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

Words Count: 15560

LIST OF ABBREVIATIONS

AI	Artificial Intelligent
API	Application Programming Interface
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 Modeling Language
XML	Extensible Markup Language

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This project is designed to show the benefits of using applications, which are ontology driven, in term of browsing and querying for information. It demonstrates the use of the represented knowledge between machines instead of sharing just raw data. Since, the web full of raw information that could or could not has relevancy with each other, OWL ontology language came to represent the knowledge of domains instead of raw data. OWL simulates the intelligence behind the reasoning process in addition to knowledge representation. By doing this, relations between different objects within a domain are represented as well.

Browsing and querying are two of the main characteristics of a retrieval system. Usually, user tries to figure out the functionalities of a user interface or some instructions are provided to guide the user. As for querying, in conventional querying system that is keywords based rather than the underlying concept, the process of retrieving information depends on recalling specific keywords. This method suffers some issues like the recall of keywords and ambiguity in the search query formation process.

Those issues can be reduced be adopting ontology-based method and faceted-based search mechanism. Representing knowledge within ontologies will drive the interface and take care of guiding the user toward building only valid search queries. The recall problem will be reduced since user does not have to remember keywords and all relevant query elements derived automatically from the ontology. As for ambiguity, faceted-based search is introduced to narrow and personalize the search result.

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

The application has the same basic functionalities with the Manchester Pizza finder and it is called The Manchester Sushi Finder that is a tool to query for sushi based on included and excluded ingredients. Although, the main ontology used is based on a sushi menu restaurant, does not mean only sushi ontology will work. In 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 of this or any other university or other institute of learning.

INTELLECTUAL PROPERTY STATEMENT

ACKNOWLEDGMENT

1 INTRODUCTION

1.1 Motivation

As the trend nowadays to try making machines more intelligent [1], sharing knowledge of information instead of sharing the raw data in the web is becoming more desirable. Thinking computers that are able to able to understand, sharing knowledge and simulate the reasoning process of human would seems an idea from Artificial Intelligent (AI) fiction movie. Semantic web is helping in converting 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 been 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 Transfer Protocol (HTML). They are just codes to the machines and they display it to users regardless of what knowledge needed. So, Sematic web came along for machines to make sense of retrieved information. Ontologies are used to represent knowledge and make inferences from that knowledge using machines computational capabilities and some reasoning techniques such as description logics.

An intelligent way of representing knowledge needs an intelligent way of browsing and retrieving it. There are a lot of intelligent browsers that is 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 would be easy and it will save time, due to the fact that 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, because the explicit display of the options of constructing a query. Ideas like manuals and the help menu in the menu bar of a UI would seem absolute comparing to the self-guided UI. Additional technique to make the UI smart is to user faceted browsing. The idea behind faceted browsing is to personalize the search and get more specific results by suggesting some filters. Faceted browsing is very related to ontology driven UI since both provide some information about the query while been constructed [5].

There are some systems, that are ontology driven UI, exist such as TAMBIS and SEWASIE. TAMBIS is a system that gather and analysis bioinformatics information from different sources through one interactive UI. While SEWASIE meant to access multiple sources of data and help user through out constructing the exact needed query.

The idea of ontology driven UI is not new. In this project, will try to build an application on the top of existing tool (the Manchester Pizza Finder) with new functionalities and enhancements. The new tool is called the Manchester Sushi Finder; since it is build mainly for sushi ontology that was previously developed by Ontology Engineering course unit. This does not mean the tool will run only sushi ontology, but it can run ontologies with similar structure, concept domain and have specific configurations.

1.2 Aims

The aim of this project is to investigate and demonstrate the benefits of using OWL ontologies and OWL API within ontology driven UI application as shown in the project page [6]. As well as, making the process of checking and testing ontology easier for students by uploading their ontologies, this will be shown by implementing a configurable and flexible UI. So that most of the configurations will lay in the ontology file, and the UI could browse other ontologies that contain some specific configurations as annotations. The application is called the Manchester Sushi Finder, where a user can construct queries to search for sushi based on included and excluded ingredients defined in conceptual model represented in ontology file.

1.3 Objectives

To achieve the aim of the project, the aim is divided into several of objectives. These objectives are:

- Gather project requirements.

- Increase the reusability of the UI by making it configurable to suite content of other conceptual models.
- Increase the usability of the UI by showing the languages available in the ontology with their percentage and switch between them.
- Increase the accessibility of the system by applying filters on the content of the conceptual model or/and on the result of the search query. By introducing the notion of filters and facets search to access more specific information.
- Increase the accuracy of the system, so users can only construct valid queries and they get the intended results. Making the UI driven by ontology and using the faceted browsing along with will increase the accuracy of what needed to be queried.
- Provide 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 tree, and list. Users have more 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 allow users to browse pizza toppings and construct queries to get certain kind of pizza. During constructing query, user chooses included and excluded toppings. As a result, pizza that matches specified criteria will be shown in the result window. This tool can run only one static pizza ontology.

The new tool (The Manchester Sushi Finder) can run 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 has the ability to upload different ontologies during the runtime. Filters decided in the ontology file as annotation and displayed in the tool if they are 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 would show languages if the ontology is labeled with different languages and would show also the percentage of the languages according to the ontology.

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

1.5 Structure of the Dissertation

To be written…

1.6 Conclusion

To be written…

2 BACKGROUND

2.1 OWL

OWL is a semantic web language that represents things about the world, group 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 [8]. It is able to represent explicit and implicit things [7]. The intention behind creating OWL is to be used not only by human but also by applications [9]. OWL can be access from machines because it is based on computational logic so that the machine using some software can reason over them [7]. There are two versions of OWL: OWL and OWL 2 [7]. OWL is a W3C recommendation since 2004, and then OWL 2 was published in 2009, followed with a second edition in 2012 [7, 10]. OWL 2 is just an extension and revision of the original OWL publish in 2004 [7]. 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 OWL working group [9] as scattered. This information could mean something for humans but not for machines. So, the semantic web gives explicit meaning for this information. As a result, integrating and processing the information would be easier for machines.

In 2012, OWL 2 has been introduced by OWL working group [11]. It is not different than OWL, it could be seen as an extension of OWL with some additional features. OWL 2 has several syntaxes and semantics; usually a developer needs only one syntax and one semantic to use OWL. Figure 1 shows the structure of OWL 2.

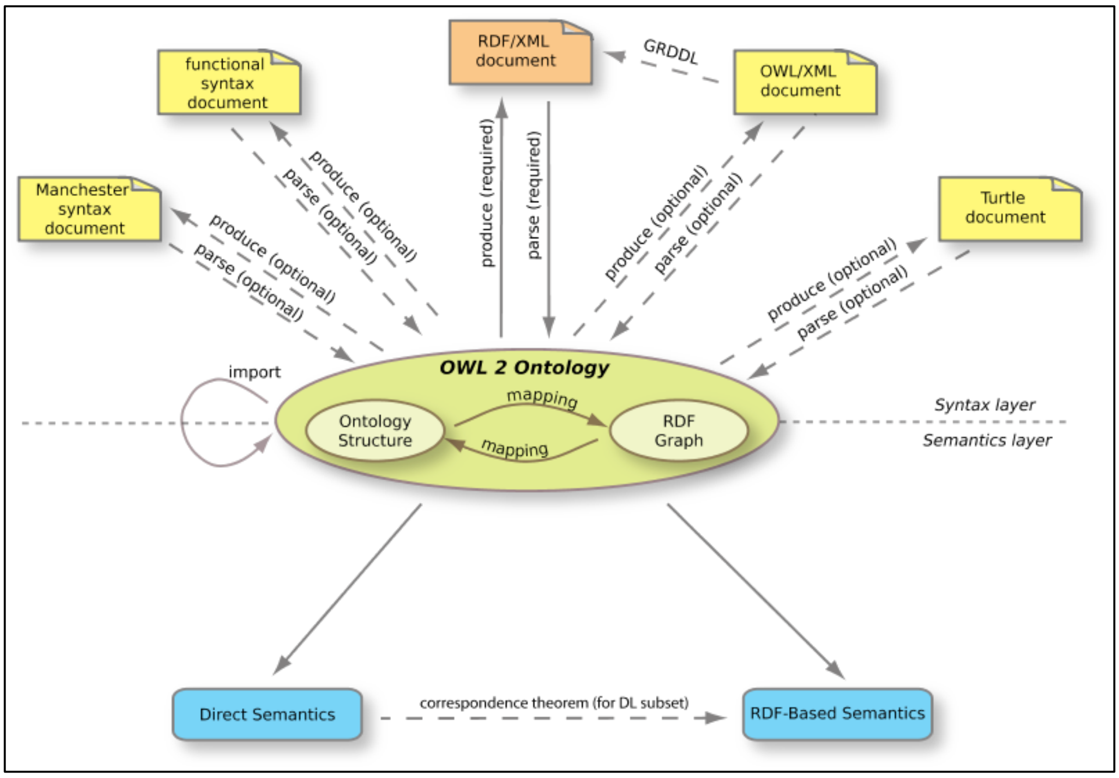


Figure 1: The structure of OWL 2 [11]

OWL files or documents are called ontologies, as described in [7]. The purpose of these ontologies is to make it easier of machine to access information in the web and preform reasoning on them. These ontologies can be put into the web or into a local computer depending on the need. One of the advantages of ontologies in the web is that they can be referenced from or reference to other ontologies. Ontologies can be placed in a local computer to be used locally.

2.1.1 OWLClasses

To be written…

2.1.2 OWLObjectProperties

To be written…

2.1.3 OWLAnnotationProperties

To be written…

2.2 OWL API

In Wikipedia [12], Application Programming Interface (API) is described as a set of protocols that make sure the software components interact with each other in the right way, and it could take many forms in different areas. It is used in the web as a set of Hypertext Transfer Protocol (HTTP). Also, it has heavy use as libraries of programming language. API is used in different forms such as libraries of programming languages. For example, Java APIs. In object-oriented languages like java, the API is a set of classes and methods to be accessed and used. Basic examples would be like using the inputting and outputting classes e.g. (BufferedReader and BufferedWriter classes in java). Since this project will be built using java-programming language, the API used is a java API which is called OWL API. OWL API is a set of classes and methods that facilitate the access to Web Ontology Language (OWL) ontologies. It creates objects that represent ontologies objects and manages the interactivity between them and any other program.

OWL API is described in [10] as an Application Programming Interface for the purpose of specifying how to interact with OWL Ontologies. OWL ontologies can be created, manipulated, and reasoned over using OWL API. It has been available since almost the same time of OWL. OWL API went through several revisions following the development of OWL. OWL API 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.

OWL API takes out the burden of parsing and serializing OWL ontologies from the developer back, since it has been taken care of in the implementation of it [10]. It has been implanted using java. OWL API comes also with some capabilities such as loading and saving ontologies.

OWL ontologies being accessed using OWL API only through OntologyManager interface [10]. OntologyManager interface manage all changes in ontology as seen in Figure 2 below shows UML diagram of how ontologies would be managed using OWL API.

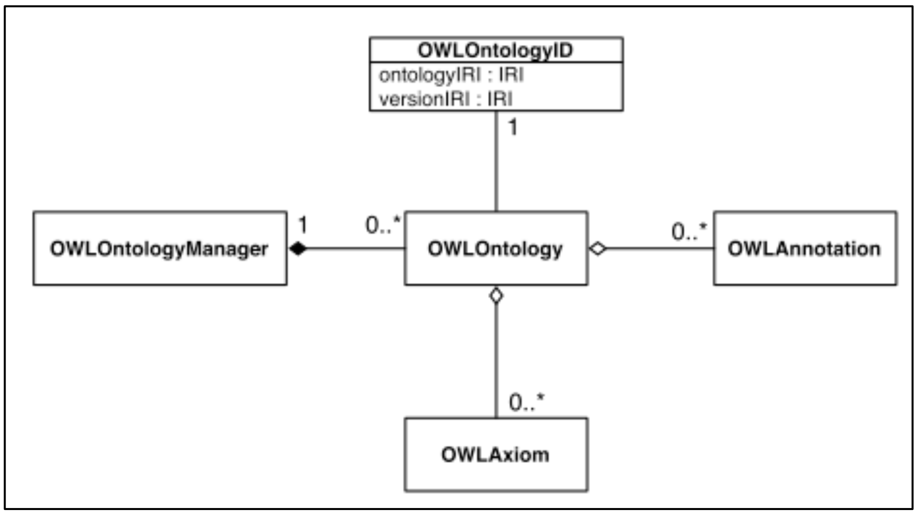


Figure 2: UML diagram showing the management of ontologies using OWL API [10]

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

As for query using OWL API, it does not offer much as a query mechanism [10]. Since, 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 retrieved from a source or multiple sources contains needed data. Every retrieval system needs some mechanisms to retrieval relevant needed information. Nowadays, there is more information probably resides on the cloud than it was in the recent years [13]. As the amount of data on the web grows dramatically, the need to find and retrieve relevant information becomes more important. Getting the wanted results is becoming more problematic because of the amount of the data on the web and the technique used. Usually, in a search engine the results range between relevant and irrelevant. It would be nice for a user to query for something and get the most relevant result that meets his/her needs.

There are different methods of retrieving data from the web. Keyword-based search method could be the oldest among the others.

Keyword-based search method use specific words call “Keywords” that are linked with database records [14]. It would be the default and the usual choice to use in search, since it has been used over long time. This method seems easy to use, as it is resemble natural language which could be understandable 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, keyword-based search method suffers 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 keyword-based search add more ambiguity, when the user want just to browse around to find out what is there or 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 by ontology-based information retrieval method. The growth in the ambiguity issue, would be solve be using multi-faceted search method which would guide the user during constructing the search query. So, the use of knowledge base and concept base would be more desirable than just providing arbitrary information. In addition, a sense of Artificial Intelligence is also felt since machines can make inferences based on some rules.

