Feasibility of the Application of Semantic Web Ontologies to Enhance Question Answering Systems in Practical Domains

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Douglas Jose Holmes

August 2012
Non-Exclusive Distribution License

In order for California State University Channel Islands (CSUCI) to reproduce, translate and distribute your submission worldwide through the CSUCI Institutional Repository, your agreement to the following terms is necessary. The author(s) retain any copyright currently on the item as well as the ability to submit the item to publishers or other repositories.

By signing and submitting this license, you (the author(s) or copyright owner) grants to CSUCI the nonexclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video.

You agree that CSUCI may, without changing the content, translate the submission to any medium or format for the purpose of preservation.

You also agree that CSUCI may keep more than one copy of this submission for purposes of security, backup and preservation.

You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. You also represent and warrant that the submission contains no libelous or other unlawful matter and makes no improper invasion of the privacy of any other person.

If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant CSUCI the rights required by this license, and that such third party owned material is clearly identified and acknowledged within the text or content of the submission. You take full responsibility to obtain permission to use any material that is not your own. This permission must be granted to you before you sign this form.

IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN CSUCI, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT.

The CSUCI Institutional Repository will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Title of Item

Feasibility of the Application of Semantic Web Ontologies to Enhance Question Answering Systems in Practical Domains

3 to 5 keywords or phrases to describe the item

Computer Science Ontology Question Answering

Authors' Name (Print)

Douglas José Holmes

Author(s) Signature

Douglas Holmes

Date

8-5-2012

This is a permitted, modified version of the Non-exclusive Distribution License from MIT Libraries and the University of Kansas
Feasibility of the Application of Semantic Web Ontologies to Enhance Question Answering Systems in Practical Domains

by
Douglas José Holmes

Computer Science Program
California State University Channel Islands

Abstract
This thesis focuses on applying a Semantic Web ontology to a QA system to determine if the data returned provides useful and meaningful answers to the questions. A system of this type can be used in the creation of a QA system that provides context-relevant answers to a natural language question. In this, I provide a field overview to discuss concepts relevant to the semantic web and question answering, as well as provide context in the form of existing question answering systems. I describe the design implementation and operational concepts of the AcademyQA, a question answering system that depends on an OWL Ontology known as the Academy Ontology. I explain the automatic processes used to classify the question, and construct useful answers to those questions. I construct a series of trials for the system, and analyze the results of the system's operations. Finally, I compare the Academy QA system to several existing question answering systems, and discuss potential future work based on this system.
Acknowledgements

I would like to thank Douglas Irving Holmes Jr., who was an inspiration for this thesis, and provided a wealth of knowledge during the construction of the Academy Ontology, constructive criticism during the completion of the paper, and useful information concerning question answering, semantic web, and ontologies. I would also like to thank my advisor, Dr. Andrzej Bieszczad, who provided insight and feedback necessary to complete this paper, and for his instruction and guidance throughout my time at California State University Channel Islands. This extends to the students and faculty of California State University Channel Islands as well; I am grateful to be a part of such a welcoming community. Finally, I would like to thank my family and friends for their encouragement and patience throughout the process of this work’s completion. Without their support, this work would have never been possible.
TABLE OF CONTENTS

Chapter 1: Introduction  4
  1.1 QA Systems  4
  1.2 Semantic Web  5
  1.3 Ontology  7
  1.4 QA using the Semantic Web  7
  1.5 Overview of Thesis  8

Chapter 2: Field Overview  9
  2.1 Semantic Web  9
    2.1.1 Data representation and exchange with XML  10
    2.1.2 RDF and RDFS  11
    2.1.3 OWL  12
    2.1.4 SPARQL  13
  2.2 Semantic Web Ontology  14
  2.3 Question Answering Systems  16
    2.3.1 Natural Language Processing  16
    2.3.2 Domain  17
    2.3.3 Question Classifier  17
    2.3.4 Query Parsing  18
    2.3.5 Search Engine  19
    2.3.6 Answer Selection  19
  2.4 Example QA Systems  20
    2.4.1 Lunar  20
    2.4.2 ELIZA  21
2.4.3 True Knowledge

