotation property, since there is a need to be an object property that describes the relation between the ingredient sushi class and then its characteristic. For example, the sweet ingredient and the sweet class shown in Figure 9 needs to have some kind of link between them. Figure 21 shows the use of two annotation properties hasRole to define the role of Sweet class as a facet and hasProperty to specify the object property used to liken the sweet class with ingredient classes. Notice here that the value of the annotation property hasProperty is IRI of hasSweetness object property shown in Figure 10.


Class hierarchy: Sweet

- Thing
  - Category
  - CookingStyle
  - NamedSushi
  - PriceType
  - Shape
  - Spiciness
  - SushiIngredient
  - SushiPlatter
  - SushiPlatterCalories
  - SushiTray
  - Sweetness
    - NonSweet
    - Sweet

Annotations: Sweet

Annotations

hasProperty
hasSweetness

hasRole
facet

Figure 21: Facets are determined in the ontology

4.4 Conclusion

This chapter presented the design process of the important aspects of the project. It started with the design of the UI based on the user stories in section 4.2 to determine the functionalities of the tool. Then the chapter moved to details the design of different user stories, which have the role of determining the functionalities of the system. Finally, it gives an example of designing ontology that is used be the tool.

The next chapter gives details of the implementation process of the System.

5 IMPLEMENTATION

This chapter explains the implementation process of the system, "the Manchester Sushi Finder." Moreover, it contains some limitations that occurred during the implementation phase.

5.1 User Interface Implementation

This section includes information regarding the Manchester Sushi Finder implementation process. Firstly, it provides details about the technology and software process models that are used during the implementation. Then, it discusses their new functionalities and provides snapshots of the tool that is built on top of the Manchester Pizza Finder. Finally, it elaborates on the limitations that were faced during the implementation phase.

5.1.1 Java

The Manchester Sushi Finder was implemented using the Java programming language and was built on top of an existing tool, the Manchester Pizza Finder. Another reason to use the Java programming language would be the use of OWL API, which is a Java interface. Therefore, continued implementation in Java eases the process of the compatibility with the old tool and the accessibility of the old basic functionalities.

Both [21, 22] characterize java as two components in one: a programming language and a platform. The Java programming language is considered to be a high-level programming language. It has been described as simple, object-oriented, portable, and robust. The program is written in plain text, and then it gets compiled to bytecodes that are understandable to the Java Virtual Machine (JVM). Finally, it is run using the JVM that is available on most operating systems. Platforms usually consist of a software component that lies on top of the second part, which is the hardware component. Java is considered to be a platform as it contains the software without the hardware. Since JVM is available in most operating systems, Java can be run on different platforms. Moreover, it is simple to learn and use, and it is object-oriented and supports the reusability of modules. Although Java has a lot of advantages, it also has a major disadvantage, which is the slowness caused by the platform-independent environment.

5.1.2 Maven

The Manchester Pizza Finder is implemented using Java and developed with Maven. Since the Manchester Sushi Finder is built on the top of its predecessor, it is only natural to use Maven technology for the building process.

[23] defines Maven as knowledge accumulation. It is a tool that simplifies the building process. The goal of Maven can be summarized with five objectives. These objectives are easiness, uniformity, informativity, guidance, and migration to new features. The first objective is that Maven makes the building process easy for the developer since there is no need to know about the underlying mechanisms. The second offers a uniform way of building a project using the Project Object Model (POM). The third objective is providing the developer with a set of useful information. The fourth objective involves taking the best practices of the building process and steering the project in that direction. The last object talks about the easiness of updating the installation of Maven, so any change to Maven will be available to the developers.

5.1.3 OWL API

The OWL API has been used to link ontologies with tools (UIs) that are developed. As mentioned in section 2.2, the OWL API facilitates the creation, manipulation, and reason on ontologies used with tool. Using OWL API is an advantage since the project is developed using Java, and the OWL API is a Java library. In addition, the previous version of the Manchester Pizza Finder used the OWL API. Since the Manchester Sushi Finder is built on top of the earlier one, using it in the newer version makes it easy to use of the basic old functionalities that have been implemented before.

The OWL API has contributed many positive aspects to this project. The main contributions are reading ontologies, searching for annotations, rendering ontologies in different languages, and querying for result. It facilitates reading ontologies as they are being uploaded to the tool. It provides the tool with ingredient classes, switches between languages, and retrieves the results based on the entered criteria.

The next section will shed light on the complementary side of using the OWL API within this project.

5.1.4 Using Ontology Annotations

Annotation properties are one of the main pillars of this project. A prior study [24] defined annotations as metadata that can be associated with different parts of the ontology. The previous version of the UI of the Manchester Pizza Finder stores the configurations regarding the ontology in an external file. Therefore, why not store them in the ontology file itself instead?

The new finder uses the stored configurations as annotations inside the ontology. As a result, every ontology with these standard annotations will be used by the UI. Annotations determine the ingredient class to be displayed to the user, as shown in Figure 5. According to Figure 6, they have been used to determine the base class and the property used, in order to form a query. In addition, some new functions have been added by using annotations. Filters and facets are specified by using the hasRole annotation property, as mentioned earlier and shown in Figure 20 and Figure 21. Almost everything in the ontology that has been used by the UI is specified by using the hasRole annotation property.

5.1.5 Iterative and Incremental Development

The development process of the Manchester Sushi Finder has followed the iterative and incremental development model [25]. Since the iterative and incremental development model is a combination of the iterative design and incremental build model, five iterations are involved in development process. Each iteration is responsible for a major functionality that has been added to the tool. During the first iteration, configurations were added to the sushi ontology, and the tool was modified accordingly to be able to use ontologies containing the configurations mentioned in Figure 6. In the second iteration, filter functionality was implemented. New annotations were added to the ontology specifying filters, and then new algorithms to obtain the filters and display them were coded. In the third iteration, the sushi ontology was modified by adding new annotations to determine the facets to be applied to the search results, and then the tool was modified to find these facets and display them to the user while also adding their functionality. The fourth iteration is for displaying and switching between languages. In this iteration, the sushi ontology was modified by adding labels with different languages and then modifying the tool to display these available languages to the user. Finally, an additional view was added to the tool to offer two different views: tree and list view. After each iteration, a new working version of the tool was produced.

5.1.6 Uploading the Ontology to the Tool

The first thing a user needs to do after being annotated is upload the ontology. When the tool is run for the first time, the tool asks the user for required input like ontology, icon, and logo location; the dialog box is illustrated in Figure 22. Then the entered information gets stored in a configuration file that is created. Figure 23 demonstrates later runs after the first time. The tool retrieves the information from the configuration file and displays it to the user. Then the user chooses whether to modify this information and overwrite it or leave it unchanged. If the user decides to change the ontology, logo, or icon location, the user will see a screen like the one shown in Figure 22. However, if the user clicks on the OK button, the ontology will be loaded to the tool, and the application will launch.

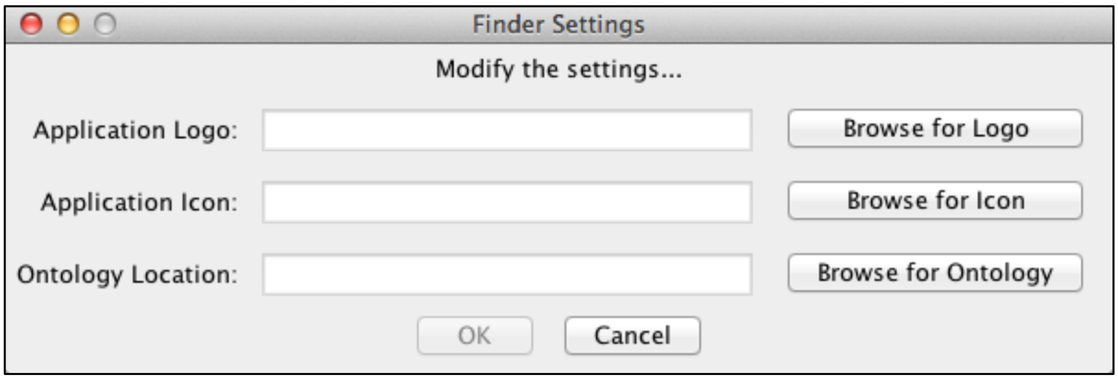
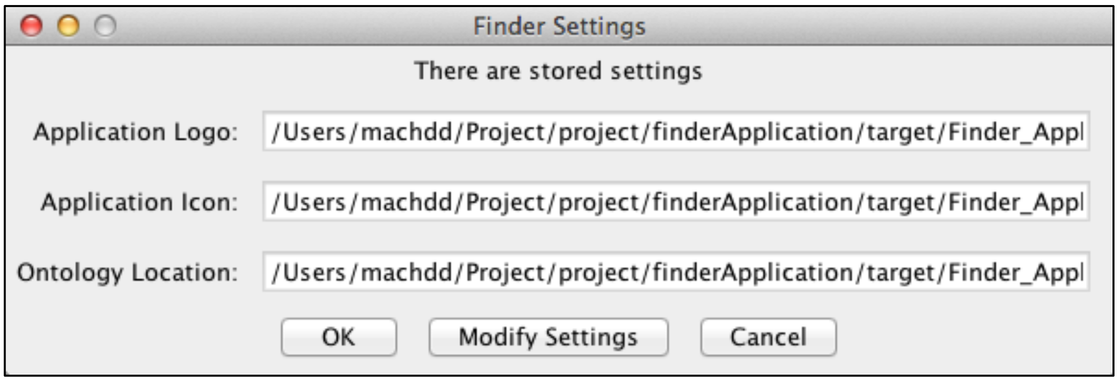


Figure 22: Tool's first run



Application Logo: /Users/machdd/Project/project/finderApplication/target/Finder_Appl

Application Icon: /Users/machdd/Project/project/finderApplication/target/Finder_Appl

Ontology Location: /Users/machdd/Project/project/finderApplication/target/Finder_Appl

Figure 23: Tool's later runs

5.1.7 The Tool uses different Ontologies

The tool uses only ontologies with special annotations that are mentioned in both Table 19 and Figure 5; if the ontology does not have these special annotations, then the tool will not work. When the tool is run, it first searches for a class using the hasRole annotation property with the constant value "IngClass." Then, it retrieves that class and all of its subclasses and renders them. Finally, the ingredients are displayed to the user. Figure 24 shows the sushi ontology ingredient class on the left side of the application window. The sushi ontology is the main ontology for this project as it was called the Manchester Sushi Finder. The previous version of the tool uses one ontology at a time since it requires some modifications on the configuration file. The pizza ontology was annotated with the standard annotations to drive the Manchester sushi finder. Figure 25 shows that the tool can use ontologies with standard annotations. Figure 24 and Figure 25 show the ability of uploading different ontologies by the same tool.