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

2.4 Ontology Based User Interface

In general [15], developing user interfaces hindered by the knowledge of the user. Letting the user known what can he ask for and constructing a meaningful search query using the user interface is the major issue. To remedy this issue, solutions have been proposed. Some of those solutions would be making every option in the user interface available to reduce the recall issue. Another solution would be writing manuals to the user to follow. These solutions might be providing more complexity and other problems. The former solution could overwhelm the user with all of the options available whether needed options or not. The latter solution could increase the load on the user to study and spend time on something that needed to be recalled eventually. Ontology driven user interface would be the most suited solution, since ontologies are based on conceptual model rather than just terms [3, 15]. This conceptual model gives a map for the user to follow upon constructing queries [15].

Ontology based user interface is defined by [3, 15], as a user interface that allows the user to construct and manipulate queries based on some domain concept stored in ontology. This domain concept drives the user interface, where there is no need for manuals or shove all available options in the user interface, since the ontology based one which should act as a guide for the user. It 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 out what he/she needs. The user does not have to any thing about the underlying conceptual knowledge. TAMBS give the illusion of retrieving from single source while it read from multiple sources and convert selected options to appropriate query languages that match sources'.

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

According to [15], there are two kind of concepts that the DL model support. 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 considered as defining new concepts definitions. Compositional concept can be formed using some services provided by DL. Reasoning about the concept definitions is done through the services provided by DL. These services are:

- **Satisfiability**: make sure that the concepts are consistent.
- **Subsumption**: create composite concepts definition from assertion made on the original concepts definition.
- **Classification**: make new classification hierarchy based on the subsumption relationship.
- **Retrieval**: retrieve any individual that is part of 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 annotations 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 interact based on. A tool was developed based on some animals ontology and annotations would be a good example of what could annotations do in term of user interface interactivity. This tool allow to brows for Apes' photos based on some annotated criteria such as the quality of the image and the environment where it been 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 major role in driving the user interface to make more flexible. This application is based on The Manchester Pizza Finder application, which will be elaborated on a separate section, except this one would have some enhancements.

2.5 Faceted Based Search

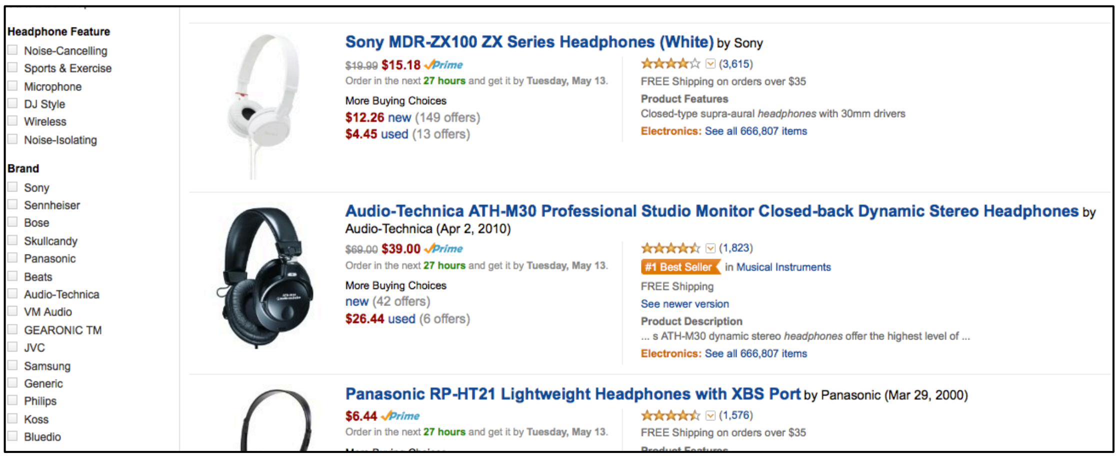
Ontology based user interface only provides the taxonomy and conceptual hierarchy and broad search capabilities. With the ontology conceptual hierarchy, user still can get broad search results. Transition from general to more specific results needs some kind of smart retrieval mechanism. Facet-based search along with ontology based user interface would guide the user toward constructing valid search queries and personalizing the search queries to suite the user needs. As using ontology in user interface development eliminates the recall element, using faceted-based search eliminates the ambiguity constructing the query and gets the intended results. So, ontology helps in returning relevant results. But faceted-based search assists in taking those relevant results and returning the most exact results. Ontogator is a system that combines the two methods: ontology driven interface and faceted based search [14]. The intent of Ontogator to search of particular image with specific annotations [14].

Ontology support faceted classification system as it provides a taxonomic order. Taxonomic order allows multiple way of viewing results rather that pre-determined one [14, 18]. Faceted search based on faceted classification system where information element are dimensions called facets [18]. Faceted search is an intelligent and efficient retrieval mechanism that allows the users to filter a collections of information based on some facets [18, 19]. Faceted search is known also as “faceted navigation” or “faceted navigation” [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 are faceted browsing and faceted search. Faceted browsing is constructing search queries by selecting some provided filters (Facets) [14]. In faceted browsing, user is provided with 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 in personalizing the search result to suit what is needed. Faceted search is used heavily in e-commerce websites like Amazon[1] or eBay[2]. Figure 3 shows the use of faceted search in Amazon.

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

[2] http://www.ebay.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 faceted search in e-commerce website (Amazon)

Faceted search could be applied in two ways, either unidirectional or bidirectional. In unidirectional way, either applies it beforehand on a collection of selection that is browsed to construct the query or on the result of a query so it can be refined more. In the bidirectional way, it is to apply it both beforehand and afterward running the search query. Both serve the same purpose which is to personalized the search and make it easy to suit the user's needs.

As in [14], facets the filtering process can be represented in more formal way. If C is selected category, where (C_i = 1, .., n) representing all categories selected, and C is also represents $(S_{i,1}, \dots, S_{i,k})$. Each category consists of subcategories. For example, in the pizza finder application user may choose Spicy Ingredient to get all pizza that are spicy but the category "Spicy Ingredient" could have subcategories like "Hot Pepper". Query in DL format would be:

$$(S_{1,1} \vee \dots \vee S_{1,k}) \wedge (S_{2,1} \vee \dots \vee S_{2,k}) \wedge \dots \wedge (S_{n,1} \vee \dots \vee S_{n,k})$$

Where $(S_{1,1} \vee \dots \vee S_{1,k})$ is the whole facet and S is the subcategory within that facet. In other words, the S's are subcategories of "Spicy Ingredient" which they might include Hot Pepper as an S, and the disjunction of the S's represents the category "Spicy Ingredient".

Facets bring benefits to applications, some of these benefits are [14]:

- **Guidance**: facets guide the user toward constructing valid search queries.
- **Transparency**: facets give the user idea of what is available and help in browsing the content.

- **Lucidity**: facets help in removing ambiguity caused by synonymous and homonymous query terms.
- **Relevance**: facets help with pre-compute partial results on selecting choices.

2.6 Ontology Visual Querying

The idea of visual querying is constructing a search query visually using drag and drop instead of the traditional way. Same idea can be applied on ontology-based interfaces that would be more powerful because of the benefits of the ontology-based applications. The interface would guide the user to build interactive meaningful queries by 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 there since almost the beginning of textual query languages. Almost all visual querying languages have two features in common. The two features are: (1) a model to represent the stated query and (2) a way to of constructing the query. Since visual querying languages invented to query from a data structure, it is only natural for its evolution to follow the development of data structure [5, 20]. A simple example of visual querying would in Microsoft Access.

A major benefit from ontology visual querying is the ease of querying, since user only drag and drop what needed to be queried. 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 concept can be defined either directly like defining class or indirectly like making inference of something. Therefore, creating query is the same as creating new concept such as the TAMBIS system and SEWASIE system [2, 3, 5]. Another advantage would be helping users, who 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 specific pizza based on some topping choices. User can include and exclude any toppings, and based on that the result would satisfy the query. The use of DL reasoner is present in this application, since it generated the filtering criteria (pizza topping) and their categories in the runtime. It is also make sure that the constructed queries and results are consistent. Based on the choices made the DL reasoner retrieve result that fulfill the input query. This application shows the use of ontologies, OWL API, and the power of building ontology-based interface, and faceted browsing.

2.8 Conclusion

In [4], the Manchester Pizza Finder 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. OWL API is an important component as any part of the application if not more important. OWL API manages all the communications between the application and the pizza ontology. OWL API is implemented using Java. So, Pizza finder is developed using Java. This makes the communication between the application and the ontology easier. OWL API have full access to the pizza ontology, it can preform operations on the ontology like make sure it consistent.

Pizza finder is considered ontology-based application, since it is the ontology that derives the interface and provides a conceptual hierarchy of a pizza domain. In ontology-based application, user does not need to recall keyword or know query language on querying for 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. User can browse around to figure out what specific toppings are needed. It based on the knowledge of pizza domain, not on keywords.

Pizza finder personalizes the query construction process by providing some filters (Facets). As a result, the results would suite the user needs. Pizza finder uses faceted-base in query building, so the user will be guided to construct only valid queries. User does not have to recall what keyword to search for something in the domain. User has the option in querying for broad or specific pizza in the domain based on chosen facet. Figure 4 shows the use of facets in pizza finder, use can query for example for vegetable topping pizzas or can query for more specific thing in vegetable topping category such as Tomato topping.

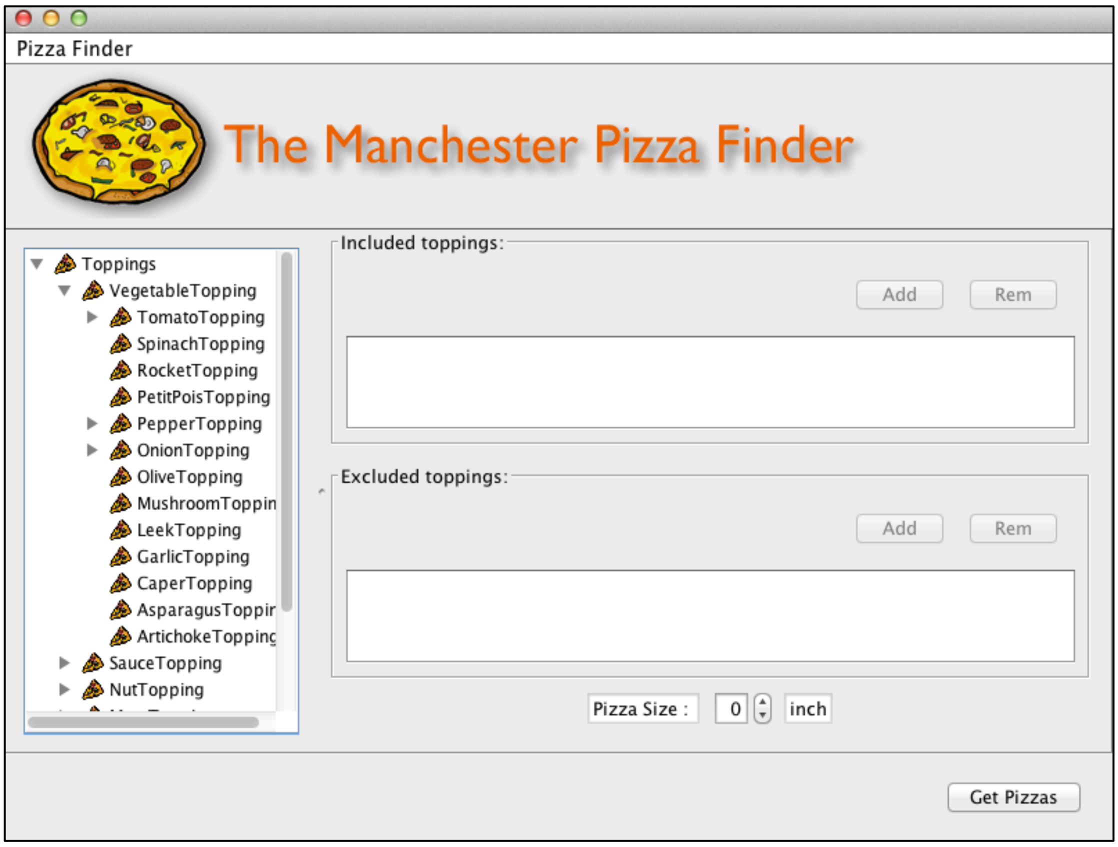


Figure 4: Facets are in the left hand side used to specify what is needed exactly as topping [4].

Sushi finder is considered an extension for pizza finder. Some enhancements of pizza finder are to be introduced. In Sushi finder, the application is more flexible than pizza one. The application should be able to work for any given ontology regardless of domain, but should follow some standard annotations. Another enrichment would be the use of annotations to drive the user interface sort of dynamically. Keeping the configurations in the ontology itself make it easier to for the user interface to be flexible. The whole application would be configurable in term of labels and languages being used. It configurable within itself, no need for changing configuration files. Facet search is introduced, user can filter the choosing options based on some configuration done in the ontology as annotations. In addition, user can apply filters on the query results.

3 RESEARCH METHODS

3.1 Research Methodology

As mentioned in the objectives section, is to develop a system that will:

1. Find specific sushi based on some ingredients choices. Include and exclude criteria for the ingredients are being used.
2. Students can run the system using an ontology they have developed for their coursework, in condition that they annotate their ontology in some way. The user interface is flexible since all labels can be configured.
3. Users can view the hierarchy of the ingredients in different views like tree and lists.
4. User can filter the hierarchy of the ingredients based on some facets, as well as, filtering the result of the search query. Due to the faceted search, search will be more personalized instead of broad and general one.