2.4.4 EphyraQA

Chapter 3: Analysis of the Academy QA System

3.1 Question Answering System Analysis

3.1.1 Question Parsing

3.1.2 Query requirements

3.1.3 Answer Extraction

3.1.4 System Architecture

3.1.5 Use Cases

3.2 Ontologies for the Academy QA System

3.2.1 DOLCE

3.2.2 Academy Ontology

Chapter 4: Implementation of the Academy QA System

4.1 Tools Used

4.1.1 Protégé

4.1.2 Java

4.1.3 Eclipse

4.1.4 Jena

4.2 Implementation of the Academy QA System

4.2.1 Analysis and Development of the Academy Ontology

4.2.2 Analysis and Development of Questions using SPARQL Queries

4.2.3 Application of Jena tools

4.2.4 Development of the QA Processing System

Chapter 5: Analysis of Results
5.1 Analysis of Experiments 42

5.1.1 Test Set 42

5.1.2 Limitations 48

5.2 Comparison to Other Work 49

5.2.1 OpenEphyra 50

5.2.2 EAGLi 51

5.2.3 Existing Ontologies 52

Chapter 6: Conclusion 53

6.1 Future Work 54

References 56
Chapter 1: Introduction

1.1 QA Systems

Question Answering (QA) is a task discussed in natural language processing that deals with retrieving accurate answers to natural language questions using a text corpus. Natural language processing, itself, is the process of a computer extracting meaningful information from an input consisting of natural language, or producing output in natural language [2]. A QA system is expected to demonstrate a means of interpreting user input, and to answer that input in a manner that is interpretable by the user [1].

When building a QA system, there are three common factors that come into play: the domain, the question, and the answer. Today, such systems often focus on answering factoid questions, which look for concise facts as answers (e.g. "Which planet has the most moons in our solar system?"); list questions, which look for a list of factoids (e.g. "What are the five largest moons that orbit Saturn?"); definitional questions, which ask for descriptive information on a specific topic (e.g. "What is a planet?"); and complex questions, which imply answers that are composed from multiple sources (e.g. "What planet is closest to our own, and how much fuel does it take a space shuttle to perform a round trip?"). Complex questions are beyond the scope of this work, and are something to be explored further in future systems [1].

In a particular information domain, we look at the corpus of the system, or the set of texts from which the QA system draws its answers. QA systems are often separated into closed or open domain systems. Closed-domain systems look at answering questions for a specific topic. For such systems, you do not necessarily expect to receive an accurate answer to questions not concerned with the domain. For example, if you asked a system focused on Chemistry a question about Football, you are likely to receive a nonsensical response. However, because the domain is focused on answering questions related to a specific topic, the corpus can be structured to correspond to specific phrases related to that domain, allowing for a broader range of questions to be asked within that domain and improving precision, accuracy and level of detail in responses.

Open domain systems, on the other hand, attempt to answer any question provided to the system. These systems depend on a lot more data within the corpus to work with, providing a far greater number of answerable questions. But these systems also require a greater level of question interpretation and computation in order to provide suitable answers. Consider, for example, the vague question "Which country is largest?" This is a question that a naive user might ask a QA system, intending to find out which country in the world right now has the largest land area. But because of the vagueness of the question, an entirely open domain system also has to take into consideration different periods of time, interpretations of the word "country," and different scales of which the term "largest" applies to, such as land mass, population density, or economic power. As a result, a large number of answers which may not match the user’s intent are returned.