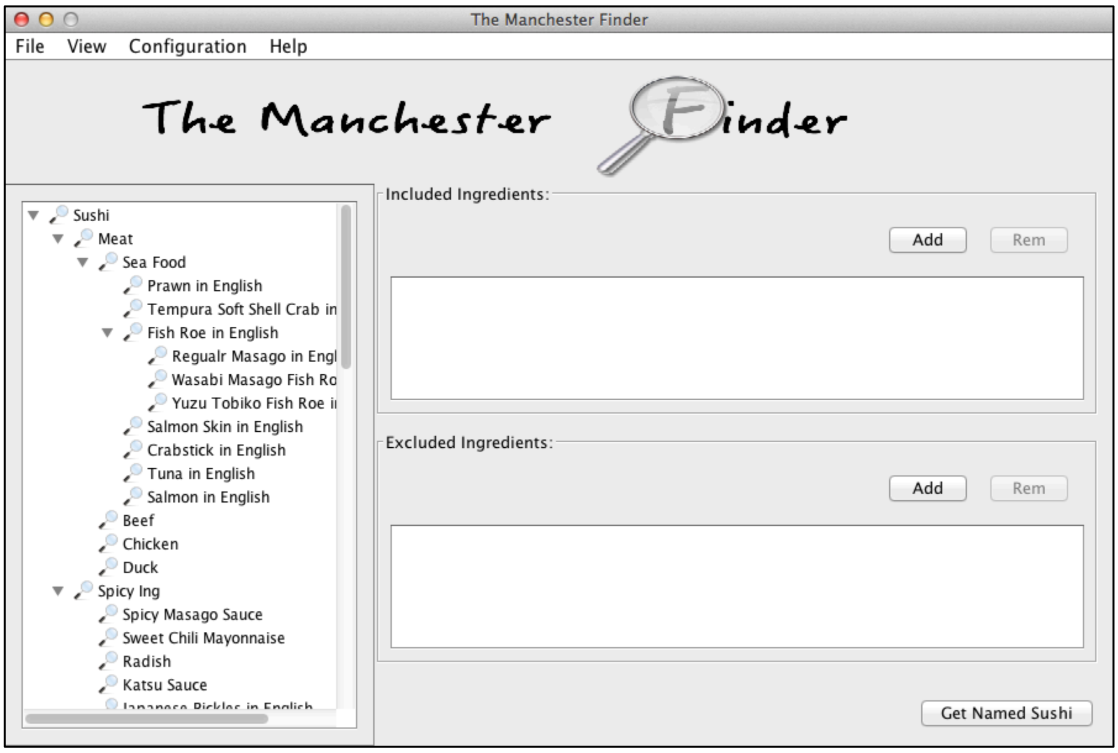


Figure 24: Sushi ontology ingredients

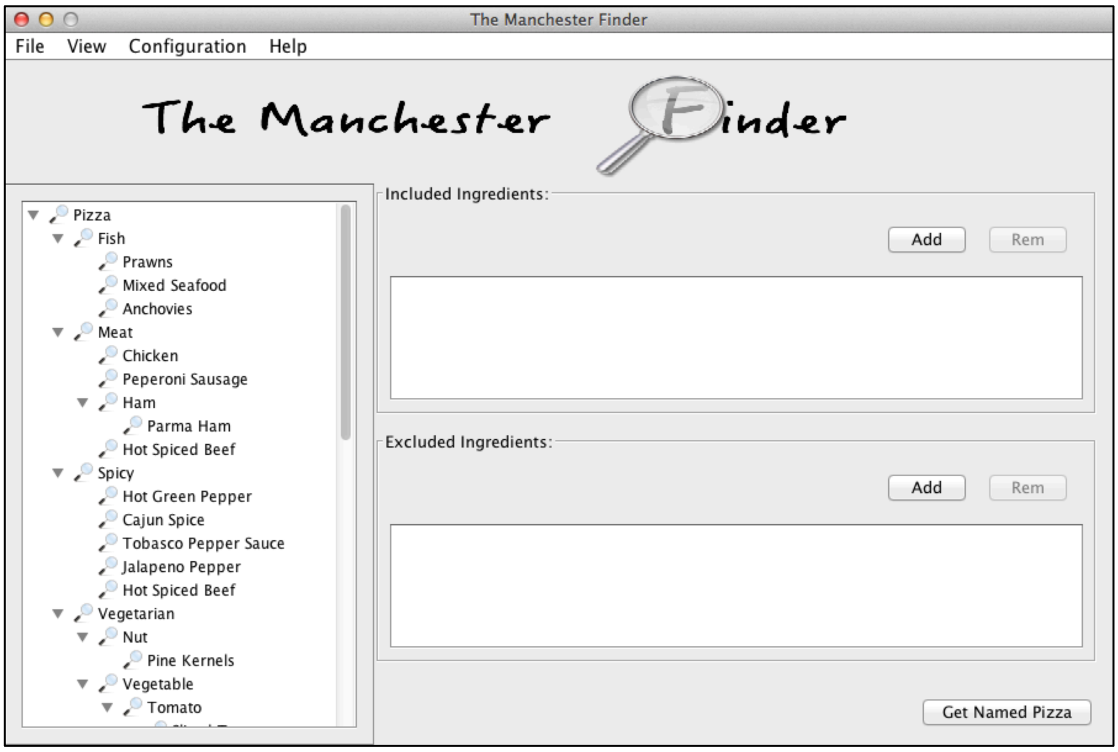


Figure 25: Pizza ontology ingredients

5.1.8 Filter Ingredients

Figure 24 shows the sushi ingredients without specifying filters in the sushi ontology. After annotating some subclasses of the ingredient class using the hasRole annotation property with the constant value "filter," as mentioned in Table 22, the tool will begin to display these filters. In Figure 26, three classes show as filters; these classes were annotated to be used as filters, and they include spicy, vegetarian, and vegan classes. The hasRole annotation property is associated with each one of these classes. When the application starts, the tool begins searching for classes with filter annotations in them and displays them to the user. After checking one of the filters, the tool highlights all classes that match the checked filter and disables the rest. As a result, the user can only select the highlighted ingredients.

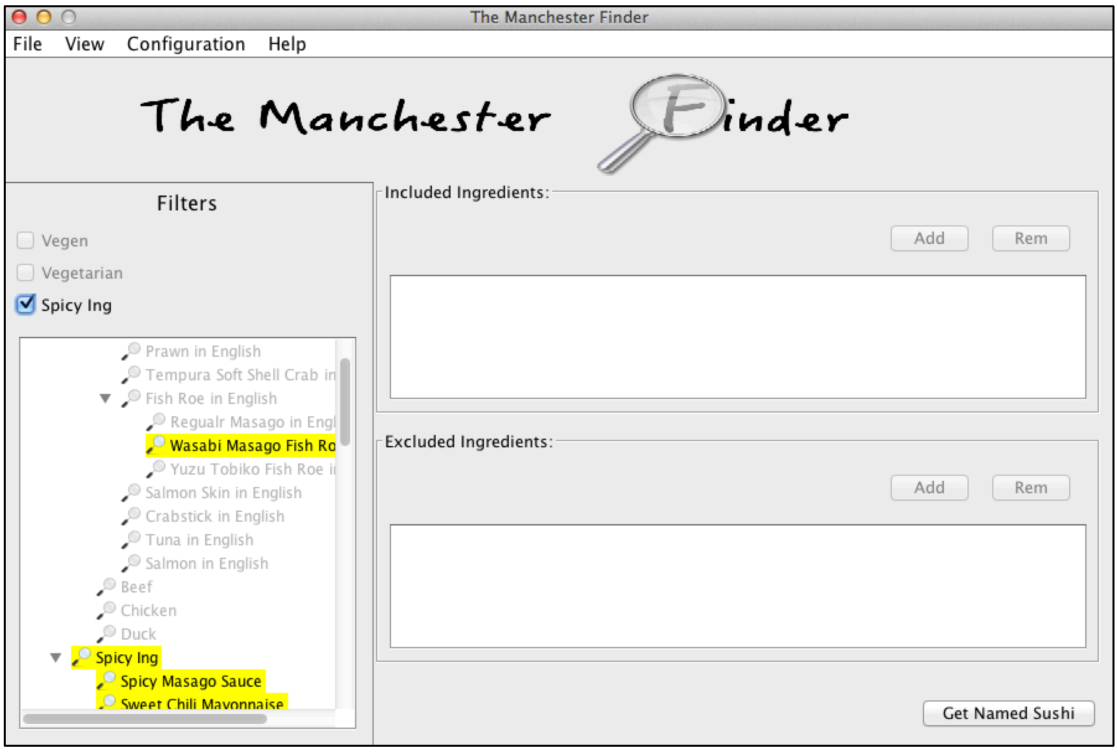


Figure 26: Sushi ontology filters

5.1.9 Filter the Search Result

Filters that are applied to the search results are called facets. Figure 27 shows the result of a search query that was applied to the sushi ontology without specifying any facets to be applied to the results. In order to apply facets to the search results, annotation properties need to be associated with value partition classes, as shown in Table 25. Value partition classes like the Spicy class need to have two annotation properties: hasRole and hasProperty. The hasRole property defines the role of the class that is a facet, and the hasProperty property determines which object property links the Spicy class with an ingredient class. After the sushi ontology has been annotated with facets annotations, a number of facets will be displayed when the user tries to query for something, depending on the annotations. Figure 28 shows the use of facets; when the user chooses one facet, the search results will change to match these facets. Facets are associated with ingredients rather than the original results. When the Spicy facet is chosen, all sushi components that have spicy ingredients will show up in the result panel. If the user chooses all results, then the unfiltered results will appear.

The facets used on the search result are represented in form of queries as follow:

Included ingredients:

[BaseClass] and ([Property] some [[selected Ing$_1$] and ([facet Property] some [facet$_i$])]) { i = 1, …, k}

[BaseClass] and ([Property] some [[selected Ing$_2$] and ([facet Property] some [facet$_i$])]) { i = 1, …, k}

…

…

[BaseClass] and ([Property] some [[selected Ing$_n$] and ([facet Property] some [facet$_i$])]) { i = 1, …, k}

Excluded ingredients:

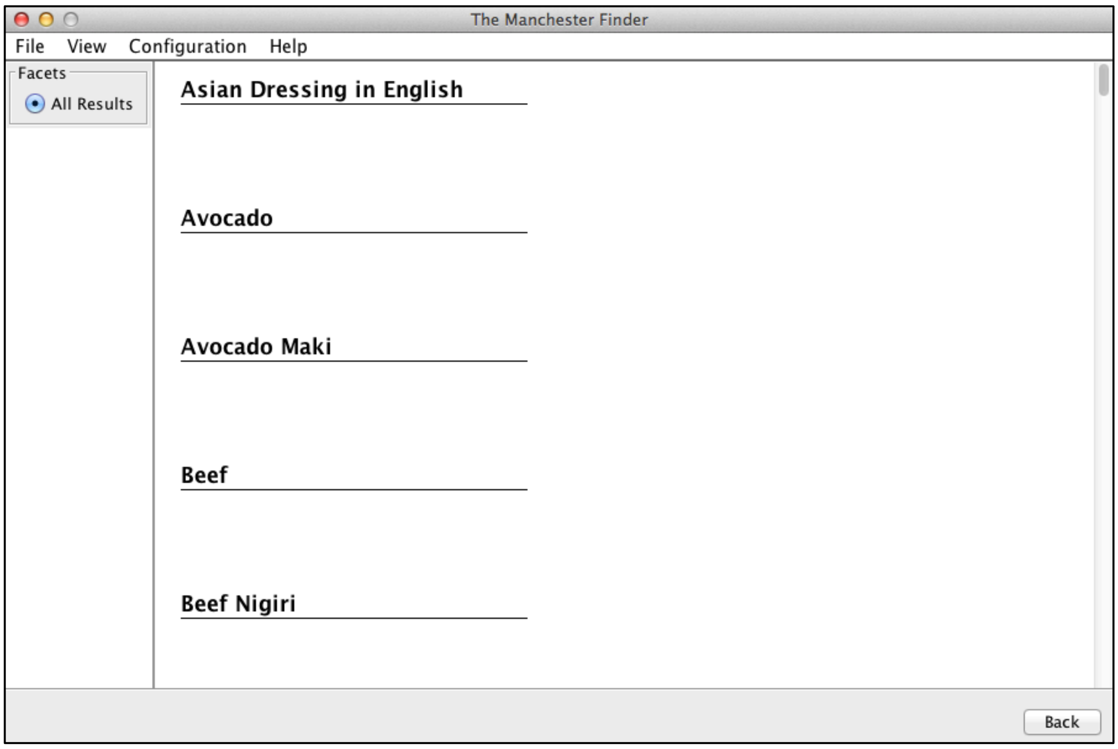
[BaseClass] and not ([Property] some [[selected Ing$_1$] and ([facet Property] some [facet$_i$])]) { i = 1, …, k}

[BaseClass] and not ([Property] some [[selected Ing$_2$] and ([facet Property] some [facet$_i$])]) { i = 1, …, k}

…

…

[BaseClass] and not ([Property] some [[selected Ing$_n$] and ([facet Property] some [facet$_i$])]) { i = 1, …, k}


The Manchester Finder

File View Configuration Help

Facets

(●) All Results

Asian Dressing in English

Avocado

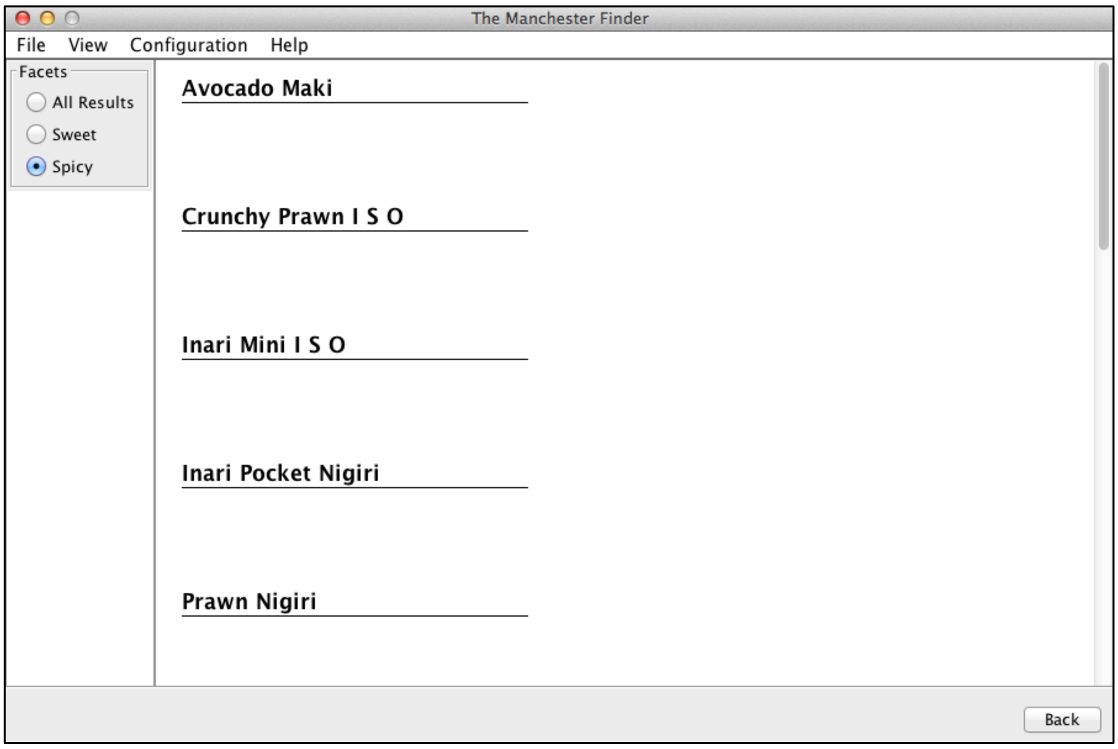
Avocado Maki

Beef

Beef Nigiri

Back

Figure 27: No facet is specified


The Manchester Finder

File View Configuration Help

Facets

( ) All Results

( ) Sweet

(●) Spicy

Avocado Maki

Crunchy Prawn I S O

Inari Mini I S O

Inari Pocket Nigiri

Prawn Nigiri

Back

Figure 28: Spicy facet

5.1.10 Displaying and Switching Between languages

The display of different languages in the tool depends on the ontology used. The tool shows available languages used to label the different classes in the ontology. The default language used is English. If the ontology has not been labelled with any language other than English, the tool will display the options and results in English. If the ontology was not labelled at all, the tool will show the names of the classes instead. Figure 29 demonstrates the available languages used to label the ontology classes. The tool shows the available languages along with the percentage of their representation to the whole ontology. For example, labels in the French language represent 51.83% of the ontology. Users can see the different languages that will go into the configuration, as well as the available languages. Figure 29 shows four different languages: English, French, Spanish, and Arabic. When the application is run, it starts looking for labels with different languages and calculates their percentages. Figure 30 shows the content of the ontology in the selected languages. Here, French is selected, so all the ontology components that are labelled with French will be shown in French; the ones without a French label will display in the default language, which is English. After querying, the results are returned in the selected language as well.

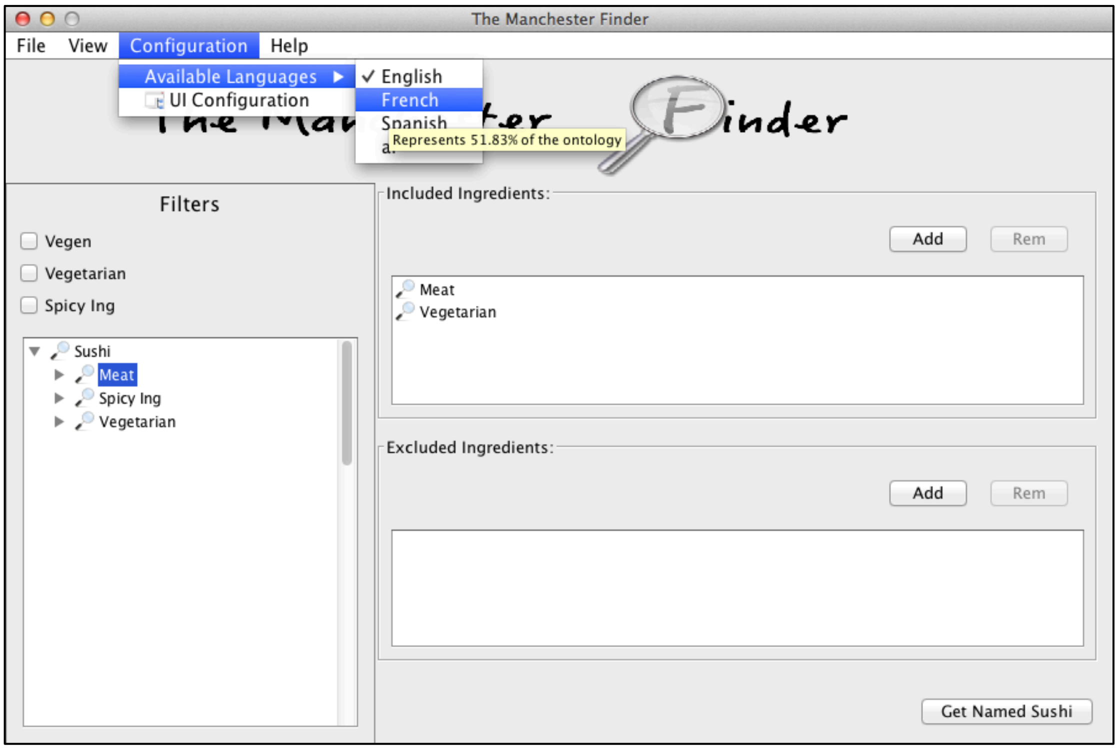


Figure 29: Shown available languages in the ontology

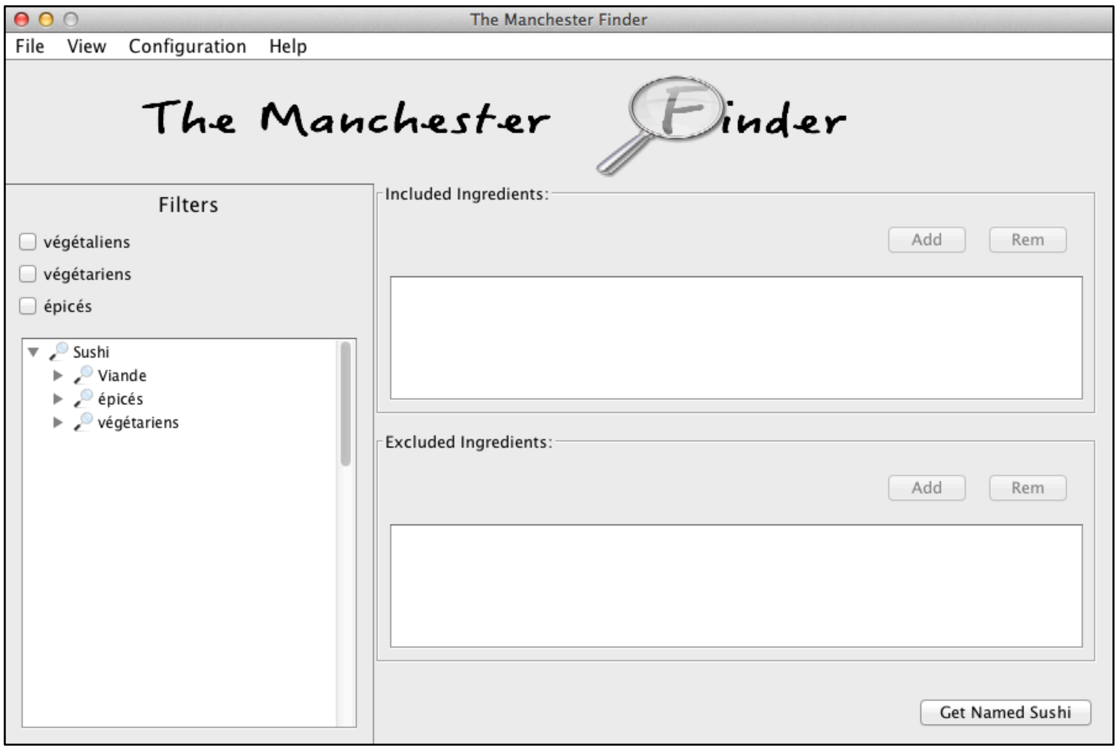


Figure 30: View after selecting the French language

5.1.11 Miscellaneous

There are two miscellaneous items that have a direct relationship to the UI and enhance it rather than being related to OWL ontology or the OWL API. These are important since they ease the process of using the UI; they are the different ingredient views and changing the UI label and button text during runtime.