To achieve the objectives of this project, it 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 important role in software development process which is requirements gathering. The stakeholders of the project are three; the end user who will use the system, the system provider who will provide the tool to the end user and probably does some configurations to the tool, and ontology engineer. Since meeting all of these stakeholders hard, 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 develop. Meetings have been setup to discuss the requirements (what exactly should be done?). Some conversations were held with my fallow students, who attended with me the ontology engineering for semantic web course, regarding if they had this system before how it would help them and what functionalities would be needed. All of those meetings and discussions provided more details and helped in understanding some of the requirements. Some were understood later on.

3.1.2 Background Study Stage

In the second stage, background research and survey of relevant literature was carried out along with exploring techniques to be used in the project. This has been done using journals, articles, publications and existing systems with similar functionalities. A fair amount of time spent reviewing different literatures trying to understand different aspect of the project's requirements. Reviewing relevant literature really helped in not only having a wider knowledge of the problem domain but it helped also in understanding the requirements of the project. As a result, I have good understanding of the project and approaches taken to handle such systems and it helped me in splitting the project into small tasks.

In this stage, background study is done on OWL ontology, OWL API, ontology-based systems, faceted-based search system, ontology visual querying, and finally study the Manchester Pizza finder a system that I will build my project on.

3.1.3 Development Stage

In the development stage, first decide in the development tool that is java. Then, refresh myself with java specially swing components, and OWL API. Implementation started in early stage. As the strategy is to divide the development of the system into developing the main functionalities separately then combine them. Although that some functionalities were developed separately from the application itself, the official start of this stage will be after the second semester's exams. After exams, checking that the functionalities are working probably will be done and combine them.

The development is considered as enhancements of the Manchester Pizza Finder. They involve reading the configurations from the ontology file and act accordingly for more flexibility. Also, different view of the content of the conceptual model and the result of the search query will be considered. As well as, adding a functionality to filter the content of the model based on some criteria saved in the ontology to personalize the search even further to the user.

3.1.4 Testing Stage

In this stage, testing will be conducted on the application according to some scenarios that are predefined. These scenarios are called users stories which will be elaborated on later on in the report. Since the strategy of doing the project is to develop functionalities alone then combine them, testing is carried out during the development stage on functionalities separately and 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 his ontology to work with the application. The application would act as manual, so there will be no need for a guide for the system. The next step will be finishing the application and finalized 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:

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

First milestone was already completed, since initial report was submitted successfully in March. However, submission of second milestone in time was not so successful. So the original plan was altered. The new deadline will be in June 6. The last three milestones would be worked on in parallel due to time restriction. The final product is expected to finish beginning of August. Finally, the submission of the dissertation will be in the first week of September. Gantt chart is included in the next section.

3.2 Project Deliverables

At the end of this project the benefits of using sematic web within applications will be shown. The application will be flexible, as it will run ontology with specific annotations. In addition, it will be fully configurable. It will allow users to query for specific sushi based on some ingredients. There are three main deliverables of this project:

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

3.3 Project Evaluation Plan

In order to be able to evaluate the whole project, I need some measures to evaluate the project against. Some measures have been recognized to evaluate how good the system is and more importantly measure if the project is consider a success or not. In nutshell measuring the success of the objectives. These measures are:

- **Where 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 done within its allocated time or not.

- **Is the system accessible?**

  We can think of the system in two ways: one as a project for MSc program that should be accessible for students and lecturers who teach ontology engineering for semantic web, and as real-world application (restaurant menu). Based on that users who should access the system differ. As for the first case, since the application is desktop application and implemented on java this will assure 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?**

  As the system designed for end users regardless of what kind of end user students or real-world users, they can determine the usability of the system. There are two concepts here determining the usability and reusability of the system. Therefore, both of them need to be checked as part of project evaluation.

  - **Usability**: as an end user, one can ask several questions that will assure the system is usable. Some of these questions are:

  1. How easy to use the system?

     The idea behind using 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 spent to figure out the system?

     This question can be answered after submitting the project.

  3. Does it require experts to use the system?

The system is designed for students who take ontology engineering for sematic web. So, some degree of expertise is needed to configure the ontology to work with the application.

4. How easy to administer the system?

   Since most of the configurations are saved within the ontology, administering the system would be a trivial task.

- **Reusability**: the system is reusable in sense of running different ontologies. This is another goal of the project, making it more flexible. The important question here is how easy it is reuse the system (running different ontologies).

- **Is the system easily configurable?**

  As most of the configurations saved in the ontology file, it would be easy to configure the application interface. Ontology developer is the only stakeholder who has to deal with the configurations which are annotations in the ontology file. They are easy to write, as the ontology developer needs to follow some instructions provided with the application. The user interface should be configured automatically using annotations in the owl file.

- **Is the result of the search query narrowed down?**

  Stakeholders such as students or real world users, like restaurant customers, will have direct contact with feature. This 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 used faceted search method.

These are some questions that can help assessing the project. Most of them can be answered only after using the application by stakeholders. So, evaluation will help in a second version or the final product if it a prototype.

3.4 Project Tools

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

3.5 Conclusion

To be written …

4 SYSTEM DESIGN

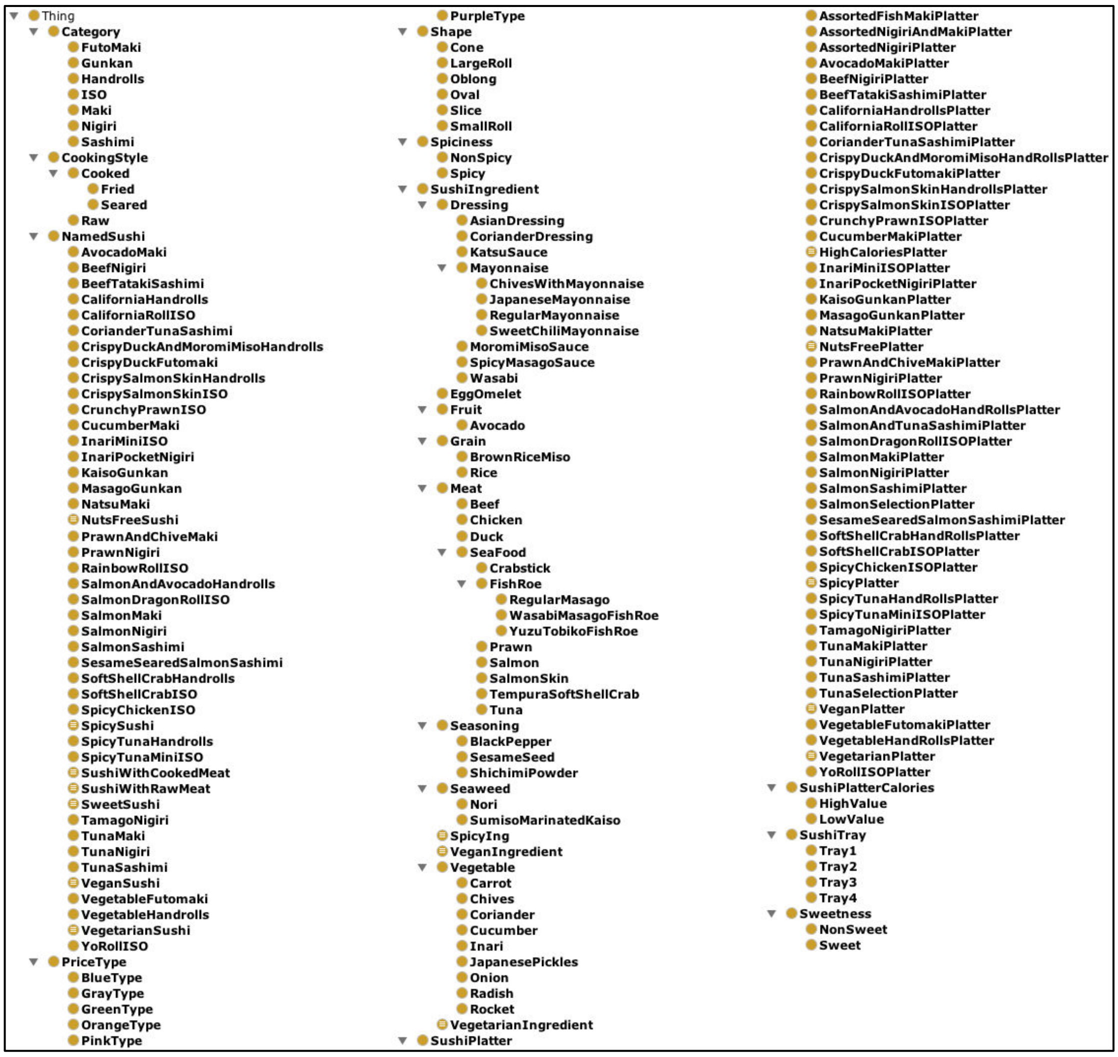
This chapter provides details about the design process of the ontology used with the tool mentioned in 1.1, and the user stories mentioned in 3.2 as well as the design process of the tool (the user interface). It starts with the design of sushi ontology which is based on sushi menu. Then, it moves to design of user stories and how they have been used to specify system functionalities. Finally, it details the design of the user interface.

4.1 Sushi Ontology Design

In this section, details of designing Sushi Ontology are provided. It shows the class hierarchy, object properties, and annotation properties. Then, it explains semantics used. Finally, it provides some examples of DL queries that are applied on the ontology.

This ontology was developed using Protégé.

4.1.1 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 5: Class Hierarchy of the sushi ontology

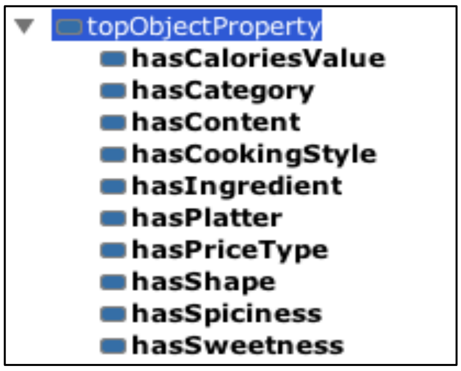
Figure 5 demonstrates the classes' hierarchy of the sushi ontology. The Thing class is default super class of all other classes. The subclasses of Thing class describe sushi concept domain. Notice that there are two main classes: NamedSushi and SushiIngredient under the generic class Thing. NamedSushi class describes different names of sushi such as AvocadoMaki and BeefNigiri. While SushiIngredient class describe the different ingredients of sushi.

SushiIngredient class classifies the ingredients of sushi from general to specific ingredients. Notice the meat class is general class, while Beef and Duck class demonstrate specific concepts. Seafood class specifies the ingredients furthermore to describe the seafood concepts domain. Notice also equivalent ingredients classes, that 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, spicy class could be use to specify that I want only spicy sushi.

4.1.2 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 6: Object properties of the sushi ontology

Figure 6 illustrates the object properties used within the ontology. Notice there is a default generic object property exists here also. These object properties represent the relations between the classes. hasIngredient object property play an important role since it connects NamedSushi class with SushiIngredients class. The domain of hasIngredients NamedSushi and the range is SushiIngredients. Any specified sushi query use hasIngredients property in constructing it.

The properties hasSpicness and hasSweetness represent test of sushi ingredients. They also represent the relations of SushiIngredients and wither 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.

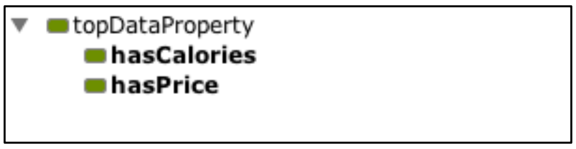


Figure 7: Data properties of the sushi ontology

Figure 7 illustrates the data properties used in the ontology. There are two properties hasCalories that link specific sushi platter and its caloric value. In the other hand, hasPrice link specific sushi platter with its price value. The caloric value and price value both are static value entered by the ontology developer.

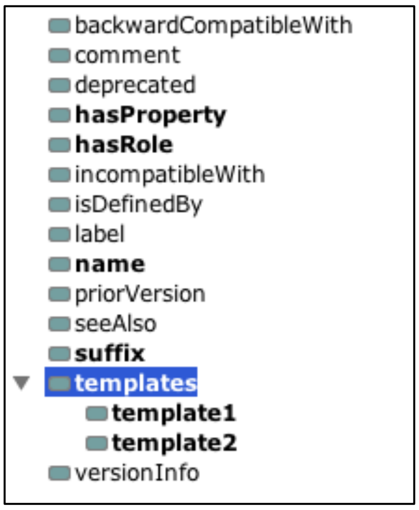


Figure 8: Annotations properties of the sushi ontology

Figure 8 shows the annotations properties that used to annotate the ontology. Annotation properties used to attached metadata to different part of the ontology. Annotation properties are important in this project, since they are being used in specifying filters, facets, and even in driving the UI. User can define new annotation properties such as hasRole, hasProperty. hasRole property is used in identifying the role played by as certain class such as a role of a filter, facet, or ingredient class. suffix property is used to state the suffix used in the class hierarchy if any.

4.1.3 Expressing Semantics

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

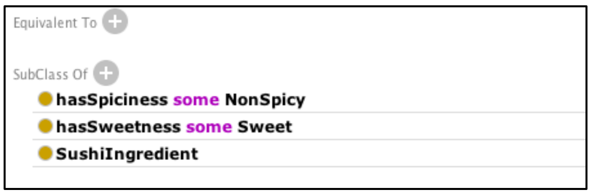

```
Equivalent To
SubClass Of
  hasSpiciness some NonSpicy
  hasSweetness some Sweet
  SushiIngredient
```

Figure 9: Semantics of EggOmlete class

Figure 9 shows the semantics of EggOmlete under SushiIngredient class. This semantics state the taste of EggOmlete is NonSpicy and it is Sweet by using the object property hasSpiciness and hasSweetness.