Once the question has been identified within the context of the QA system’s domain, there must also be a means of using information presented in the question to search the corpus
for a response. This can be done through keyword-based techniques, finding relevant words or phrases and filtering or ranking results in the corpus based on that. Such a method is known as a “shallow” QA method, and works well with a large redundant data source where the answer is a reformulation of the question (e.g. “What is Red?” “Red is a color.”). There are situations, however, where keyword matching based on the content of the string does not work; in particular, words where ambiguities in natural language (e.g. synonyms, jargon, wordplay, etc.) can change the context of a question completely (e.g. “case,” “author,” “instrument,” “barrier,” “capital,” etc.). In such situations natural language processing techniques, such as semantic and contextual processing, are needed to extract a correct answer. This is known as a “deep” QA method.

Over the years, QA systems have grown in relevance and necessity. In this past year, we have seen a QA system which was able to compete against other humans in a game of Jeopardy, through IBM’s Watson[4]. We have seen public presentations of the medium through True Knowledge[7] and Wolfram Alpha[9]. We have even seen QA systems extended in personal assistant applications, such as Siri[26] and Evi[7]. As we pursue this subject, we have been able to peruse even larger domains, including the World Wide Web for general QA purposes, as is done with EphyraQA[1]. One area that has become more important as the domain for QA systems have become more open, is how questions are asked, and how the system interprets those questions. If the data in the corpus are well defined, the number of possible interpretations of the question are more limited. But this requires that data on the Web are well defined.

1.2 Semantic Web

The goal of the Semantic Web is to extend the World Wide Web to better define data for use by both humans and machines. Through the use of software agents, and other computing technology such as search engines and wikis, we are able to retrieve and parse documents in order to obtain information from the World Wide Web. These information searches are often implemented using string matching algorithms to return documents, or pages of information related to the original input. Such applications give users a means of fast access to information, but rely on the user to search through and then read results to find the information most relevant to them. This is because the application does not have any means of interpreting the semantics of the user’s search; it just attempts to match the string to similar strings on a Web page, or wiki article. The serious shortcoming of these kinds of implementations is that, because the application does not understand the input, it cannot reliably return cogent information that is relevant to the user. That is to say, that the search often returns results that are not relevant to the user’s intent, and that it may also fail to return relevant results if they do not contain the specified string sets.

The World Wide Web Consortium (W3C) is addressing this problem through an effort to specify the semantics of the data available on the World Wide Web in a machine readable fashion. This effort is known as the Semantic Web. According to the W3C, “the term “Semantic Web” refers to W3C’s vision of the Web of linked data. In this approach, the semantics of data are defined in Web accessible structured schema, commonly known as ontologies. Semantic Web
technologies, then, enable people to create data stores on the Web, build vocabularies, and write rules for handling data” [28].

It is because of this that QA Systems are an ideal application for the Semantic Web. Through the use of ontologies, we are able to provide computer systems with a well-defined semantic index or “ontology” for a particular domain corpus. This allows for techniques such as semantic parsing, as demonstrated in the EphyraQA system, to be used in identifying relevant terms in a question for providing relevant answers [1]. Thus, instead of searching for specified strings - patterns of literals - applications are able to search for objects that relate to the definition, meaning, of those strings.

Semantics are critical for QA in several ways. One of these is seen in the interpretation of user input - that is, in specifying the question. While this might appear trivial in a system operating in a very limited domain, or one that relies on questions framed in a constrained natural language, it often is not. Moreover, as domain and language constraints are relaxed, lexical ambiguities and imprecision introduce significant problems in correctly parsing questions. Nevertheless, it is essential for a system to be able to interpret a question in order to properly answer it.

This can be accomplished, albeit imperfectly, using inductive linguistic syntactical analysis and statistical methods with varying degrees of success. Alternatively, sufficiently unambiguous meaning of both the terms in a question and their grammatical relations can be assessed by reference to a domain ontology. Such an ontology contains logical definitions of all significant concepts in the domain and the relations between them. Since these definitions are represented as formal logical statements, such an ontology supports automatic machine inference about those concepts. These inferences then enable a proper interpretation of a question, regardless of the manner in which it is posed.