The tool has two views for the ingredients: tree view and list view. In the tree view, ingredients are organized in a tree. Depending on the on the ontology that is used, the tree will change. The tree view contains a filter mechanism that is generated from the ontology. As shown in Figure 30, filters that are produced from the ontology are only present in the tree view. In contrast, the list view contains all of the ingredients in alphabetical order. List view has a different filtering mechanism since it filters the ingredients based on text. If the user enters text in the filter field, the ingredients that contain the text are shown in the viewer. The tool changes the view based on the user preferences; however, the default view is the tree view. Figure 31 shows the list view and the use of the filter field.

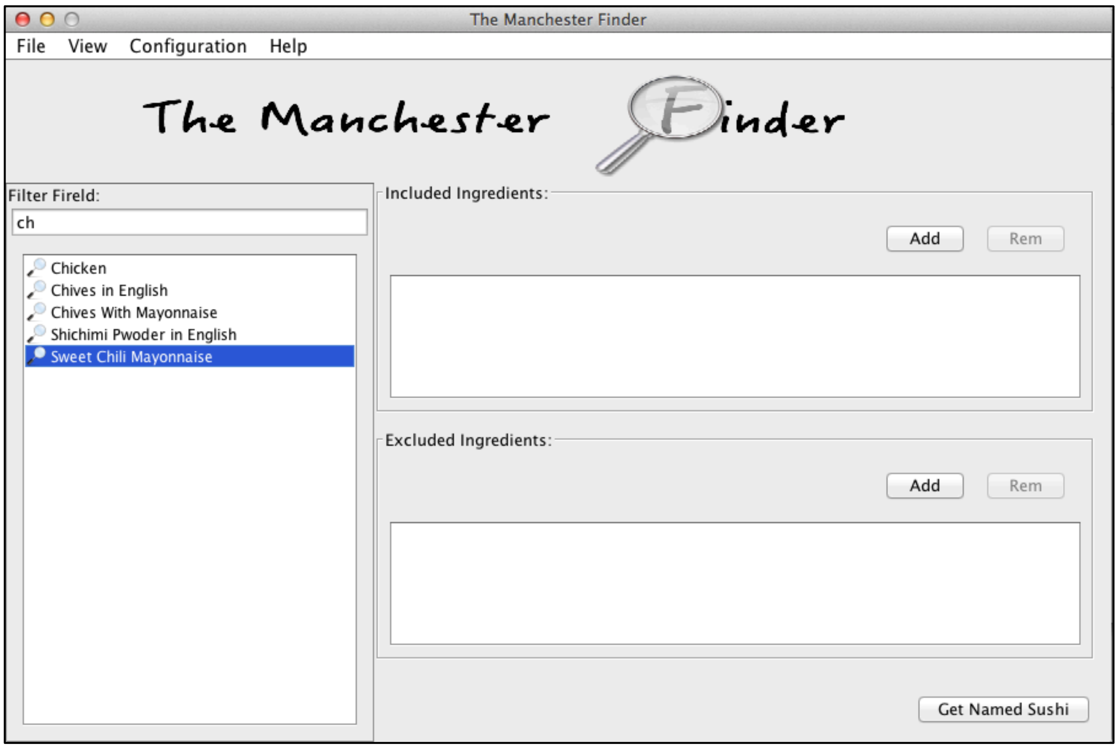


Figure 31: List view

Since the UI has the ability to display different languages depending on the labelled ontology, labels and buttons in the UI need to be changed to match the selected language. As shown in Figure 32, the user enters the desired text in any language, and clicks OK. The application then refreshes its views to reflect the entered text. Three labels can be changed: the filters heading label in the tree view and the Includes and the Excludes labels in the query panel. On the other hand, three fields can be used to add, remove, and get buttons.

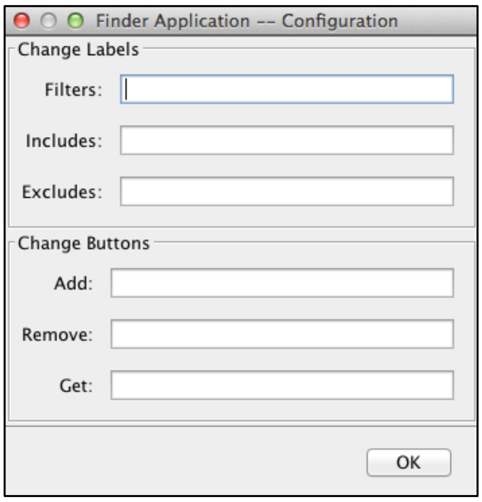


Figure 32: Change label and button text

5.2 Limitations

This section demonstrates some of the limitations that are faced during the implementation phase. There are a number of things that can go wrong. Most of the enhancements that interact with the ontology via the OWL API use ontology annotations. Annotations as metadata for the ontology classes drive most of the functionalities in the UI. There are two major limitations faced:

1. Human nature
2. OWL annotations

At the beginning of the project, it seemed acceptable to use annotations to drive the UI and its functionalities. Humans eventually will make mistakes. An ontology developer may misspell some of the annotations or may even forget to annotate the ontology that will be used. A developer may annotate the wrong classes or the wrong object properties. In contrast, the OWL itself suffers from some limitations regarding annotations. In the OWL, there are two types of classes: a named class that is created and defined by the ontology developer, and an unnamed class, which is a class expression. OWL does not support arbitrary class expressions to be annotated. It has to be encapsulated within a class to be annotated. Filters usually appear as filters options but not in the content that are selected from, moreover it filters the content displayed. Since the filters in OWL are subclasses of ingredient class then they show up in the ingredients displayed, which contradict the idea of filters. The best use of filters is with unnamed classes since the named ones are visible to the user; unnamed ones are perfect definitions for a filter. For example, the spicy class could be a filter because it filters all classes that are spicy. Therefore, the user should see the spicy filter but not the spicy class in the ingredient list, but that is not always the case. OWL annotation properties cannot be associated with unnamed class expressions; they can only be associated with IRI or Anonymous Individual, as stated in OWL 2 specifications in Figure 33. As a result of this limitation, the user sees the filter as a filter option and as a class in the ingredients hierarchy.

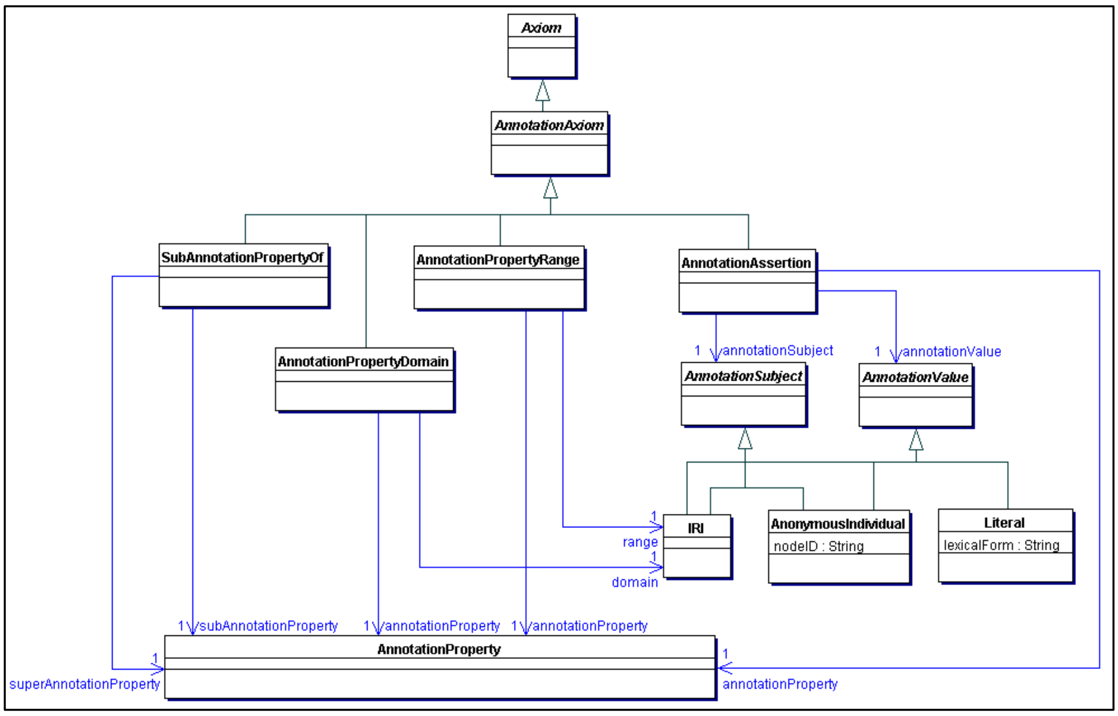


Figure 33: Annotations of IRIs and Anonymous Individuals in OWL 2 [24]

5.3 Conclusion

This chapter illustrated aspects of the implementation phase and provided a brief overview of the technologies used to implement the Manchester Sushi Finder. Then the chapter explained the major functionalities' implementation process. Finally, it discussed some limitations faced during the implementation phase.

The next chapter shows the testing process of the Manchester Sushi Finder.

6 EVALUATION AND CRITICAL ANALYSIS

In this chapter, an evaluation of the Manchester Sushi Finder tool is presented. The tool is evaluated based on an online survey.

6.1 Survey

The survey is accessible online via the SurveyMonkey website [26]. This website provides a free service to create surveys. A survey for the purpose of evaluating the Manchester Sushi Finder tool has been created. After the questions are created, the website then provides the developer with a link to access the questions and answer them [27]. This survey has been approved by the computer science school's committee [28]. The next subsections provide details about the survey. Details include the questions, participants, results, and the hypothesis acceptance.