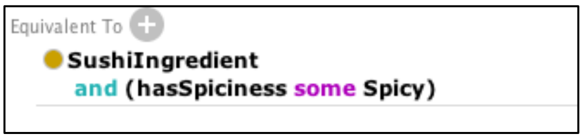

```
Equivalent To
  SushiIngredient
   and (hasSpiciness some Spicy)
```

Figure 10:Semantics of SpicyIng class

Figure 10 states the semantics of SpicyIng class. The meaning of this semantics is any SushiIngredinet that is Spicy. The sweet characteristic could be able here also using hasSweetness property and Sweet class to indicate Sweet class.

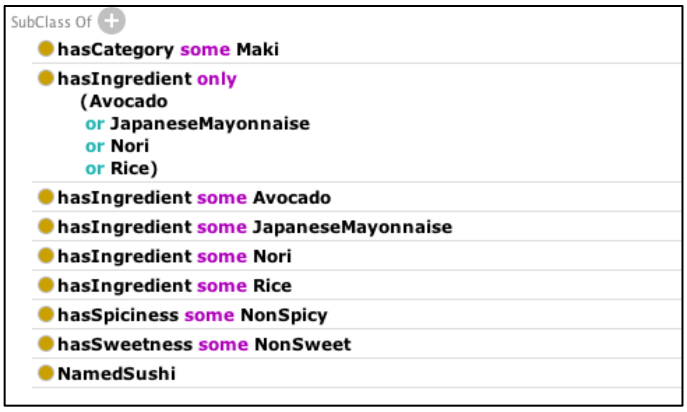

```
SubClass Of
  hasCategory some Maki
  hasIngredient only
      (Avocado
       or JapaneseMayonnaise
       or Nori
       or Rice)
  hasIngredient some Avocado
  hasIngredient some JapaneseMayonnaise
  hasIngredient some Nori
  hasIngredient some Rice
  hasSpiciness some NonSpicy
  hasSweetness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

Figure 11: Semantics of AvocadoMaki class

Figure 11 illustrates the semantics of a NamedSushi class AvocadoMaki. The first one specifies a suitable category for AvocadoMaki that is Maki Category. hasIngredient property is used to specify the ingredients of AvocadoMaki. Notice it uses some to include the ingredients and use only to state that only these ingredients and nothing else is included. 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 12, three things need to be considered for the tool to run different ontologies. BaseClass, IngredientClass, and the Property used to represent the relation between BaseClass and IngredientClass. Figures Figure 13, Figure 14, and Figure 15 illustrate the semantics in details.

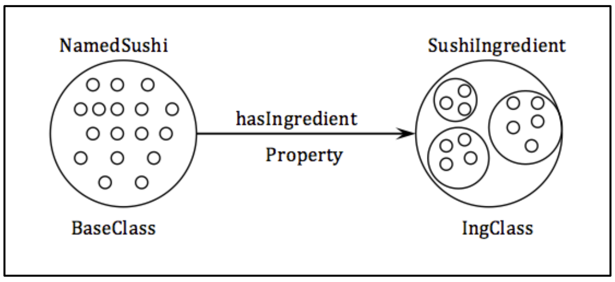
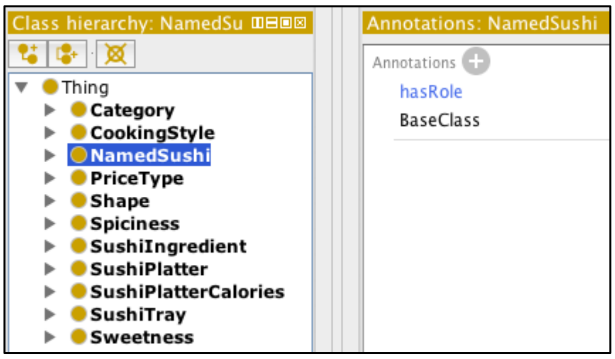


Figure 12: Basic annotations diagram of sushi ontology

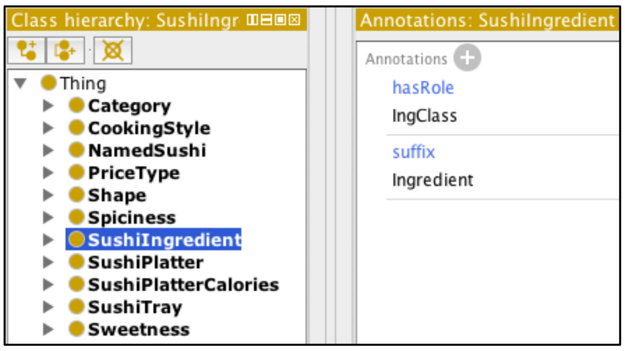


Thing
- Category
- CookingStyle
- NamedSushi
- PriceType
- Shape
- Spiciness
- SushiIngredient
- SushiPlatter
- SushiPlatterCalories
- SushiTray
- Sweetness

hasRole
BaseClass

Figure 13: hasRole annotation used in NamedSushi class

Figure 13 shows that NamedSushi class use hasRole annotation property to indicate this is the BaseClass by using constant value "BaseClass".



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

hasRole
IngClass

suffix
Ingredient

Figure 14: hasRole annotation used in SushiIngredient class

hasRole annotation property is being used in different locations within the ontology. Here in Figure 14, SushiIngredient Class is annotated using hasRole annotation property with the constant value "IngClass" to indicate that SushiIngredient class has the role of the Ingredient class.

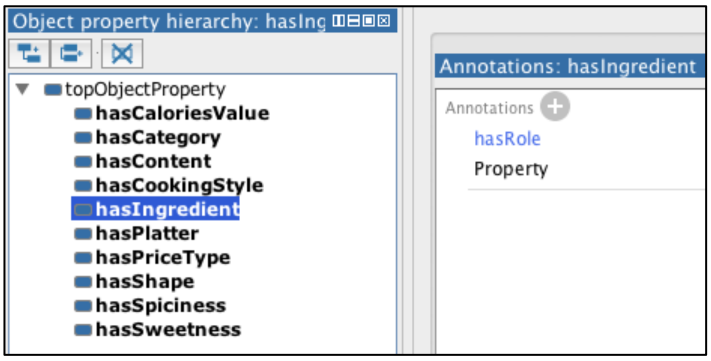


Figure 15: The use of hasRole annotation property in hasIngredient object property

Figure 15 shows the use of hasRole annotation property in hasIngredient object property. It means hasIngredient object property plays the role of the property that describes the relation the ingredient class and the base class.

hasRole is used also in determining filters and facets. Filters need only to be indicated by the hasRole annotations property with constant value "filter" for the tool to know it is a filter. Figure 16 shows that VeganIngredient class is a filter.

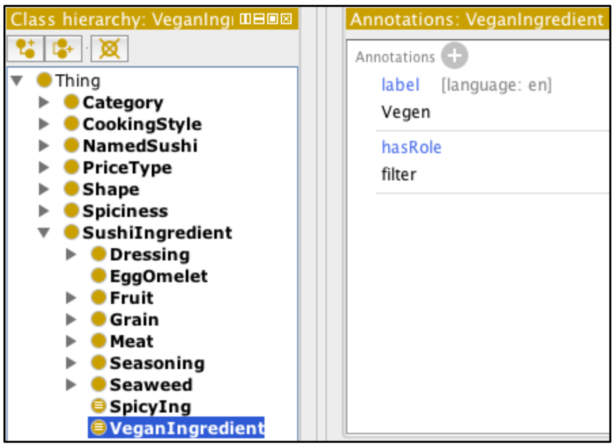


Figure 16: hasRole annotation property used to determine VeganIngredient as a filter

Defining facets require more than using hasRole annotation property, since there is need to be an object property that describe the relation between the ingredient sushi class and the its characteristic. For example a sweet ingredient and sweet class shown in Figure 5 need to have some kind of link between them. Figure 17 shows the use of two annotation properties hasRole to define the role of Sweet class as facet and hasProperty to specify the object property used to like the sweet class with ingredient classes. Notice here the value of annotation property hasProperty is IRI of hasSweetness object property shown in Figure 6.

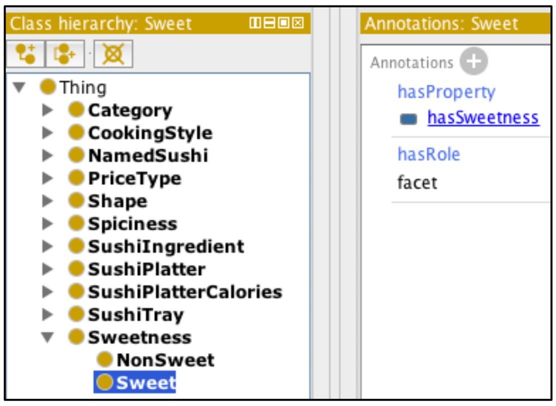


Figure 17: Facets are determined in the ontology

4.2 User Stories

This section demonstrates the functionalities of the tool in details. User story approach is used to determine the functionalities. User stories act as a guide to specify needed functionalities in clear way. All the user stories follow the idea of the "3 C's": The card, the conversation, and the confirmation. Where, the card is the general idea of the functionality, the conversation is the role of different stakeholders and what require from them, and the confirmation is the details of the functionality. These user stories are demonstrated on the below tables:

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

The card

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

Story Narrative: Run 1

As end user, **I want** to run the tool, **so that** I can use it.

As tool provider, **I want** upload subject ontology to the tool, **so that** end can use it.

As ontology developer, **I want** to annotate subject ontology, **so that** it can be running by the tool.

The conversation

As 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 tool provider, **I want** upload subject ontology to the tool, **so that** end can use it.

- Tool runs annotated ontologies.
- Tool does not run annotated ontologies.

As ontology developer, **I want** to annotate subject ontology, **so that** it can be running by the tool.

- Ontology is annotated with basic annotations to run it by the tool.

The confirmation

As 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 tool provider, **I want** upload subject ontology to the tool, **so that** end can use it.

1. Click on the tool icon to run it.
2. Brows for ontology.
3. Brows for tool icon.
4. Brows for tool logo.
5. Click on OK to upload the files and configure the tool.

Table 2: Run 1 user story

The card

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

Story Narrative: Run 2

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

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

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

The conversation

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

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

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

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

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

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

The confirmation

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

1. Click on file menu.
2. Click on import new ontology.
3. Browse for the new ontology file.
4. Click OK.
5. Tool prompts the user if the same ontology is been used and if user wishes to

overwrite it. 6. Application refreshed with new ontology. **As** tool provider, **I want** load different ontologies during runtime, **so that** end can use them. 1. Click on file menu. 2. Click on import new ontology. 3. Browse for the new ontology file. 4. Click OK. 5. Tool prompts the user if the same ontology is been used and if user wishes to overwrite it. 6. Application refreshed with new ontology. **As** ontology developer, **I want** to annotate subject ontologies, **so that** they can be running by the tool. 1. Annotate the ontologies with hasRole object property. 2. Specify the base class using hasRole property with the constant value “BaseClass”. 3. Specify the ingredient class using hasRole property with the constant value “IngClass”. 4. Specify the object property liking base class with ingredient class using hasRole property with the constant value “Property”.