This demonstrates the other area that semantics are useful and perhaps necessary in QA: formulating an answer. Without an appreciation of the meaning of a question as well as the data retrieved in response to it, it is difficult to formulate an answer to a natural language question. While, with some luck, the words in the answer sentence may exactly match those in the question, the meaning behind those words may be completely different. A well defined corpus allows for an application to extrapolate specific relations between phrases that allows questions to be answered in a broader domain [1].

The World Wide Web has been used in some QA Systems as a large, redundant data source, and it seems likely that practice will continue to expand in scope and detail. By creating information on the Web that is accessible and also well-defined, we are able to create applications that are able to parse strings into lexical knowledge bases, and provide information to the end users that is relevant to their original query. As the Semantic Web proliferates, the accuracy and effectiveness of both closed and open-domain systems utilizing it will certainly improve. A QA System that demonstrates this is True Knowledge, discussed in greater detail in the following section, which uses ontologies created both through user interaction with the site, and integration with existing Semantic Web ontologies as a corpus, as well as a means of extending the meaning of natural language queries [2].
1.3 Ontology

The Semantic Web provides developers with standard specifications of representation languages, query languages, inference engines, infrastructure, and other tools necessary to create Web-based ontologies that both define data with respect to a particular domain (e.g., Web site, corporation, region, etc.) and enable logical interoperability with other specified ontologies. As Guarino et al. describe it, “these ontologies give us a means to share fundamental rules of vocabulary and knowledge with other users of the Semantic Web. Describing this semantics, i.e. what is sometimes called the intended meaning of vocabulary terms, is exactly the job ontologies do for the Semantic Web [25].”

An ontology is a formal, explicit specification of a shared conceptualization [42]. We use ontologies as a structure for a model for representing knowledge. Another way of putting it is to analyze the differences between a taxonomy, which is also a structure for representing knowledge. The primary difference between a taxonomy and an ontology is seen in the nature of allowed relationships between terms. A taxonomy is a representation through classification; that is, the only relationship described is subsumption (is-a). On the other hand, an ontology is used as a representation through all knowledge - including classification - that are typically represented as logical axioms. These relationships are, in fact, constrained only by the logical fragment that underpins the ontology. In other words, if we were to put a red fox in terms of a taxonomy, we observe that the red fox is a *Vulpes vulpes* [42] which is subsumed by (is-a kind of) canid. If we were to put a red fox in terms of an ontology, we observe not only that a red fox is classified as *Vulpes vulpes*, but also that the red fox “has-nose” long narrow snout, “has-tail” bushy tail, “has-dietary-habit” omnivore, “has-prey” types of animals that it preys on, along with the classifications of those animals, their features, eating habits, et cetera. Rather than attempt to describe things just by naming them, we describe everything about them, which allows us to derive more about them.

This structure is crucial for the advancement of natural language processing. In nearly every QA system described in the following chapter, there is employed some form of ontology to extract information from the corpus. In the case of the Semantic Web, the corpus is often integrated directly with the ontology itself [6]. With a fully realized Semantic Web ontology linked with other ontologies, it may be possible to even create QA systems that are able to interpret and resolve user input to a greater extent than many modern-day search engines are capable of. This will be more apparent as projects such as Watson [5] and Google’s Knowledge Graph [49] advance forward.

1.4 QA using the Semantic Web

This thesis focuses on applying a Semantic Web ontology, represented using a language standardized by the W3C known as the Web Ontology Language (OWL), to a QA system in the areas of question parsing and searching, then determining if the data returned provides useful and meaningful answers to the questions. The goal of this is to assess the feasibility of a Semantic Web QA system using an OWL ontology as a corpus. With the assistance of a foundational ontology on the back end, and a framework to interpret and extract information from that ontology for the front end, it is possible to implement an extendable QA system that can
adequately answer questions not just for a single ontology, but any ontology that extends that particular foundational ontology. The objective of this is to retrieve and parse information from an OWL ontology using queries written in the SPARQL Protocol and RDF Query Language (SPARQL).