6.1.1 Questions

The Manchester Sushi Finder was evaluated based on the evaluation plane in section 3.3. The questions have been divided into eight categories: configuration, views, tree filters, list filters, languages, facets, instructions, and feedback. The details of the categories are shown in Tables 1, 2, 3, 4, 5, 6, 7, and 8:

Category: Configuration		
Questions		
No.	**Question**	**Answer**
1	The application uses any ontology with standard annotations.	Rating
2	It is easy to annotate the ontology to make it work with the application.	Rating
3	The instructions to annotate the ontology are easy to follow.	Rating

Table 1: Configuration questions

Category: Views		
Questions		
No.	**Question**	**Answer**
1	The multi-view is a neat feature.	Rating
2	Having a multi-view eases data browsing.	Rating

3	It is easy to switch between tree and list views.	Rating

Table 2: Views questions

Category:	Tree Filtering	
Questions		
No.	**Question**	**Answer**
1	Filters are determined by the ontology file.	Rating
2	If the ontology specifies filters without annotation, then no filter will appear in the application window.	Rating
3	It is easy to add annotation to specify filters.	Rating
4	Filters narrow the content that the query is constructed from.	Rating
5	Filters ease the process of constructing a search query.	Rating

Table 3: Tree filtering questions

Category:	List Filtering	
Questions		
No.	**Question**	**Answer**
1	Filtering is not specified in the ontology file.	Rating
2	Filtering works with any ontology without any configuration.	Rating
3	Filtering eases the process of finding a specific ingredient.	Rating
4	Filtering narrows the content from which the query is constructed.	Rating

Table 4: List filtering questions

Category:	Language used in the ontology	
Questions		
No.	**Question**	**Answer**
1	The application shows the available languages used to be labelled within the ontology.	Rating
2	The application shows the percentage of language represented in the ontology.	Rating
3	It is easy to display different languages.	Rating
4	The default language is English.	Rating

Table 5: Used language in the ontology questions

Category:	Facets
Questions	

No.	Question	Answer
1	Facets help to narrow the search results.	Rating
2	Facets are determined in the ontology.	Rating
3	It is easy to add facets annotations to the ontology.	Rating
4	Different facets are displayed based on the ontology.	Rating

Table 6: Facets questions

Category: Instructions		
Questions		
No.	**Question**	**Answer**
1	The given instructions were easy to follow.	Rating

Table 7: Instructions question

Category: Comment		
Questions		
No.	**Question**	**Answer**
1	If you have any comments, please write them down here.	Text

Table 8: Comment question

The answers to the questions in Table 1 - Table 7 are multiple-choice type ratings where the answer consists of five Likert-type choices: Strongly Agree, Agree, Neither Agree nor Disagree, Disagree, and Strongly Disagree. The question in Table 8 is open ended and allows the user to input feedback in general by typing comments or suggestions.

6.1.2 Participants

This subsection discusses the best participants to evaluate the tool and answer the survey, the method of communication with the participants, the evaluation process, and the number of people who participated.

The right people to evaluate the tool are people who have basic knowledge about ontology engineering. Since the tool requires a configured ontology using annotations, people who know how to annotate ontology and where to annotate are the suitable ones to evaluate the tool. This experiment was conducted with the help of some classmates who have taken the Ontology Engineering and Web Semantics Web course unit. Therefore, they have the basic knowledge to annotate ontology and configure, run, and evaluate the tool. Participants were approached online via an email asking them to help evaluate my MSc project. The email contained the jar file of the application, the annotated ontology (the sushi ontology), un-annotated ontology (the pizza ontology), the logo image, the icon image, and an instructions file. First, the participant was asked to run the tool with an ontology that had specific annotations and to observe all the functionalities of the tool. After that, the participant was asked to annotate another ontology with specific annotations to make it work with the tool. For each major functionality to be added and viewed, the participant was required to add some annotations to the ontology. The email was sent to about 10 students. However, only 5 students responded and completed the evaluation and answered the survey. The lack of participants is due to not finding more people with knowledge in ontology engineering. Since the evaluator needed to have basic knowledge about ontology engineering, anyone without this knowledge cannot evaluate the tool.

6.1.3 Results

This subsection shows the participants' responses as collected by the SurveyMonkey website. Notice that most of the questions included a Likert-type rating scale where the answers ranged from Strongly Agree to Strongly Disagree. The last question is open-ended and contains a text box so the users can provide feedback. Answers follow the same structure as the one mentioned in 6.1.1, and the questions were divided into eight categories.

Figure 34 and Table 9 demonstrate the results of questions in the configuration category. For the first question, 100% of the participants thought that the tool was able to use ontologies with specific annotations; 60% of these participants strongly felt that this was the case. The second question received similar results; 100% of the participants agreed that the process of annotating the ontology to make it work with the tool was easy. Three-fifths (60%) strongly agreed, while two-fifths (40%) just agreed. The result for the last question in the configuration section was satisfactory, since 100% strongly felt that the instructions provided to annotate the ontology were easy to follow.

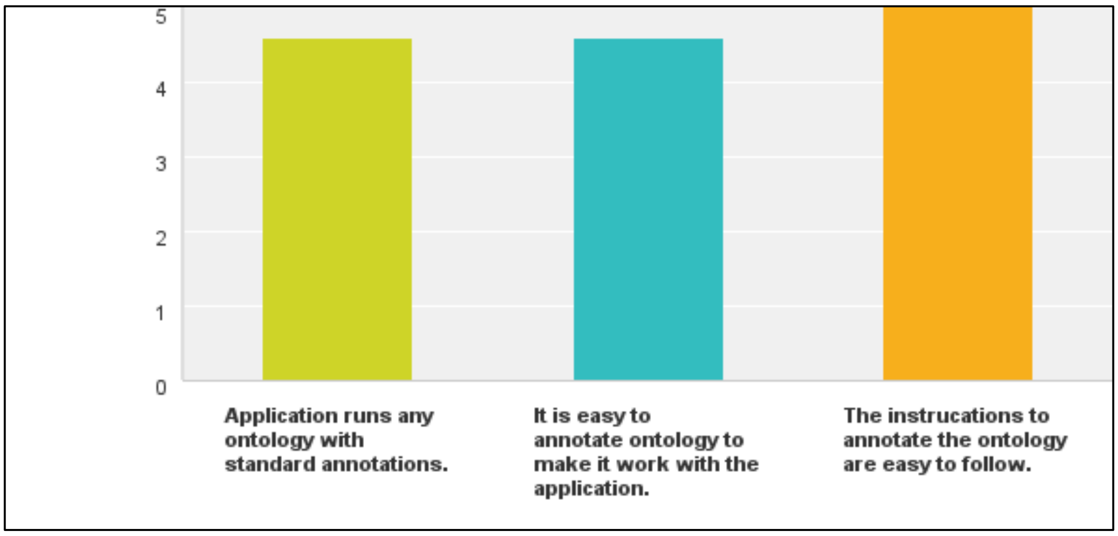


Figure 34: Result chart for the configuration section [26]

	Strongly Disagree	Disagree	Neither Disagree Nor Agree	Agree	Strongly Agree	Total	Average Rating
Application runs any ontology with standard annotations.	**0%** 0	**0%** 0	**0%** 0	**40%** 2	**60%** 3	5	4.6
It is easy to annotate ontology to make it work with the application.	**0%** 0	**0%** 0	**0%** 0	**40%** 2	**60%** 3	5	4.6
The instructions to annotate the ontology are easy to follow.	**0%** 0	**0%** 0	**0%** 0	**0%** 0	**100%** 5	5	5

Table 9: Result data table for configuration section [26]

Figure 35 and Table 10 demonstrate the results of the questions of the views section. There are three questions. For the first one, 100% of the participants reported that having different views was a neat feature; 40% liked it very much, whereas the rest were just satisfied with it. In the second question, all the evaluators thought that having different views made it easier to browse the ingredients easier. The majority (60%) strongly agreed that the process of browsing for ingredients was easy. Four-fifths (80%) of the participants reported that switching between views was easy, whereas the other 20% had a neutral opinion. Among the 80% who agreed, 60% strongly felt that it was easy to switch between views.

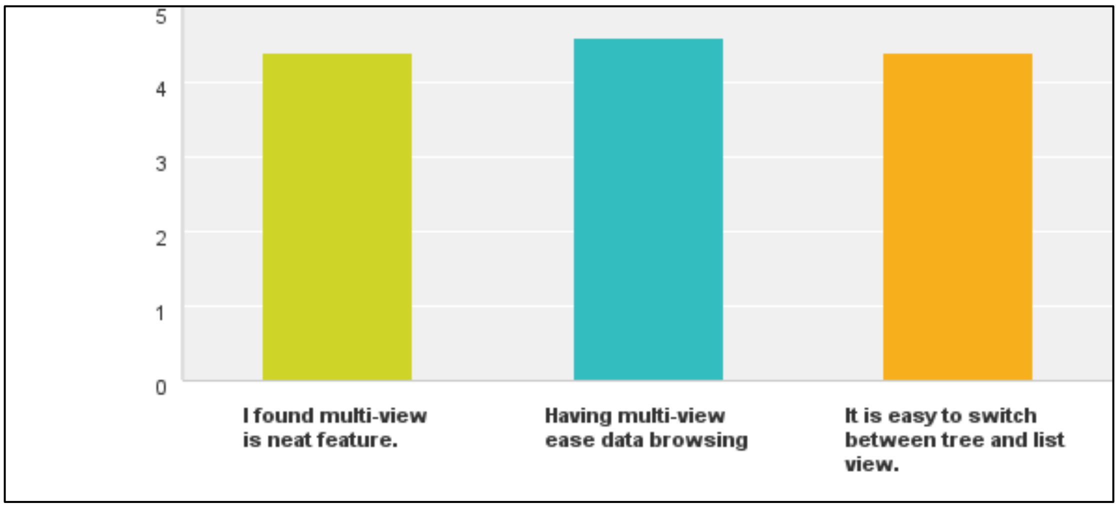


Figure 35: Result chart for the views section [26]

	Strongly Disagree	Disagree	Neither Disagree Nor Agree	Agree	Strongly Agree	Total	Average Rating
I found multi-view is neat feature.	**0%** 0	**0%** 0	**0%** 0	**60%** 3	**40%** 2	5	4.4
Having multi-view ease data browsing	**0%** 0	**0%** 0	**0%** 0	**40%** 2	**60%** 3	5	4.6
It is easy to switch between tree and list view.	**0%** 0	**0%** 0	**20%** 1	**20%** 1	**60%** 3	5	4.4

Table 10: Results data table for the views section [26]

Figure 36 and Table 11 demonstrate the results of the tree filter section where the questions concerned the filters in tree view. There were five questions in this section; 80% of the participants reported that the filters are specified in the ontology file. However, the remaining 20% did not have opinions about it. For the second question, all the participants felt if no filters were specified in the ontology, then none would show in the application tool. All the participants strongly agreed with the statement that adding filter annotations was very easy to do. For the fourth question, all evaluators reported that filters help to narrow down the list of ingredients to construct search queries. Finally, 80% responded that having filters eased the process of constructing search queries, whereas the remaining 20% had no opinion.