Table 3: Run 2 user story

The card
As a [user], I want [functionality], so that [value]
Story Narrative: View
As end user, **I want** view ingredient in tree and list views, **so that** I can browse and query for results using different views.
The conversation
As end user, **I want** view ingredient in tree and list views, **so that** I can browse and query for results using different views. • Tool provides two views of the content “ingredients”: tree and list view. • Tool allows switching between views during runtime.

- Tool allows selecting contents from the two views..

The confirmation

As end user, **I want** view ingredient in tree and list views, **so that** I can browse and query for results using different views.

1. Click on view menu on the menu bar.
2. Click on tree or list view.
3. Application refreshed with new view.

Table 4: View user story

The card

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

Story Narrative: Language

As 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.

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

The conversation

As 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.

- Tool shows available languages used in ontology file.
- Tool shows the percentage of a language usage in ontology file.
- Tool switches languages upon choosing a language.

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

- Label classes in the ontology with different languages.

The confirmation

As 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 configuration menu on the menu bar.
2. View available languages drop down menu.
3. Click on wanted language.
4. Application refreshed with new language.

As ontology developer**, I want to** label the ontology classes with different languages, **so that** user can choose appropriate language. 1. Open ontology with Protégé. 2. Annotate classes with label annotation property. 3. Choose language for the label. 4. Write down the label value as constant value. 5. Click Ok. 6. Save ontology file.

Table 5: Language user story

The card As a [user], I want [functionality], so that [value]
Story Narrative: Tree View Filter
As end user, **I want** to filter the ingredient tree, **so that** I can browse and query for results easily. **As** ontology developer**, I want to** specify filters on the ontology, **so that** user can filter ingredient tree.
The conversation
As end user, **I want** to filter the ingredient tree, **so that** I can browse and query for results easily. • Tool shows filter options. • Tool filter ingredients by highlighting them and disable the rest. • Tool does not show filters if they are not specified in the ontology. **As** ontology developer**, I want to** specify filters on the ontology, **so that** user can filter ingredient tree. • Annotate ingredients classes to be used as filters.
The confirmation
As end user, **I want** to filter the ingredient tree, **so that** I can browse and query for results easily. 1. On 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** ontology developer**, I want to** specify filters on the ontology, **so that** user can filter ingredient tree. 1. Open ontology with Protégé. 2. Annotate wanted ingredient classes with hasRole annotation property. 3. Write down the “filter” value as constant value. 4. Click Ok. 5. Save ontology file.

Table 6: Tree View Filter user story

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

Table 7: List View Filter user story

The card As a [user], I want [functionality], so that [value]
Story Narrative: Query
As end user, **I want** to query for sushi using the tool, **so that** I got result easily.

The conversation
As end user, **I want** to query for sushi using the tool, **so that** I got result easily. • Tool shows ingredients and two input list for included and excluded ingredients. • Tool shows the result that sushi ingredients match included and excluded ingredients.
The confirmation
As end user, **I want** to query for sushi using the tool, **so that** I got result easily. 1. On list or tree view, select ingredient. 2. Click on the add button to add selected ingredient to included or excluded lists. 3. Click on the rem button to remove selected ingredient to included or excluded lists. 4. Click on run button. 5. Tool shows the results. 6. Click on back button to go back and form another query.

Table 8: Query user story

The card As a [user], I want [functionality], so that [value]
Story Narrative: Facet
As end user, **I want** to filter the result of a query, **so that** I can narrow is down and find wanted result quickly and easily. **As** ontology developer**, I want to** specify facets on the ontology, **so that** user can filter search query result.
The conversation
As end user, **I want** to filter the result of a query, **so that** I can narrow is down and find wanted result quickly and easily. • Tool shows facets options upon getting the result. • Tool filter result. • Tool shows new result upon applying the facets. • Different facets have different filtered results. **As** ontology developer**, I want to** specify facets on the ontology, **so that** user can filter search query result.

- Annotate value partition classes to be used as facets.

The confirmation

As end user, **I want** to filter the result of a query, **so that** I can narrow is down and find wanted result quickly and easily.

1. Upon getting the result, choose one of the radio buttons in the facet section.
2. Result view refreshed with the new filtered result.
3. Choose All Result radio button to get the original result.

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

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

Table 9: Facet user story

4.3 User Interface Design

This section details the design process of the tool (UI). It also shows the interaction design with ontology like the one mentioned in 4.1. The main ideas behind the design are developing a UI that is driven by ontology, the ability to run different ontologies, using different languages used in the ontology, as well as using filters on the browsed content and facets on the search result. To details those furthermore, this section starts with first idea that is building ontology driven UI. Then it moves to the second one the flexibility of using different ontologies with the UI. After that, using different languages available in the ontology. In addition, it elaborates on applying filters on the shown content and applying facets on the search result. Finally, it shad the light on some UI enhancements like importing ontologies during run time or having different views of the browsed contents.