The ontology is a university catalogue ontology, referred to as the “Academy” ontology. This ontology builds on the foundational ontology known as the Domain Ontology for Linguistics and Cognitive Engineering (DOLCE). The Academy ontology was constructed from the perspective of students, professors, or administrators attempting to find information about respective areas concerning a university. The operational concept is to take a controlled natural language question from one of these academic groups, parse key words and phrases into the Academy ontology using SPARQL, and determine the meaning of that question to formulate a query and extract an accurate set of results.

The motivation behind this is to evaluate these emerging technologies, and determine whether these technologies apply in a practical domain. It has been demonstrated through other semantic QA systems, such as EphyraQA[1] and True Knowledge [7], that it is feasible to use the World Wide Web as a corpus for a QA system. The goal of this thesis is to demonstrate that Semantic Web can be used in conjunction with the techniques introduced with these systems.

Taking this a step further, a future goal based on capabilities derived from this thesis, is the creation of a QA system that provides relevant answers to a natural language question. The concepts demonstrated in this thesis are a basis for an intelligent QA system capable of inferring the meaning of the question via ontology, then selecting an answer via a combination of semantic reasoning and statistical threshold. From there, such a system is able to use other Web ontologies as potential domains for answering questions without having to extend those ontologies to accommodate the system.

1.5 Overview of Thesis
This work is divided as follows:
- Chapter 1 introduced the discipline of QA, as well as a brief summary of different approaches to QA systems. It also discussed the motivation for using the World Wide Web as a data source for QA System, and the importance of Semantic Web in QA Systems.
- Chapter 2 provides an in-depth field overview, bringing into consideration the different parts of a QA system, explain the Semantic Web stack and how it applies to this thesis, and cover a range of QA systems to provide insight into the techniques used for existing semantic applications.
- Chapter 3 provides an analysis of the Academy QA system used for this thesis, as well as the Academy ontology that is used to interpret queries and serve as the corpus.
- Chapter 4 examines the tools used for the creation of the Academy QA system and Academy ontology, and the process of setting up both of these.
- Chapter 5 evaluates the results of the QA System, and compare these results to those created using similar open domain QA Systems.
- Chapter 6 provides a conclusion to this work with a summary of the work completed in this thesis, and provides details for future work in QA using the Semantic Web.
Chapter 2: Field Overview

This chapter focuses on the technologies involved in building the Semantic Web question answering system. This includes a description of the basic mechanics of a question answering system, including natural language, question classification, query parsing, and answer selection. This chapter also discusses aspects of the Semantic Web, and how they pertain to this thesis.

2.1 Semantic Web

The Semantic Web is a term first coined by Tim Berners-Lee of the World Wide Web Consortium to denote a collaborative movement started in the late 1990’s. The intent is to convert the perspective of the Web from a collection of documents, known as Web pages, into a “Web of data” that can be accessed and processed directly, and indirectly, by machines [11]. Through use of the Semantic Web, we are able to collect, organize and use the vast information resources found throughout the World Wide Web into a common representation that can be read by an arbitrary collection of applications to generate a proper response suitable to that application.

This section discusses some of the technologies that make up the Semantic Web stack, shown in figure 2-0 below, as they apply to this thesis. Section 2.1.1 describes XML, and its purpose as the underlying syntax for data exchange on the Semantic Web. Section 2.1.2 discusses RDF and RDFS, and how they are used to define and describe logical concepts and propositions on the Semantic Web. Section 2.1.3 covers the Web Ontology Language, OWL, which is used to represent knowledge in Semantic Web ontologies. Finally, section 2.1.4 covers the SPARQL query language, and its purpose of traversing Semantic Web taxonomies and ontologies.