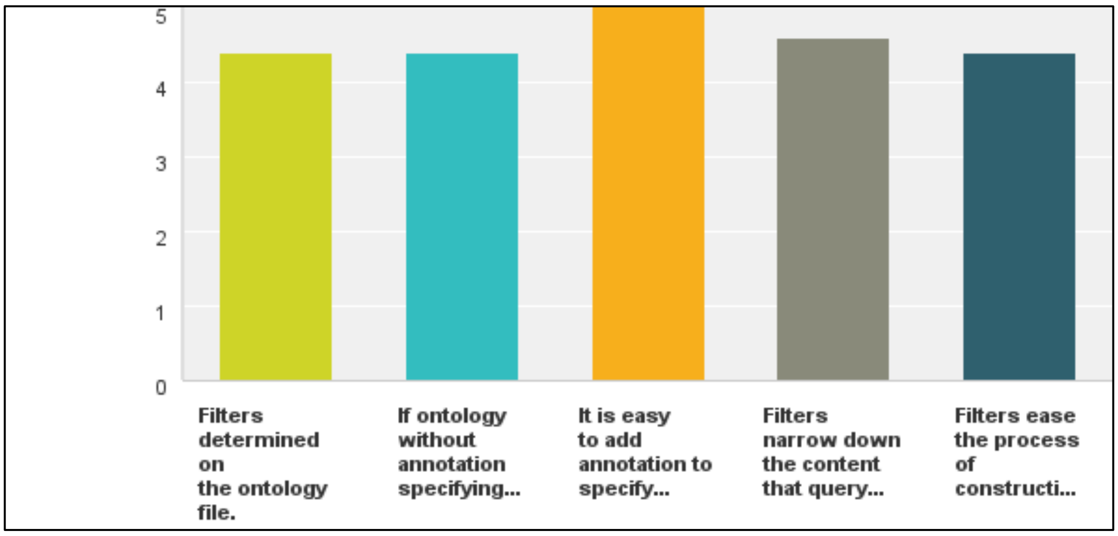


Figure 36: Result chart for the tree filtering section [26]

	Strongly Disagree	Disagree	Neither Disagree Nor Agree	Agree	Strongly Agree	Total	Average Rating
Filters determined on the ontology file.	**0%** 0	**0%** 0	**20%** 1	**20%** 1	**60%** 3	5	4.4
If ontology without annotation specifying filters, then no filter will appear in the application window.	**0%** 0	**0%** 0	**0%** 0	**60%** 3	**40%** 2	5	4.4
It is easy to add annotation to specify filters.	**0%** 0	**0%** 0	**0%** 0	**0%** 0	**100%** 5	5	5.0
Filters narrow down the content that query constructed from.	**0%** 0	**0%** 0	**0%** 0	**40%** 2	**60%** 3	5	4.6
Filters ease the process of constructing a search query.	**0%** 0	**0%** 0	**20%** 1	**20%** 1	**60%** 3	5	4.4

Table 11: Result data table for the tree filtering section [26]

Figure 37 and Table 12 show the result of the question in the list filtering section; 80% agreed that list filters were not specified in the ontology, whereas 20% thought that they were. As for the second question, all participants agreed that list filtering works with any ontology. The result of question three shows most of the participants

(80%) agreed that the filters ease the process of finding specific ingredients, whereas the rest (20%) had no opinion. The last question's result shows that 80% of the participants agreed that filtering narrows down the content of the ingredients to form a search query; however, the remaining 20% had no opinion.

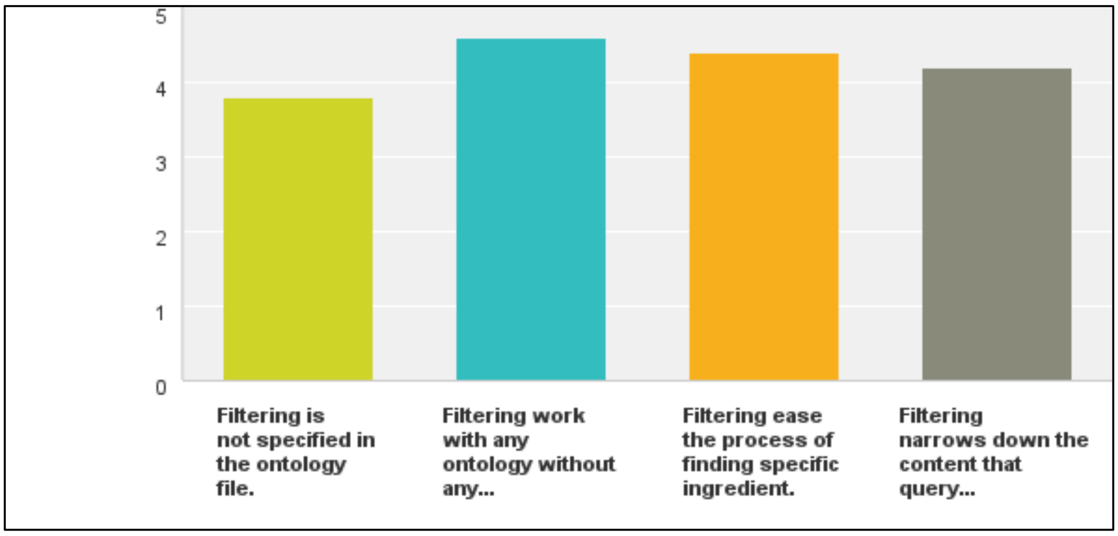


Figure 37: Result chart for the list filtering section [26]

	Strongly Disagree	Disagree	Neither Disagree Nor Agree	Agree	Strongly Agree	Total	Average Rating
Filtering is not specified in the ontology file.	**0%** 0	**20%** 1	**0%** 0	**60%** 3	**20%** 1	5	3.8
Filtering work with any ontology without any configuration.	**0%** 0	**0%** 0	**0%** 0	**40%** 2	**60%** 3	5	4.6
Filtering ease the process of finding specific ingredient. between tree and list view.	**0%** 0	**0%** 0	**20%** 1	**20%** 1	**60%** 3	5	4.4
Filtering narrows down the content that query constructed from.	**0%** 0	**0%** 0	**20%** 1	**40%** 2	**40%** 2	5	4.2

Table 12: Result data table for the list filtering section [26]

Figure 38 and Table 13 demonstrate the results of the questions for the language section. In the first question, 80% agreed that the tool shows the available languages that can be used to label ontology. However, 20% had no opinion. For the second question, the majority (60%) did not see a percentage of language represented in the ontology, whereas 40% did. In the third question, all participants agreed that displaying different languages was easy. As for the last question, the majority agreed that English was the default language; only 20% did not have an opinion.

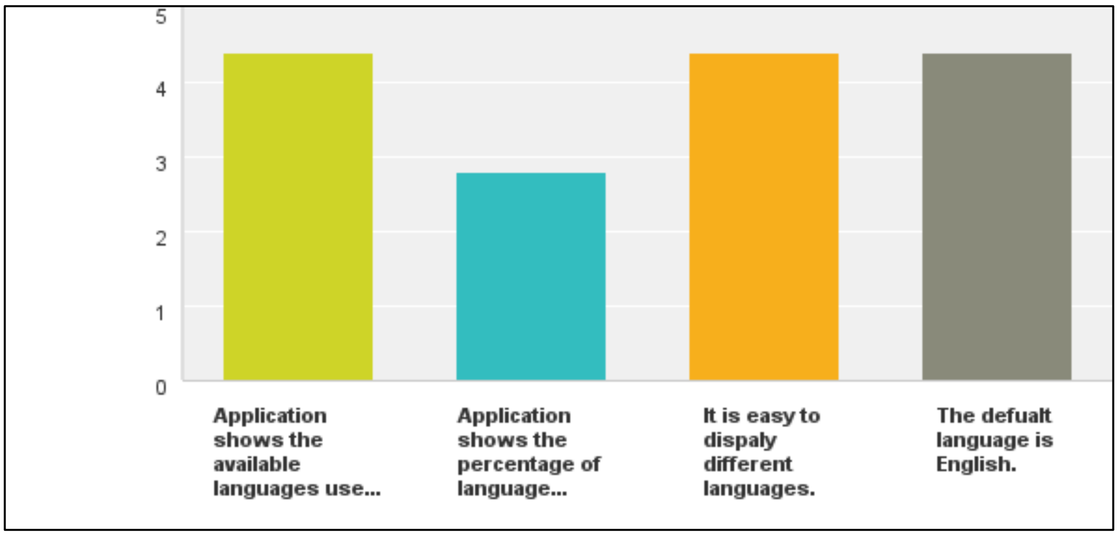


Figure 38: Result chart for language section [26]

	Strongly Disagree	Disagree	Neither Disagree Nor Agree	Agree	Strongly Agree	Total	Average Rating
Application shows the available languages used to be labeled within the ontology.	**0%** 0	**0%** 0	**20%** 1	**20%** 1	**60%** 3	5	4.4
Application shows the percentage of language represented in the ontology.	**0%** 0	**60%** 3	**0%** 0	**40%** 2	**0%** 0	5	2.8
It is easy to display different languages.	**0%** 0	**0%** 0	**0%** 0	**60%** 3	**40%** 2	5	4.4
The default language is English.	**0%** 0	**0%** 0	**20%** 1	**20%** 1	**60%** 3	5	4.4

Table 13: Result data table for the language section [26]

Figure 39 and Table 14 show the results of the questions in the facets section; 60% of respondents agreed that facets help narrow down the results of the search query. However, 40% offered no opinion. In the second question, all participants reported that facets are determined in the ontology. Similar results were found for both questions three and four. All participants thought that it was easy to annotate ontology with facets annotation, and the displayed facets depended on the ontology and its annotations.

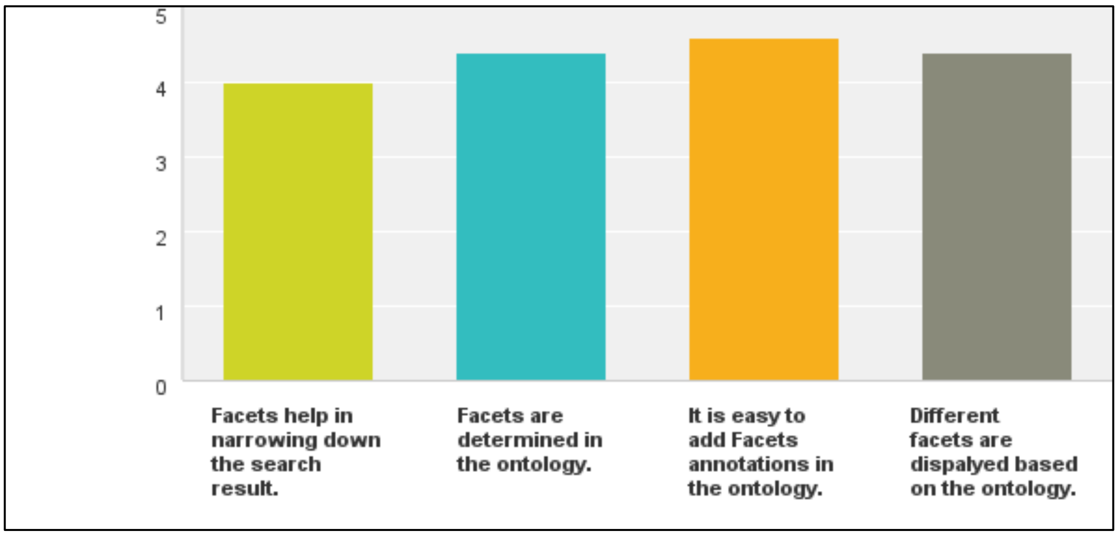


Figure 39: Result chart for the facets section [26]

	Strongly Disagree	Disagree	Neither Disagree Nor Agree	Agree	Strongly Agree	Total	Average Rating
Facets help in narrowing down the search result.	**0%** 0	**0%** 0	**40%** 2	**20%** 1	**40%** 2	5	4.0
Facets are determined in the ontology.	**0%** 0	**0%** 0	**0%** 0	**60%** 3	**40%** 2	5	4.4
It is easy to add Facets annotations in the ontology.	**0%** 0	**0%** 0	**0%** 0	**40%** 2	**60%** 3	5	4.6
Different facets are displayed based on the ontology.	**0%** 0	**0%** 0	**0%** 0	**60%** 3	**40%** 2	5	4.4

Table 14: Result data table for the facets section [26]

Figure 40 and Table 15 show the result of the instructions section, which contained only one question. All participants (100%) agreed that the given instructions were easy to follow.