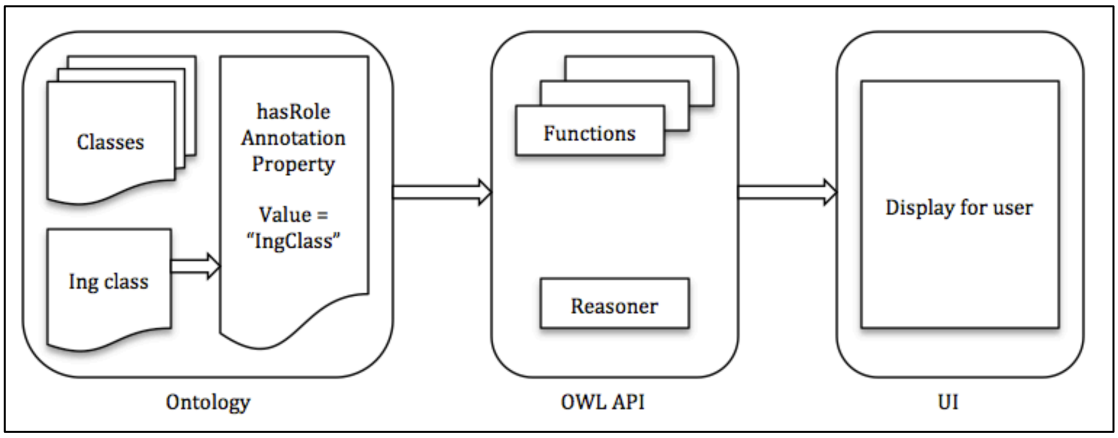


Figure 18: Ontology Driven UI diagram

Figure 18 demonstrates how the UI is driven by ontology. It shows three important parts: the ontology where it represents food domain concept, OWL API that link the ontology with UI and preform operations on the ontology programmatically, and the tool (UI) that displays content, result, and controls functionalities. The content (knowledge of concept domain) is preserved in ontology file. Ontology could have many classes like the one mentioned in 4.1, Ingredient class is the important one since it is displayed for the user and it constructs the search query. To start with opening the ontology programmatically, its location is needed like any other file to be open via code. So user is asked for the ontology location as it mentioned in Table 2: Run 1 user story. After getting the ontology location and open it, Ingredient class is retrieved by using a combination of two things: annotating the Ingredient class before hand using hasRole annotation property with the value "IngClass" as it mentioned in Table 3: Run 2 user story and using OWL API to retrieve the class with the annotation specified. An algorithm is implemented to search for the class that uses "hasRole" annotation property and the value "IngClass" by using OWL API to access the ontology, make sure it consistent, and get the ingredient class. The Ingredient class gets rendered using in a tree and a list OWL API; and displayed, so 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 reduced the amount of recall of remembering keywords, and will has the clarity for constructing meaningful search query.

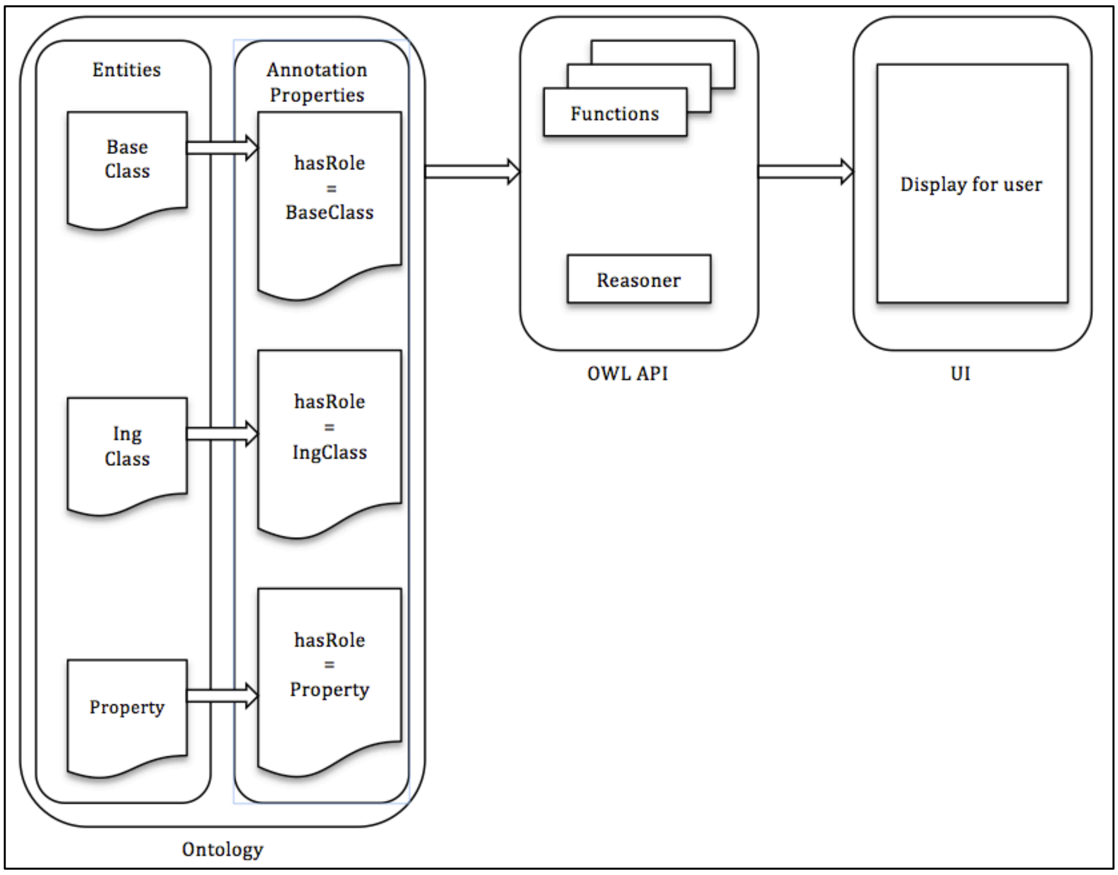


Figure 19: Three main things to run different ontologies

While Figure 18 shows how to display the ingredient class to the user by annotating it using hasRole annotation property. Figure 19 illustrates the idea of running different ontologies using the UI. The UI needs more information than just an ingredient class to preform queries and gets results. As it mentioned in Table 3: Run 2 user story and it is shown in Figure 19, it needs three components to execute queries: the base class, the ingredient class, and the object property that link the previous two classes. An example of a DL search query would be like: [BaseClass] and ([Property] some [IngClass]). Where BaseClass is the named thing that we want to query for, IngClass the ingredients of BaseClass, and Property is the like between the two. To make ontologies work with the UI; base class, ingredient class, and the property are needed to be annotated with hasRole annotation property. The value of hasRole for base class is “BaseClass”, ingredient class is “IngClass”, and property is “Property”. The tool searches for the classes and object properties, which are annotated with hasRole and have the values “BaseClass”, “IngClass”, and “Property”, using OWL API. Then, UI displays the ingredient class to the user to select from the included and excluded ingredients. Then, the tool form different queries depending on the selected included and excluded ingredients. These queries would be like:

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 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 it stated in Table 5: Language user story, displaying languages depend in the ontology itself if it is labeled with more than language or not. As an assumption the default language is English. The tool looks for labels with language associated with them and calculates the percentage of the language to the whole ontology. The UI displays available languages in 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 of there is no language associated with class labels, the name of the ontology class is displayed.

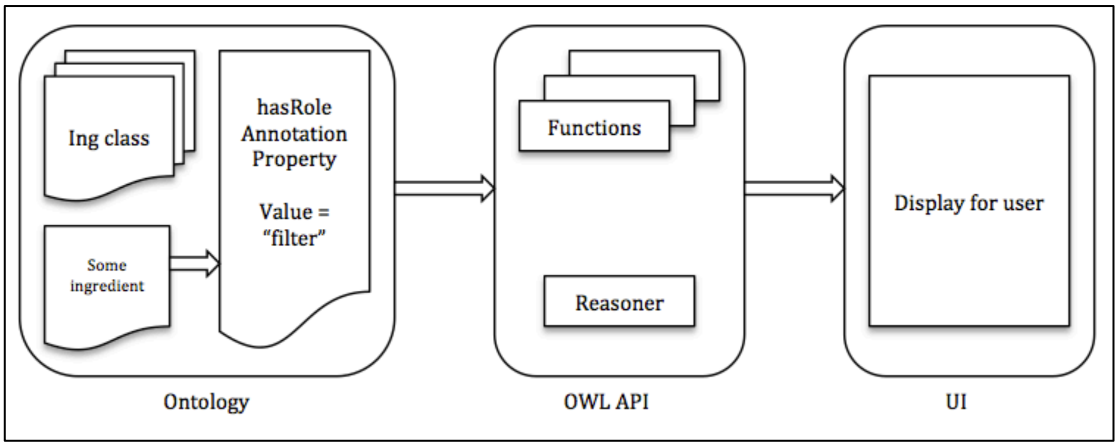


Figure 20: Filter annotation

Figure 20 shows the design of adding filters to the tree view within the tool. As stated in Table 6: Tree View Filter user story, if the ontology contains filter annotations associated with ingredient class then the filter will appear in UI otherwise it won't appear. Filtering using the tree's view depends on the ontology annotations. hasRole annotation is used here also, where an ingredient class is annotated with the value "filter". Filters usually are class expressions like any class with spicy ingredient. After annotated ingredient classes, the tool looks for classes with hasRole annotation property and the value of "filter". Then, the tool shows to the user specified filters. Finally, upon choosing a filter the ingredients get rendered. The ingredient classes that match filtering criteria are highlighted with yellow color and the rest are disabled and cannot select from. As mentioned in Table 7: List View Filter user story, filtering in the list view is completely different than the one in the tree list. Filtering in the list view is text based filtering, where is the user input a text in the input text filed. 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.

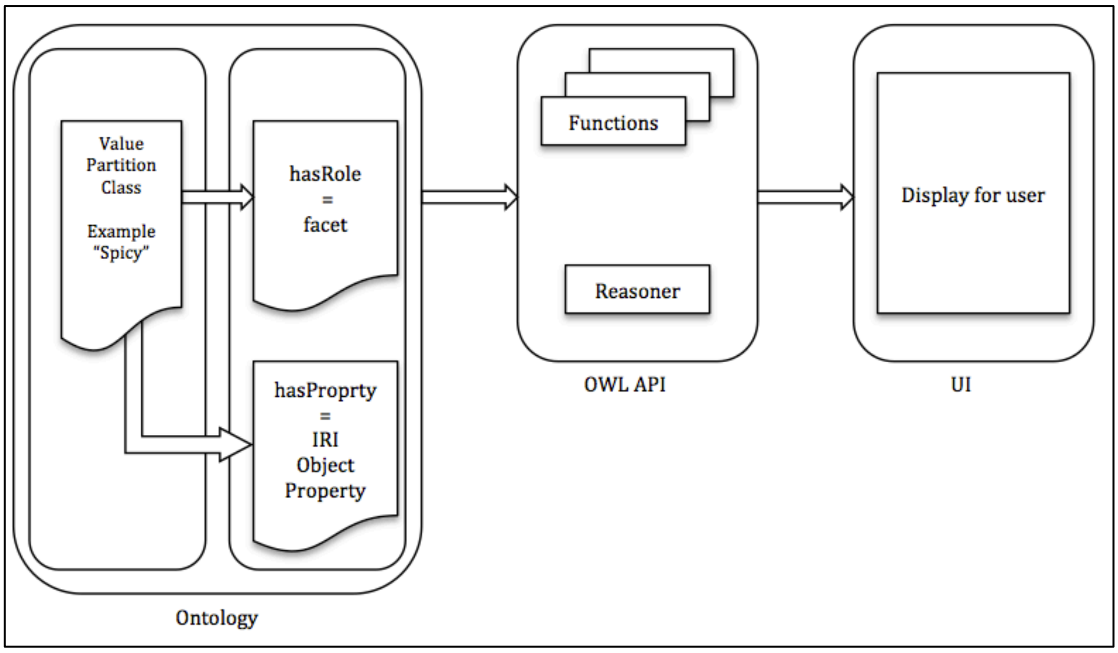


Figure 21: Facet design in the ontology

Facets require more work in the annotations than filters. As Figure 21 shows, the value partition class such as Spicy class needs to be annotated using two annotation properties. Firstly, we use hasRole annotation property with the constant value "facet" to identify that this class is been used as facet. Secondly, we use hasProperty annotation property with IRI value of the object property that is used to link the value partition class and an ingredient class. The tool then search for classes annotated with hasRole annotation property with a constant value of "facet". After finding the facet class, the tool gets the object property used to like ingredients classes with the facet. Finally, when the user queries for something and get the result the UI shows facets that are specified in the ontology; the result changes based on the selected facet.

4.4 Conclusion

This chapter presented the design process of the important aspects of the project. It started with the design of the sushi ontology that was used mainly with the tool. Then it moves to the design of different user stories, which have the role of determining the functionalities of the System. Finally, it shows the design process of the UI based on the ontology that was in used, and the user stories that specify the functionalities of the UI.

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

5 IMPLEMENTATION

This chapter talks about 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 contains the details of the Manchester Sushi Finder implementation process. Firstly, it provides details about the technology and software process model that are used during the implementation. Then, it discusses the new functionalities and provides snapshots of this tool that built on the top of the “Manchester Pizza Finder”. Finally, it elaborates on some limitations that were faced during the implementation phase.

5.1.1 Java