Figure 2-0. The Semantic Web stack[56].
2.1.1 Data representation and exchange with XML

XML (Extended Markup Language) is a markup language that shares features of syntax in common with HTML[47]. The primary difference between the two is that, while HTML is a declarative language used by browsers to determine how to display data, XML is used to describe and exchange data between applications. As an example, consider the following HTML:

```
<html>
  <body>
    <b>Grass is green.</b>
  </body>
</html>
```

This creates a simple document telling any compliant browser that, in the body of the document, to place a bolded string of text “Grass is green.” There is no indication to what that string of text means, only that it must be bolded (<b>) and placed in the body (<body>) of the HTML document (<html>) [20].

This is a bit different in XML, where the primary objective is to exchange data between distinct applications and aspects of what the page displays are not taken into consideration. XML is a data representation language consisting of node-value pairs, where the nodes are arbitrary tags that represent data elements and values are the data contained in between the tags. An example XML document is:

```
<xml>
  <plants>
    <grass>
      <color>
        Green
      </color>
      <waterneeded>
        Little to none
      </waterneeded>
      <sunneeded>
        Lots
      </sunneeded>
      <growingtime>
        4-7 days
      </growingtime>
    </grass>
  </plants>
</xml>
```

This can translate out to a similar statement as the one above: “the plant grass is the color green.” The difference, of course, is that there isn’t any markup defined for how to display this
on a page. This information is instead used to define aspects of grass in a manner that can be interpreted by the system. The more information added to the document, the more the system can know about the subjects covered in the document[20]. In order to support effective data exchange, the applications must have a common specification of the data elements, typically called a schema, that maps the XML to data used by the application.

2.1.2 RDF and RDFS

Although XML can effectively be used to structure data, we can make it more powerful by extending it to handle data triples, that assigns a grammar to the data. This is what RDF, the Resource Description Framework, is designed to accomplish. RDF triples indicate a subject, predicate, and an object[20] syntax to create logical statements populated by the data contained in the triple. RDF uses user resource identifiers (URIs) to identify resources, and describes resources with properties and property values. Thus users are able to assign primitive, but useful, semantics to the data.

An example of RDF can be seen in figure 2-1, below:

```xml
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:cd="http://www.recshop.fake/cd#">
  <rdf:Description
    rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
    <cd:artist>Bob Dylan</cd:artist>
    <cd:country>USA</cd:country>
    <cd:company>Columbia</cd:company>
    <cd:price>10.90</cd:price>
    <cd:year>1985</cd:year>
  </rdf:Description>
  <rdf:Description
    rdf:about="http://www.recshop.fake/cd/Hide your heart">
    <cd:artist>Bonnie Tyler</cd:artist>
    <cd:country>UK</cd:country>
    <cd:company>CBS Records</cd:company>
    <cd:price>9.90</cd:price>
    <cd:year>1988</cd:year>
  </rdf:Description>
</rdf:RDF>
```

**Figure 2-1.** A sample RDF document [19].

The RDF attribute itself specifies the namespace definition for RDF, along with the namespace definition for a CD, which contains its own values. In the first record of the document, the `<rdf Description>` element identifies a resource with an RDF definition - about, which contains a value of “Empire Burlesque.” In it, there are attributes from cd - artist, country, company, price, and year. We are able to see that the record is about a cd named ‘Empire

RDF also allows us to be more descriptive about collections of data with tags like `<rdf Bag>` (a set of possible values), and `<rdf parseType=”Collection”>` (describe a group that contains only specified members) [20].

One of the limitations of RDF is that the open ended nature of RDF makes it difficult to explicitly define classes and properties that are useful for software applications. This where extensions to RDF, such as RDF schema (RDFS), come in. RDF Schema is used to define and describe classes, properties, and other resources [58]. Classes in RDF Schema are logical notions, similar to classes in object oriented programming languages. Resources can be defined as classes, instances of classes, and subclasses of classes. The result is a hierarchical structure of classification, making RDF documents using RDF Schema resemble a taxonomy.