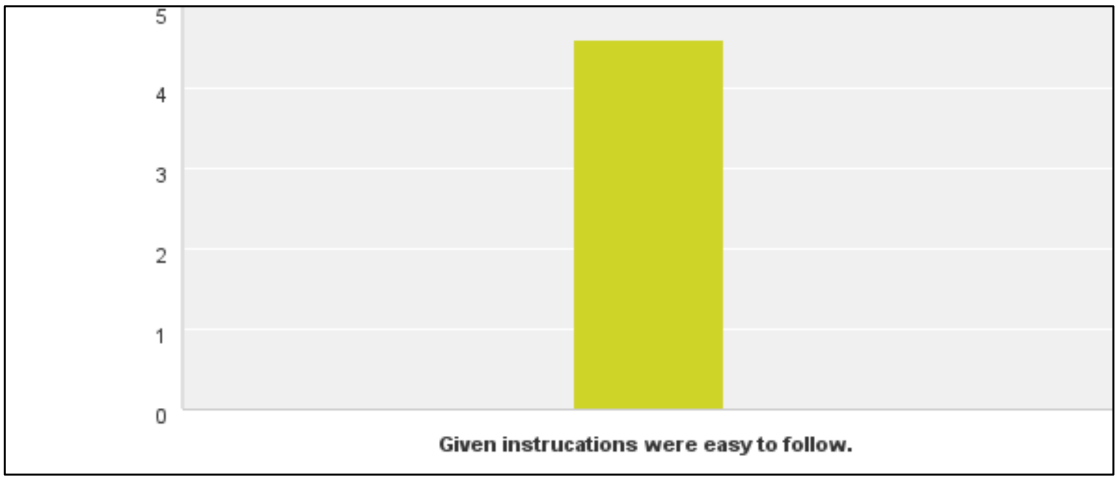


Figure 40: Result for the instructions section [26]

	Strongly Disagree	Disagree	Neither Disagree Nor Agree	Agree	Strongly Agree	Total	Average Rating
Given instructions were easy to follow.	**0%** 0	**0%** 0	**0%** 0	**40%** 2	**60%** 3	5	4.6

Table 15: Resulting data table for the instructions section [26]

The last section is the feedback section. Feedback was concerned with allowing more than one filter to be selected, and the percentage of a language is not shown. All feedback is true in some sense. Allowing more than one filter to be selected could be more desirable. The percentage of languages is shown, but the user needs to hover over the language to see it. However, this feature can be enhanced in the future to be easier to find.

6.1.4 Critical Analysis

In the project evaluation plan section 3.3, five qualities were used to evaluate the project. Those qualities intersect with the evaluation questions in section 6.1.1 to comprehensively evaluate the whole project.

	Deliverability	Accessibility	(Re)usability	Easily Configurable	Narrow down search query
Configuration	X	X	X	X	
Views	X	X			
Tree Filtering	X			X	
List Filtering	X				
Languages	X	X			
Facets	X			X	X
Instructions	X			X	

Table 16: Way for evaluating the project

Table 16 shows where the qualities to evaluate and the questions in the survey intersect. Therefore, deliverability intersects with all of the questions, which means the functionalities of all of these questions are delivered, and that measure has been met. The tool increases the accuracy due to the facet that user can configure to see different ontology, different views, and different languages. Due to the ability of the user use different ontologies, the reusability is increased as well. Following the instructions to add specific annotations with different ontologies, filters, and facets was easy, which indicates that the tool is easily configurable. According to the answers provided by the users, having facets narrows down the search results considerably. Having said all of that, the tool met the qualities used for evaluation.

Several limitations mentioned in section 5.2 like humane nature of making mistakes and inability of associating annotation properties to OWL class expressions have been faced during this project. However, there are a lot of benefits such as the flexibility to use different ontologies. There is a tradeoff relationship between using OWL annotations verses not using them.

6.2 Conclusion

In this chapter, evaluation of this project was conducted. This chapter started with the questions from the survey, and then participants who were qualified to evaluate the project were recruited. That section was followed with a discussion of the results gathered via the online survey. Finally, an analysis of the results is presented, which illustrates how the questions in the survey are related to the five qualities discussed in section 3.3.

The next chapter concludes the dissertation.

7 CONCLUSION AND FUTURE WORK

This conclusion chapter contains two sections. The first section is the project conclusion. The second one mentions any future work that can be conducted to make this project better.

7.1 Conclusion

A tool called the Manchester Sushi Finder has been developed to demonstrate the benefits of using ontologies to drive a UI. The conventional method of acquiring information depends on recalling keywords and then manually learning how to navigate a new querying tool. Moreover, sometimes it causes ambiguity when the user receives results that are remotely related to the keyword entered but are different in meaning. The tool developed for this project shows that the user is guided by the underlying ontology used with the tool. Comparing traditional keyword search and ontology based UI, the recall problem is reduced if not eliminated in the ontology based UI; since there is no need to remember keywords, and all required keywords are displayed to the user to select from and compose the search query. Ambiguity is reduced since there is no chance to receive query results that are different than expected; for example, if the user browses to query for food, then there is no ambiguity about what is needed. In addition, a facet mechanism is used to narrow down the search results even more, so ambiguity is reduced greatly.

This tool was developed based on an existing tool called the Manchester Pizza Finder. In this project, an ontology-based view and a faceted-based search method were adopted. This project has heavily depended on OWL annotations since it is the method the major functionalities have been using. OWL annotations have been used to determine roles for classes in the ontology.

Major functionalities have been developed to show the benefits of using ontology along with applications. The tool uses different ontologies with specific annotations. It also displays the content of the conceptual model using two different views: tree view and list view. It shows the available language within the ontology and switches between languages. It used the annotations to determine the filters on the content displayed in the tree view, whereas the list view has a text-based filtering mechanism and no need for a filter to be pre-determined. Instead, it makes use of facets to narrow down the search results.

Some limitations have been faced during this project. Human factors could cause potential difficulties with annotating the OWL ontologies. Developers might make mistakes, such as misspell annotations or even put them in the wrong place. Another factor is that OWL annotations have some limitations themselves, as only declared OWL entities can be annotated like declared classes. An arbitrary class that is equivalent to some OWL class expression cannot be annotated.

The project was evaluated using an online survey. The success of this project depends on five qualities that must be met: deliverability of the major functionalities, accessibility, (re)usability, configurability of the system, and specificity of the search results. These qualities are linked to questions in the survey. Despite the low sample size (five participants), the result is encouraging. All of the qualities have been met.

7.2 Future Work

This project could be improved in the following ways:

- Use the notion of query template in consideration. The current tool queries one basic thing, like sushi. Using a query template will allow users to query for sushi, sushi platter, or sushi try, etc.

- Allow more than one filter at a time to filter the ingredients.

- Find a better way to show the percentage of the language represented in the ontology.

- Add a mechanism to annotate the ontology if it is not annotated so it will work with the tool.

REFERENCES

1. Ding, L., et al., *Using Ontologies in the Semantic Web: A Survey*, in *Ontologies*. 2007, Springer US. p. 79-113.
2. Stevens, R., et al., *TAMBIS: Transparent Access to Multiple Bioinformatics Information Sources.* IBM System Journal. **40**(2): p. 532-551.
3. Catarci, T., et al., *An Ontology Based Visual Tool for Query Formulation Support.* ECAI, 2004: p. 308-312.
4. Horridge, M., *The Manchester Pizza Finder*. University of Manchester.
5. Bechhofer, S. and N.W. Paton, *Ontology Visual Querying*, in *Encyclopedia of Database Systems*. 2009, Springer.
6. Bechhofer, S. *The Manchester Sushi Finder - Project Page*. 2014 [cited 2014 March 5, 2014]; Available from: http://studentnet.cs.manchester.ac.uk/pgt/2013/COMP60990/project/projectbookdetails.php?projectid=20889.
7. Group, W.C.O.W. *Web Ontology Language (OWL)*. 2012 [cited 2014 April 19, 2014]; Available from: http://www.w3.org/2001/sw/wiki/OWL.
8. Davis, R., H. Shrobe, and P. Szolovits, *What is a Knowledge Representation?*, in *AI Magazine*. 1993. p. 17-33.
9. Group, O.W. *OWL Web Ontology Language Overview*. 2004 [cited 2014 May 13, 2014]; Available from: http://www.w3.org/TR/owl-features.
10. M, H. and B. S, *The OWL API: A Java API for OWL ontologies.* Semantic Web, 2011. **2**(Number 1 / 2011): p. 11-21.
11. Group, O.W. *OWL 2 Web Ontology Language Document Overview*. 2012 [cited 2014 May 13, 2014]; Available from: http://www.w3.org/TR/owl2-overview/.
12. Wikipedia. *Application programming interface*. [cited 2014 April 21, 2014]; Available from: http://en.wikipedia.org/wiki/Application_programming_interface.
13. *The size of the World Wide Web (The Internet)*. 2014 [cited 2014 Jun 2, 2014]; Available from: http://www.worldwidewebsize.com/.
14. Hyvönen, E., S. Saarela, and K. Viljanen, *Application of ontology techniques to view-based semantic search and browsing*, in *The Semantic Web: Research and Applications*. 2004, Springer. p. 92-106.
15. Bechhofer, S., et al., *Guiding the User: An Ontology Driven Interface.*
16. Bechhofer, S. and C. Goble, *Classification Based Navigation and Retrieval for Picture Archives*, in *Database Semantics*. 1999, Springer. p. 291-310.
17. Schreiber, A.T.G., et al., *Ontology-based photo annotation.* IEEE Intelligent Systems, 2001. **16**(3): p. 66-74.
18. Wikipedia. *Faceted search*. [cited 2014 April 23, 2014]; Available from: http://en.wikipedia.org/wiki/Faceted_search.
19. Smith, D.A. and N.R. Shadbolt, *FacetOntology: Expressive Descriptions of Facets in the Semantic Web*, in *Semantic Technology*. 2013, Springer. p. 223-238.
20. Catarci, T., et al., *Visual query systems for databases: A survey.* Journal of Visual Languages and Computing, 1997. **8**: p. 215-260.
21. *The Java Technology Phenomenon*. [cited 2014 July 31, 2014]; Available from: http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html.