The Manchester Sushi Finder is implemented using java programming language. Since the this tool is built on the top of existing tool “The Manchester Pizza finder” that was implemented using java programming language. Another reason to use jave programming language would be the use of OWL API that is a java interface. So, continuing implementing in java eases the process of the compatibility with the old tool and the accessibility of the old basic functionalities.

[21, 22] Characterizes java as two things in one, so it is a programming language and a platform. Java programming language is considered as high-level programming language. It has been described as simple, object oriented, portable, and robust. the program is written as plan text then it gets compiled to bytecodes that is understandable be Java Virtual Machine (JVM). Finally, it is run using JVM that is available in most of the operating systems. Platforms usually have consist of software part that lay on the top of the second part which is the hardware part. Java is considered as a platform since it has the software part without the hardware part. Since JVM is available in most to the operating systems, java can be run in different platforms. Moreover, it is simpler to learn and user and it is object oriented that supports the reusability of modules. Although java has a lot of advantages, it has also a major disadvantage that is the slowness because of the platform independent environment.

5.1.2 Maven

The Manchester Pizza finder is implemented using java, and built using maven. Since the Manchester Sushi finder is built on the top of its ancestor, it is only natural to use maven technology for the build process.

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

5.1.3 OWL API

OWL API has been used to link ontologies with tool (UI) that is built. As it was mentioned in section 2.2 OWL API is used to facilitate creating, manipulating, and reason on ontologies used with tool. Using OWL API is an advantage since the project is developed using java and OWL API is a java libraries. In addition, the previous version of the finder The “Manchester Pizza finder” used OWL API. Since the new finder the “Manchester Sushi finder” is built on top of the earlier one, using it in the newer version ease the use of basic old functionalities that have implemented before.

It has contributed a lot in this project. The main contributions are: reading ontologies, searching for annotations, rendering ontologies with different languages, and query for result. It facilitates reading ontologies as they are being uploaded to the tool. It provides the tool with ingredient class, switches between languages, and gets the results based on some criteria.

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

5.1.4 Using Ontology Annotations

Annotation properties is one of the main pillars of this project. [24] defines annotations as metadata that can be associated to different part of the ontology. The previous version of the UI the "Manchester Pizza finder" stores the configurations about the ontology in external file. So, why not store them in the ontology file itself instead.

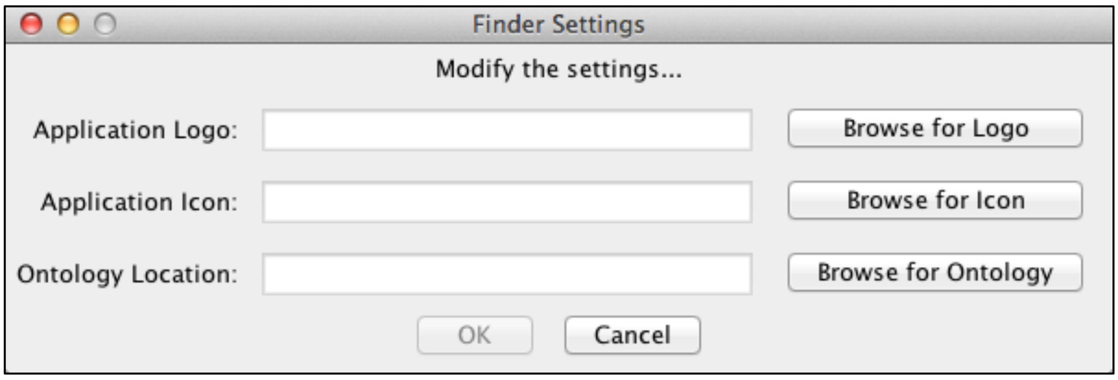
Annotations have been used by the new finder to store configurations inside the ontology. As a result, every ontology has those standard annotations stored in them will be run by the UI. Annotations determine the ingredient class to be displayed to the user as Figure 18 shows. According to Figure 19, they 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 hasRole annotation property as it was mentioned earlier in Figure 16 and Figure 17. Almost everything in the ontology that been used by the UI is specified by using hasRole annotation property.

5.1.5 Iterative and Incremental Development

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 iterative design and incremental build model, five iterations are involved in the process of the development. Each iteration responsible for a major functionality that 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 run ontologies containing these configurations mentioned in Figure 19. In the second iteration, filters functionality was implemented. New annotations were added to the ontology specifying filters, then new algorithms to get filters and display them were coded. In the third iteration, sushi ontology was modified by adding new annotations to determine the facets to be applied on the search result, then tool modified to find these facet and display them to the user plus adding their functionality. Fourth iteration is for displaying and switching between languages. In this iteration, sushi ontology was modified by adding labels with different languages and then modifying the tool to display available languages to the user. Finally, additional view was added to the tool to have two different views: tree and list view. After each iteration an new working version of the tool was produced.

5.1.6 Upload Ontology to the Tool

Uploading ontology after being annotated is the first thing a user need to do. On running the tool for the first time, the tool asks the user for required input like ontology, icon, and logo location; that 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 gets the information from the configuration file and display it to the user. Then it is the user choose to modify these information and overwrite it or not. If the user choose to modify the ontology, logo, icon location; the user gets screen like the one in Figure 22. However if the user click on OK button, the ontology gets loaded to the tool and the application starts to run.


Finder Settings

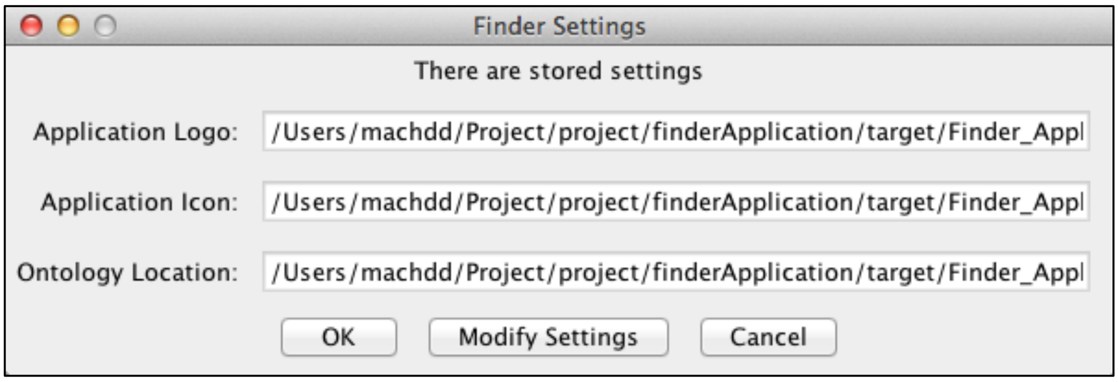
Modify the settings...

Application Logo:

Application Icon:

Ontology Location:

Figure 22: Tool's first run


Finder Settings

There are stored settings

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 run different Ontologies

The tool runs only ontologies with special annotations that mentioned in both Table 3: Run 2 user story and Figure 18. On running the tool, it first search for a class uses hasRole annotation property with the constant value "IngClass". Then, it gets 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. Sushi ontology is the main ontology for this project as it called the " Manchester Sushi Finder". The previous version of the tool which run only one ontology that is pizza ontology. The pizza ontology was annotated with the standard annotations to make it work with the tool. Figure 25 shows that the tool can run ontologies with standard annotations. Figure 24 and **Error! Reference source not found.**Figure 25 show two different ontologies running by the same tool.

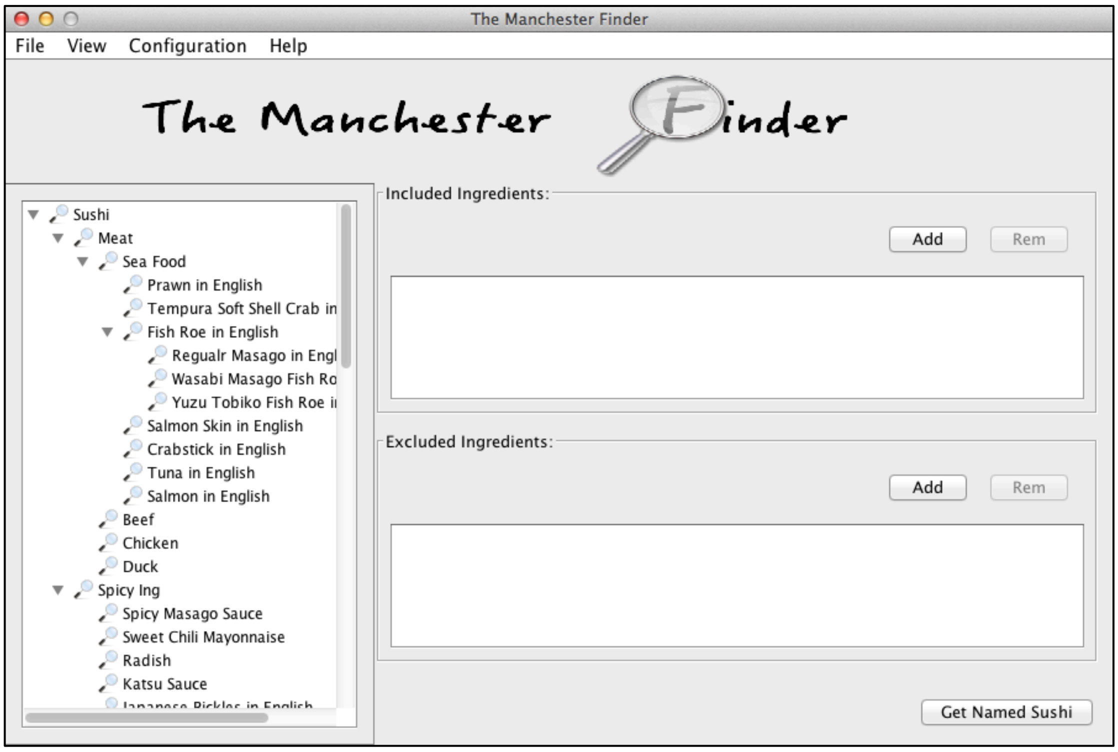


Figure 24: Sushi ontology ingredients

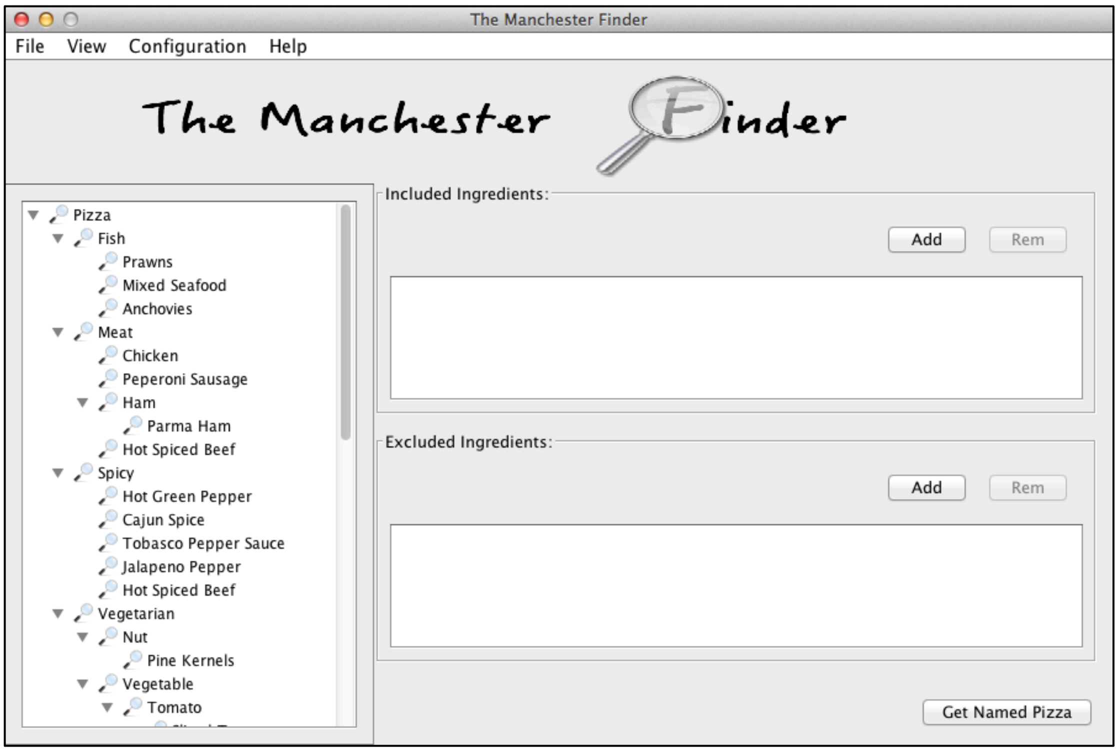


Figure 25: Pizza ontology ingredients