### 2.1.3 OWL

The Web Ontology Language (OWL) is a formal knowledge representation language, based on an extension of RDF, designed for creating and maintaining Semantic Web ontologies [15]. Since it is an extension of RDF, any OWL ontology can also be viewed as an RDF graph.

An OWL document is a document containing an ontology which can be used on the Web. Systems are capable of interpreting OWL documents without human intervention, allowing for more advanced integration for applications built to interpret them [1]. Some examples of ontologies can be found online on the Protégé Ontology Library [17].

As an example, consider the section of an ontology shown in the figure 2-2.

```xml
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/TR/2003/PR-owl-guide-20031209#PotableLiquid"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker"/>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker"/>
      <owl:allValuesFrom rdf:resource="#Winery"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#madeFromGrape"/>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasSugar"/>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

**Figure 2-2.** The Wine Ontology, provided by the W3C [18].
This ontology specifies a class “Wine.” In this section of the “Wine” class, we see four properties, or cardinality restrictions: hasMaker, with a cardinality value of 1; hasMaker, which then links to the “Winery” ontology; hasGrape, which has a minimum cardinality of 1; and hasSugar, which has a cardinality value of 1. Cardinality refers to a restriction link to the property value specified; in other words, Wine must have a single maker with a winery, and the wine must specify once whether or not it has sugar. MinCardinality is similar, but instead refers to the number of references the primary class must have; in the example above, the wine must specify at least one grape from which it was made.

Aside from defined classes and properties, OWL ontologies specify individuals, or instances of a class that are the main reason for the ontology. With individuals, we are able to populate our ontology with members in RDF. For example, in the wine class we can have an individual named “Santa Cruz Cabernet Sauvignon.” By the cardinality restrictions above, we are required to set properties for madeFromGrape, hasSugar, and at least one hasMaker. With this information, one may run a query searching for all wines in the ontology, and find not only the Santa Cruz Cabernet Sauvignon, but the “Cabernet Sauvignon” among the grapes used, a “dry” sugar content, and the “Santa Cruz Mountain Vineyard” as one of the wineries that make it [18].

The OWL 1.1 standard is split into OWL Lite, OWL Description Logic (DL), and OWL Full [15]. OWL Lite is the most restrictive of the three, and the furthest from simple RDF. It provides a means of subsumption, along with simpler logical constraints; for example, it allows the user to specify cardinality, but only to the point of saying whether or not a single object is required. OWL DL is the preferred version of the language that supports Description Logic, a decidable fragment of First Order Logic. It allows users the option of using the full OWL language, with certain restrictions, one of which is that a class with multiple subclasses cannot be the instance of another class. OWL Full is a less constrained extension to RDF, allowing for the same syntactic freedom while providing the entire feature set of OWL without computational decidability. Semantic Web developers who build tools to interpret their ontologies choose to use OWL Lite or OWL DL to construct their ontologies, while developers interested in building ontologies that leverage the full power of First Order Logic use OWL Full [15].

2.1.4 SPARQL

Query languages are used in databases to retrieve relevant information from a database. Similarly, as RDF/OWL documents become larger and more numerous, it becomes desirable to use a query language, such as SPARQL (SPARQL Protocol and RDF Query Language), to retrieve relevant information from these documents[12]. Recently, the adoption of Linked Data protocols enables SPARQL queries across the entire World Wide Web to include any document that provides a SPARQL endpoint [11]. For example, DBpedia, an effort to extract structured information from Wikipedia exposes a SPARQL endpoint at http://DBpedia.org/sparql and thereby makes all its data available to SPARQL queries. Thus, any OWL ontology can be effectively augmented by Wikipedia, without the effort of manually incorporating DBpedia in the ontology [69]. An example of a SPARQL query can be seen below:
PREFIX plants: <http://iloveplants.com/plants.rdf>
SELECT ?plantname
WHERE { x plants:plantname ?plantname
    FILTER regex(?plantname, "G")}