22. Gosling, J. and H. McGilton, *The Java language environment*. Vol. 2550. 1995: Sun Microsystems Computer Company.
23. *Apache Maven Project*. [cited 2014 July 31, 2014]; Available from: http://maven.apache.org/what-is-maven.html.
24. *OWL 2 Web Ontology Language*
Structural Specification and Functional-Style Syntax. [cited 2014 Aug 1, 2014]; Available from: http://www.w3.org/TR/owl2-syntax/.
25. Larman, C. and V.R. Basili, *Iterative and incremental development: A brief history.* Computer, 2003. **36**(6): p. 47-56.
26. SurveyMonkey. 2014; Available from: https://http://www.surveymonkey.net.
27. Al Abbas, H. *MSc Project (Software application-"The Manchester Sushi Finder")* 2014 [cited 2014 Aug 6, 2014]; Available from: https://http://www.surveymonkey.com/s/NLRQZ37.
28. Ethics, C.S. *Survey*. 2014 [cited 2014 Agu 6, 2014]; Available from: http://ethics.cs.manchester.ac.uk.

APPENDIX USER STORIES

The card As a [user], I want [functionality], so that [value].
Story Narrative: Access
As an end user, **I want** to have access to the tool **so that** I can run it. **As** a tool provider, **I want** to run the tool on a platform **so that** the customer can use it.
The conversation
As an end user, **I want** to have access to the tool **so that** I can run it. • Make the tool accessible. **As** a tool provider, **I want** to run the tool on a platform **so that** the customer can use it. • Provide the tool.
The confirmation
As an end user, **I want** to have access to the tool **so that** I can run it. 1. Click on the tool icon to run it. **As** a tool provider, **I want** to run the tool on a platform **so that** customer can use it. 1. Click on the tool icon and run it for the end user.

Table 17: Access user story

The card As a [user], I want [functionality] so that [value].
Story Narrative: Run 1
As an end user, **I want** to run the tool **so that** I can use it. **As** a tool provider, **I want** upload subject ontology to the tool **so that** end users can use it. As an ontology developer, **I want** to annotate subject ontology **so that** it can be used by the tool.
The conversation
As an end user, **I want** to run the tool **so that** I can use it. • On running the tool, it runs correctly. • The tool runs on different platforms.

As a tool provider, **I want** upload subject ontology to the tool **so that** end user can use it. • Tool uses annotated ontologies. • Tool does not use annotated ontologies. As an ontology developer, **I want** to annotate the subject ontology **so that** it can be used by the tool. • Ontology is annotated with basic annotations to allow it to be used by the tool.
The confirmation
As an end user, **I want** to run the tool **so that** I can use it. 1. Click on the tool icon to run it. 2. Click on OK to continue without modifying the settings of the tool. **As** a tool provider, **I want** to upload subject ontology to the tool **so that** the end user can use it. 1. Click on the tool icon to run it. 2. Browse for ontology. 3. Browse for the tool icon. 4. Browse for the tool logo. 5. Click on OK to upload the files and configure the tool.

Table 18: Run 1 user story

The card As a [user], I want [functionality] so that [value].
Story Narrative: Run 2
As an end user, **I want** to load different ontologies during runtime **so that** I can browse and query for results. **As** a tool provider, **I want** to load different ontologies during runtime **so that** the end user can use them. As an ontology developer, **I want** to annotate subject ontologies **so that** they can be used by the tool.
The conversation
As an end user, **I want** to load different ontologies during runtime **so that** I can browse and query for results.

- Tool allows upload ontology in the first run.
- Tool allows users to browse for different annotated ontologies during runtime.
- Tool allows files with the .owl extension to be uploaded.
- Tool prompts the user with a warning if the ontology is being used and asks if the user wants to overwrite it.

As a tool provider, **I want** to load different ontologies during runtime **so that** the end user can use them.

- Tool allows users to upload ontology in the first run.
- Tool allows users to browse for different annotated ontologies during runtime.
- Tool allows files with the .owl extension to be uploaded.
- Tool prompts the user with a warning if the ontology is being used and asks if the user wants to overwrite it.

As an ontology developer, **I want** to annotate subject ontologies **so that** they can be used by the tool.

- Provide ontologies annotated with basic metadata to be used through the tool.

The confirmation

As an end user, **I want** to load different ontologies during runtime **so that** I can browse and query for results.

1. Click on the File menu.
2. Click on Import New Ontology.
3. Browse for the new ontology file.
4. Click OK.
5. The tool asks the user if the same ontology is being used and if user wishes to overwrite it.
6. Application is refreshed with new ontology.

As a tool provider, **I want** to load different ontologies during runtime **so that** the end user can use them.

1. Click on the File menu.
2. Click on Import New Ontology.
3. Browse for the new ontology file.
4. Click OK.
5. The tool sks the user if the same ontology is been used and if the user wishes to

overwrite it. 6. The application is refreshed with the new ontology. **As** an ontology developer, **I want** to annotate subject ontologies **so that** they can be used by the tool. 1. Annotate the ontologies with the hasRole object property. 2. Specify the base class using the hasRole property with the constant value "BaseClass." 3. Specify the ingredient class using the hasRole property with the constant value "IngClass." 4. Specify the object property linking the base class with the ingredient class using the hasRole property with the constant value "Property."

Table 19: Run 2 user story

The card
As a [user], I want [functionality] so that [value].
Story Narrative: View
As an end user, **I want** to view ingredients in both tree and list views **so that** I can browse and query for results using different views.
The conversation
As an end user, **I want** to view ingredients in both tree and list views **so that** I can browse and query for results using different views. • The tool provides two views of the content "ingredients": tree and list view. • The tool supports switching between views during runtime. • The tool permits users to select content from the two views.
The confirmation
As an end user, **I want** to view ingredients in both tree and list views **so that** I can browse and query for results using different views. 1. Click on the View menu on the menu bar. 2. Click on Tree or List View. 3. The application will refresh with the new view.

Table 20: View user story

The card

As a [user], I want [functionality] so that [value].

Story Narrative: Language

As an end user, **I want** to see the available languages used in the ontology **so that** I can browse and query for results in different languages.

As an ontology developer**, I want to** label the ontology classes with different languages **so that** the user can choose the desired language.

The conversation

As an end user, **I want** to see the available languages used in the ontology **so that** I can browse and query for results using different languages.

- The tool shows available languages used in the ontology file.
- The tool displays the percentage of a language usage in an ontology file.
- The tool switches languages when the user chooses a language.

As an ontology developer**, I want to** label the ontology classes with different languages **so that** the user can choose an appropriate language.

- Label classes in the ontology with different languages.

The confirmation

As an end user, **I want** to see the available languages used in the ontology **so that** I can browse and query for results using different languages.

1. Click on the configuration menu on the menu bar.
2. View the available languages drop down menu.
3. Click on the desired language.
4. The application will refresh with the new language.

As an ontology developer**, I want to** label the ontology classes with different languages **so that** the user can choose an appropriate language.

1. Open the ontology with Protégé.
2. Annotate classes with the label annotation property.
3. Choose a language for the label.
4. Write down the label value as a constant value.
5. Click OK.
6. Save the ontology file.

Table 21: Language user story

The card
As a [user], I want [functionality] so that [value].
Story Narrative: Tree View Filter
As an end user, **I want** to filter the ingredient tree **so that** I can browse and query for results easily. **As** an ontology developer**, I want to** specify filters on the ontology **so that** users can filter the ingredient tree.
The conversation
As an end user, **I want** to filter the ingredient tree **so that** I can browse and query for results easily. • The tool shows filter options. • The tool filters ingredients by highlighting them and disabling the rest. • The tool does not show filters if they are not specified in the ontology. **As** an ontology developer**, I want to** specify filters for the ontology **so that** the user can filter the ingredient tree. • Annotate ingredient classes to be used as filters.
The confirmation
As an end user, **I want** to filter the ingredient tree **so that** I can browse and query for results easily. 1. In tree view, choose one of the check boxes in the filter section. 2. Ingredients are filtered based on the chosen criteria. 3. Filtered ingredients are highlighted. 4. The unmatched content is disabled and cannot be chosen. **As** an ontology developer**, I want to** specify filters on the ontology **so that** the user can filter the ingredient tree. 1. Open the ontology with Protégé. 2. Annotate the desired ingredient classes with the hasRole annotation property. 3. Write down the “filter” value as a constant value. 4. Click OK. 5. Save the ontology file.

Table 22: Tree View Filter user story

The card As a [user], I want [functionality] so that [value].
Story Narrative: List View Filter
As an end user, **I want** to filter the ingredient list **so that** I can browse and query for results easily.
The conversation
As an end user, **I want** to filter the ingredient list **so that** I can browse and query for results easily. • The tool shows the filter text field. • The tool filters the ingredients by displaying only the ingredients that match the input text.
The confirmation
As an end user, **I want** to filter the ingredient list **so that** I can browse and query for results easily. 1. In list view, input filter text in the text field. 2. The ingredients are filtered based on the input criteria. 3. The filtered ingredients are displayed. 4. The unmatched content is not displayed and cannot be chosen.

Table 23: List View Filter user story

The card As a [user], I want [functionality] so that [value].
Story Narrative: Query
As an end user, **I want** to query for sushi using the tool **so that** I can receive results easily.
The conversation
As an end user, **I want** to query for sushi using the tool **so that** I can receive results easily. • The tool shows ingredients and two input lists for including and excluding ingredients. • The tool displays the results for the sushi ingredients that match the included and excluded ingredients.
The confirmation

As an end user, **I want** to query for sushi using the tool **so that** I will receive results easily. 1. In list or tree view, select the desired ingredient. 2. Click on the add button to add the selected ingredient to the included or excluded lists. 3. Click on the remove button to remove the selected ingredient from the included or excluded lists. 4. Click on the run button. 5. The tool shows the results. 6. Click on the back button to navigate back to another query.

Table 24: Query user story

The card As a [user], I want [functionality] so that [value].
Story Narrative: Facet
As an end user, **I want** to filter the results of a query **so that** I can narrow them down and find the desired result quickly and easily. **As** an ontology developer**, I want to** specify facets of the ontology **so that** the user can filter the search query results.
The conversation
As an end user, **I want** to filter the results of a query **so that** I can narrow them down and find the desired result quickly and easily. • The tool shows the facet options upon retrieving the results. • The tool filters the results. • The tool shows new results upon applying the facets. • Different facets have different filtered results. **As** an ontology developer**, I want to** specify facets on the ontology **so that** the user can filter the search query results. • Annotate the value partition classes to be used as facets.
The confirmation
As an end user, **I want** to filter the results of a query **so that** I can narrow them down and find the desired result quickly and easily. 1. Upon retrieving the results, choose one of the radio buttons in the facet section.

2. The result view will refresh with the new filtered result.
3. Choose the All Result radio button to obtain the original result.

As an ontology developer**, I want to** specify facets on the ontology **so that** the user can filter the search query results.

1. Open the ontology with Protégé.
2. Annotate the desired partition value classes with the hasRole annotation property.
3. Write down the facet value as a constant value.
4. Annotate the desired partition value classes with the hasProperty annotation property.
5. Choose theIRI tab to input the value of hasProperty.
6. Choose the Object Properties tab.
7. Choose the object property that connects the ingredient with the partition class.
8. Click OK.
9. Save the ontology file.

Table 25: Facet user story