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 hasRole annotation property with the constant value “filter” as mentioned in Table 6: Tree View Filter user story, the tool starts to show these filters. In Figure 26, three classes show as filters, these classes were annotated to be used as filters; these classes are: spicy, vegetarian, and vegan classes. hasRole annotation property is associated with each one of these class. On starting the application, the tool starts searching for classes with filter annotations in them and display them to the user. On checking one of the filters, the tool highlights all classes matching checked filter and disable the rest. As a result, user can only select highlighted ingredients.

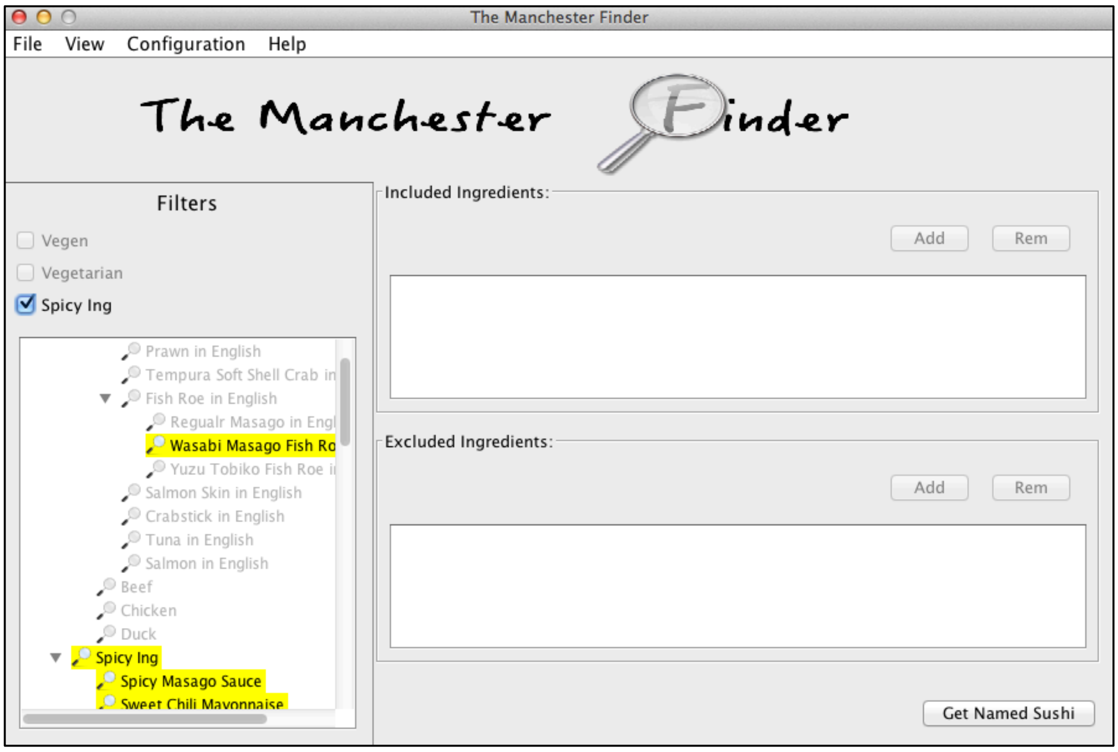
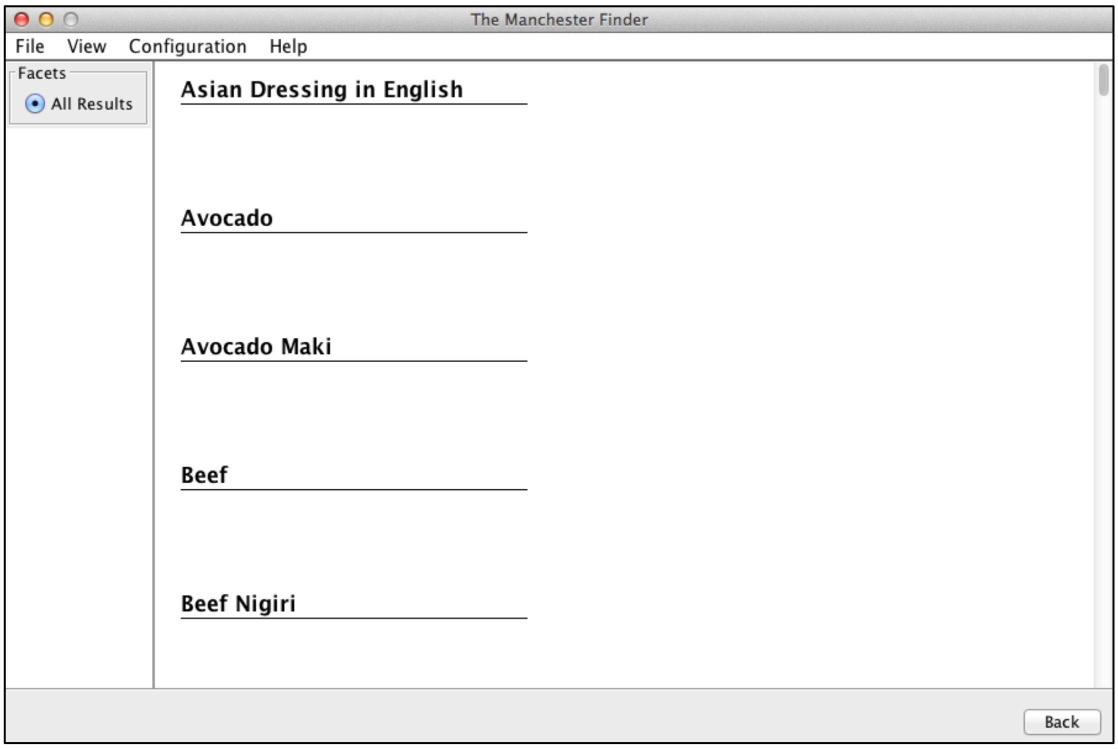


Figure 26: Sushi ontology filters

5.1.9 Filter the Search Result

Filters that applied on the search result are called facets. Figure 27 shows the result of a search query applied on the sushi ontology without specifying facets to be applied on the result. In order to apply facets on the search result, annotation properties need to be associated with value partition classes as mentioned in Table 9: Facet user story. Value partition class like Spicy class needs to have two annotation properties: hasRole, and hasProperty. hasRole property define the role of the class which is facet, and hasProperty property determine which object property link the Spicy class with an ingredient class. After annotating the sushi ontology with facets annotations; when the user try to query for something, number of facets will be displayed depending on the annotations. Figure 28 shows the use of facets, when the user chooses one facet the search result change to match the facets. Facets are associated with ingredients rather than the original result. On choosing Spicy facet, all sushi that have spicy ingredient will show up in the result panel. If the user choose all results, then the unfiltered results shows.



Asian Dressing in English

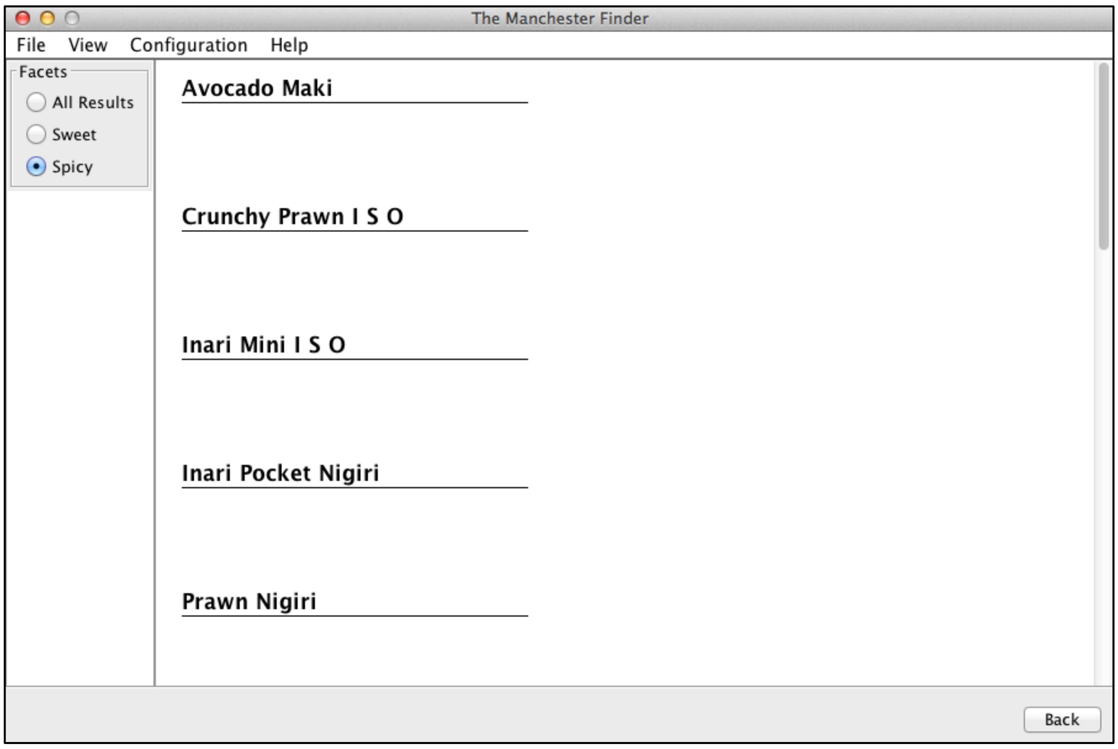
Avocado

Avocado Maki

Beef

Beef Nigiri

Figure 27: No facet is specified



Avocado Maki

Crunchy Prawn I S O

Inari Mini I S O

Inari Pocket Nigiri

Prawn Nigiri

Figure 28: Spicy facet

5.1.10 Display and Switch 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. So if the ontology wan not labeled with any language other than English, the tool shows the English. If the ontology was not labeled at all, tools shows the name of the classes instead. Figure 29 demonstrates the available languages used to label ontology's classes. The tool shows the available languages along with the percentage of their representation to the whole ontology. Notice labels in French language represent 51.83% to the ontology. User can see different languages be going to configuration, then available languages. The below figure shows four different languages: English, French, Spanish, and Arabic. On running the application, 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 labeled with French show in French except the ones without French label will display in the default language English. After querying the result comes in the selected language too.

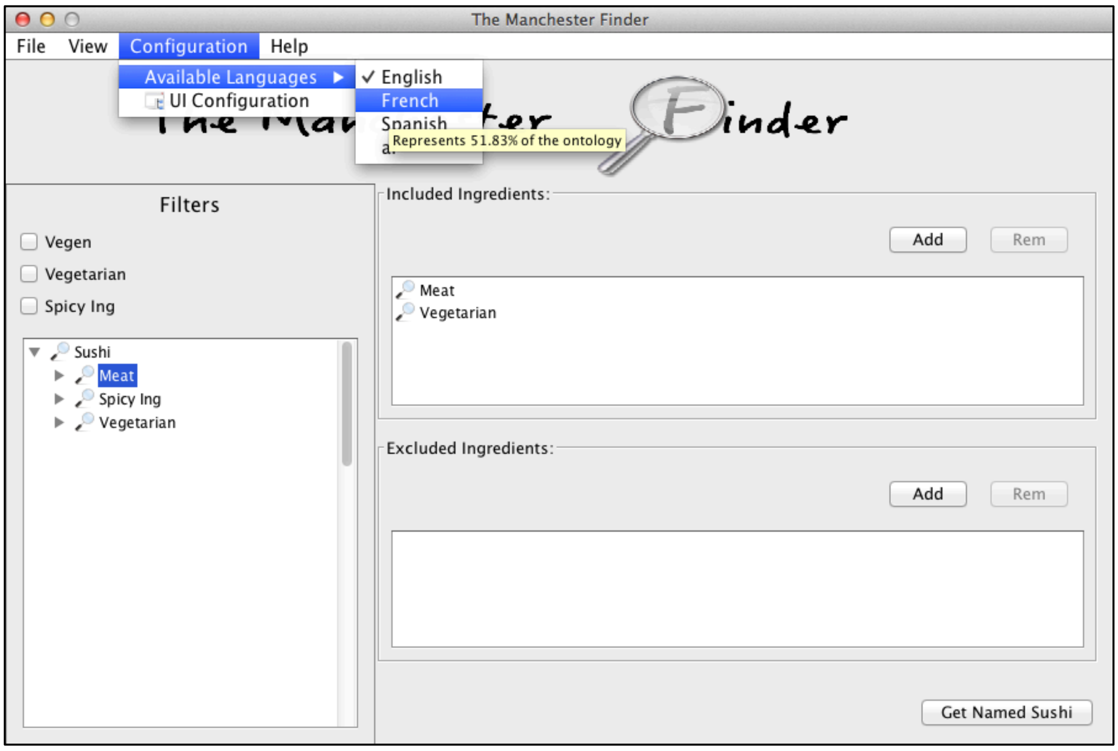


Figure 29: Shown available languages in the ontology

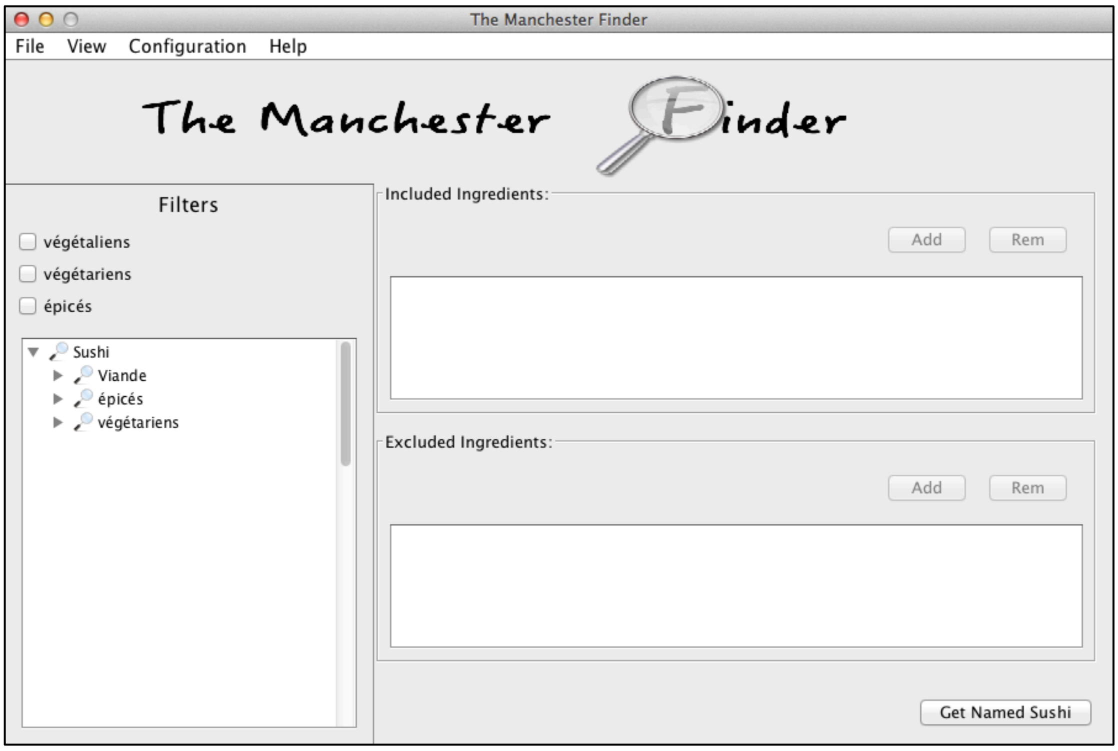


Figure 30: View After selecting French language

5.1.11 Miscellaneous

There are two miscellaneous that have direct relation to the UI and enhancing it rather than relation to OWL ontology or OWL API. These are important since they ease the process of using the UI. These miscellaneous are: different ingredients views, and changing UI labels' and buttons' 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 ontology that is used the tree changes. Tree view contains the filters mechanism that is generated from the ontology. As it is shown in Figure 30, filters that are produced from the ontology is only present in the tree view. Whereas, list view contains all of the ingredients in alphabetical order. View list have different filtering mechanism, since it filter the ingredients based on text. If user enter a text in the filter filed, the ingredients that contain the text shows in the view. 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 filter filed.

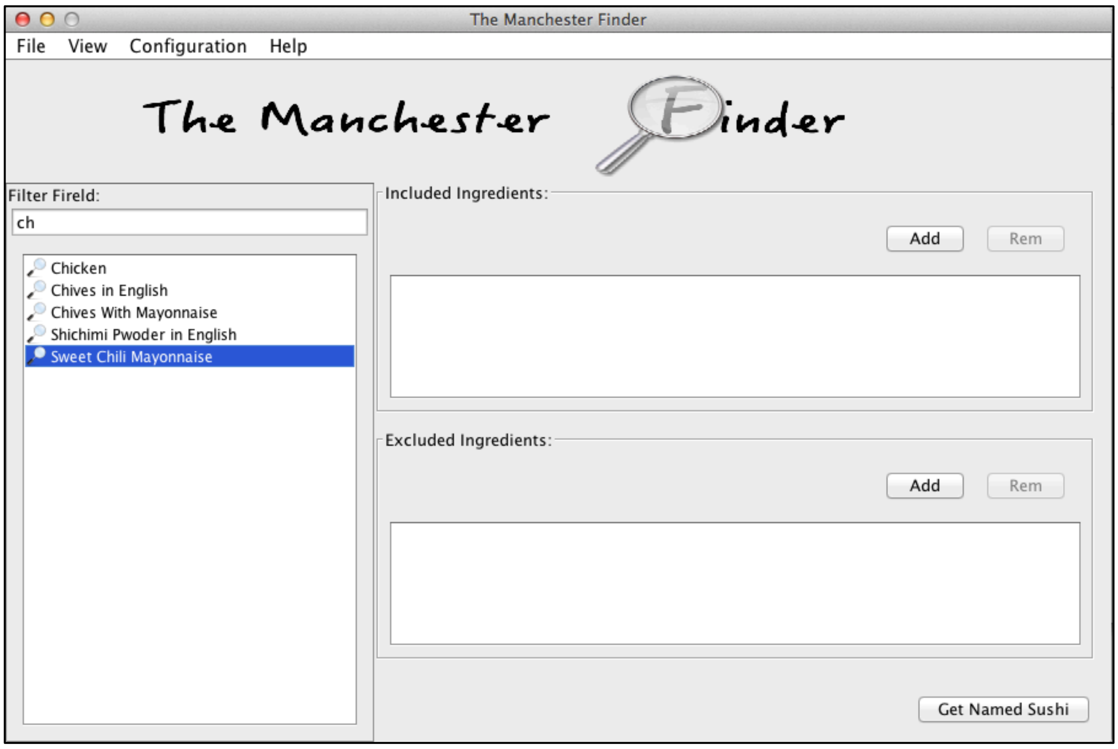


Figure 31: List view

Since the UI has the ability to display different languages depending on the labeled ontology, labels and buttons in the UI need to be changed to match language selected. As it is shown in Figure 32, user enters wanted text in any language and click OK. The application then refresh its views to reflect the entered text. Three labels can be changed the filters heading label in the tree view, the includes and the excludes labels in the query panel. In the other hand , three buttons text can be changed add, remove, and the get buttons.

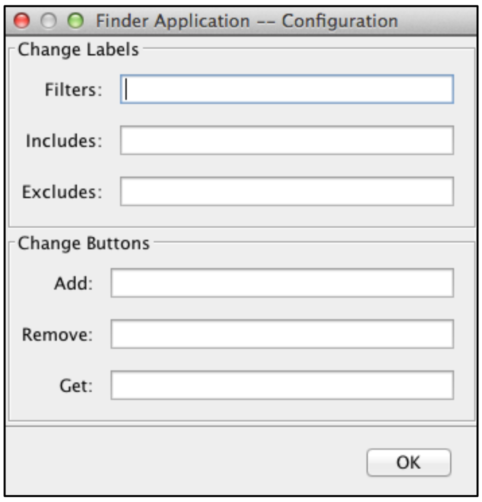


Figure 32: Change labels' & buttons' text

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 the ontology via 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 factor.
2. OWL annotations.

At the beginning of the project, it seemed perfect to use annotations to drive the UI and its functionalities. As human, we are deemed to make mistake. Ontology developer may misspell some of the annotations or even may forget to annotate the ontology that will be used. Developer may annotate wrong classes or wrong object properties. In the other hand, OWL itself suffers from some limitations regarding annotations. In OWL there are two types of classes: named class which is created and defined by the ontology developer, and unnamed class which is a class expression. The best use of filters is with unnamed classes since the named one are visible to the user, and the unnamed one is perfect definition for a filter. For example, spicy class could be a filter because it filter all classes that are spicy. So the user should see the spicy filter but not the spicy class in the ingredient, but that not the case. OWL annotation properties cannot associated to unnamed classes it can only associated to

IRI or Anonymous Individual as stated in OWL 2 specifications

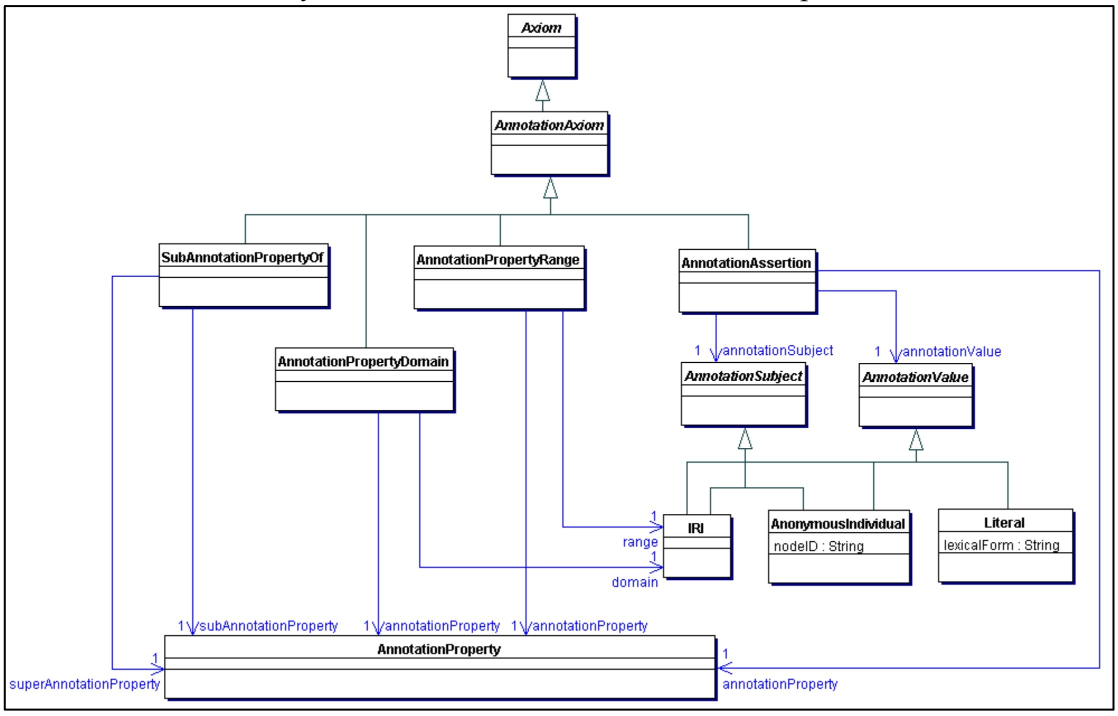


Figure 33: Annotations of IRIs and Anonymous Individuals in OWL 2 [24]. As a result of this limitation, user see the filter as filter and as a class.

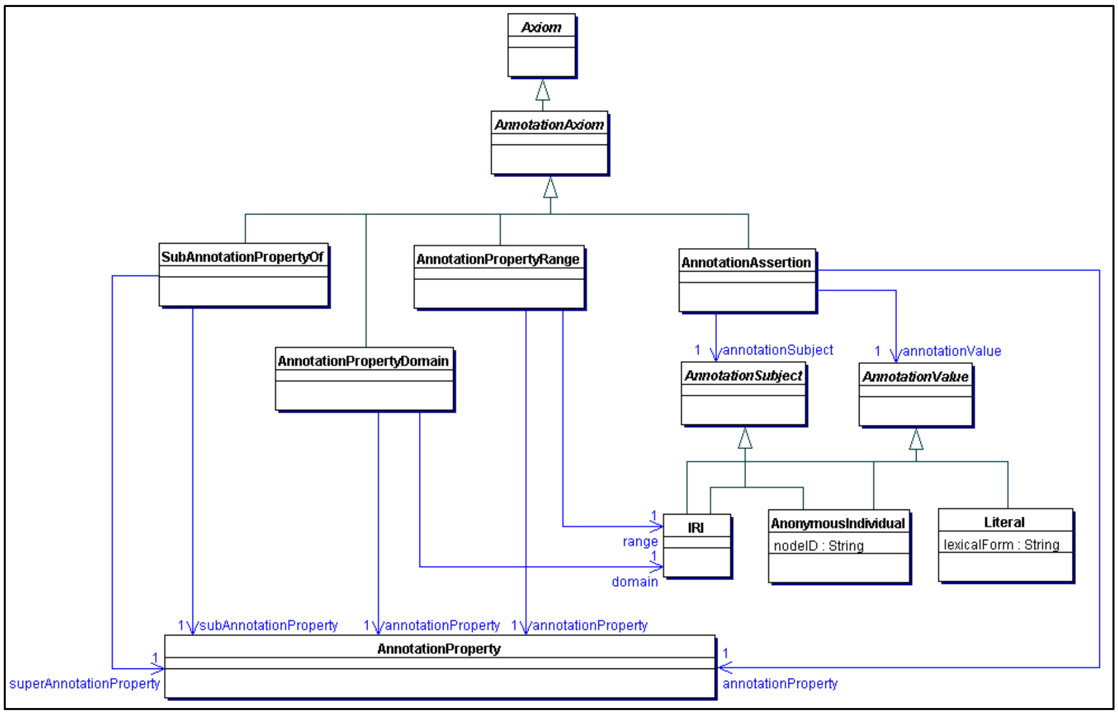


Figure 33: Annotations of IRIs and Anonymous Individuals in OWL 2 [24]

5.3 Conclusion

This chapter illustrated aspects of the implementation phase. Brief overview of the technologies used to implement the Manchester Sushi Finder. Then went through major functionalities' implementation process. Finally, it discussed some limitations faced during the implementation phase.

Next Chapter shows the testing process of the Manchester Sushi Finder.

6 TESTING

6.1 Unit Testing

6.2 Integration Testing

6.3 Conclusion

7 EVALUATION AND CRITICAL ANALYSIS

7.1 Questionnaire

7.1.1 Questions

7.1.2 Participants

7.1.3 Results

7.1.4 Hypothesis Acceptance

7.2 Conclusion

8 CONCLUSION AND FUTURE WORK

8.1 Summary of Achievements

8.2 Future Work

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.