In this case, we are selecting a plant from our RDF document - let us assume this is the document described above - with a name whose letter starts with “G.” The result of such a query may look something like what is shown in the figure 2-3.

![Table of SPARQL query results.](image)

**Figure 2-3.** Table of SPARQL query results.

SPARQL shares a number of similarities to a similar query language, SQL [66]. The SELECT and WHERE prefixes are intact, and serve similar purposes with SELECT specifying the requested columns, and WHERE specifying the filters necessary. At the same time, it also has a number of differences, such as the PREFIX tag which provides an alias when referencing a specific URI (e.g. “plants plantname” is really &lt;http://iloveplants.com/plants.rdf#plantname&gt;), the columns in the SELECT statement are variables instead of columns, and the use of data triples in the WHERE clause to filter results.

SPARQL is a standard approved by the W3C to retrieve relevant information from a RDF/OWL document[12].

### 2.2 Semantic Web Ontology

An ontology, in terms of the Semantic Web, is a description of concepts and relationships used to store, share and reuse knowledge. An ontology differs from a taxonomy in its expressivity; an ontology admits a much broader range logical and user defined relationships than are found in a taxonomy. Clearly that allows the creation of a much richer - and thus useful - collection of statements about any domain.

While a taxonomy can be viewed as a graph without any cycles (a tree), an ontology can be any kind of graph based on the given set (a model)[16]. This also means that the construction of an ontology is a complex and demanding task, because of the extent necessary to fully develop and define knowledge. It is because of this that many ontologies are modularized or split into what are known as foundational, upper, middle and domain level ontologies.

An upper level ontology provides a structure for broadly defined general concepts (e.g. with DOLCE, the default OWL class Thing is extended as an “Entity,” which can be defined as an “Abstract,” “Event,” “InformationEntity,” “Object” or “Quality”). A foundational ontology is
similar to an upper level ontology, but foundational ontologies are built to be extended by other ontologies - as such, foundational ontologies are designed to ensure and facilitate a mutual understanding between the ontologies that extend it [48]. Middle level ontologies refine concepts introduced in upper level ontologies, with respect to some particular point of view. Lower ontologies, also known as domain ontologies, further refine those concepts to address a specific domain of interest. This separation of interests removes important, but distracting meta-level issues from decisions about practical domain issues and allows the creation of various domain level ontologies that are guaranteed to have a common set of high level ontological commitments, and thus are able to interoperate at some future time.

There is some contention on whether this is necessary [59], but for ontology used in this thesis it serves an important purpose of giving any ontologies with a common foundational ontology a similar set of concepts to base an application on, much like RDF Schema gives applications a similar set of concepts to define what a class or property are [57].

Ontologies play an important role in natural language processing and, by extension, accurate QA systems. The structure of an ontology allows its users to provide as much context as is necessary, and link those objects with an underlying logical structure. Semantic Web ontologies expand on this, by providing a means of sharing these ontologies in a common framework allowing these ontologies to be linked together. It is reasonable to expect that this will lead to more precise open-domain QA systems as the Semantic Web continues to expand.

In this work, ontologies are used to extract terms from question sentences, to be placed into queries that derive result sets from a corpus compliant with the rules of that ontology. This is similar to the implementation used in EphyraQA [1], which used ontologies to 1) extract terms from questions and answer sentences, and 2) expand terms with semantically similar concepts. In the case of EphyraQA, this allows the system to accomplish two more things:

1) terms and alternative representations used at the query generation stage form queries that are independent of the formulations used in the question, resulting in a higher documentation call

2) in situations where the predicates are similar, the resulting algorithm is based on the terms themselves, rather than high level semantic arguments, giving EphyraQA greater flexibility and allowing for a more robust error parsing system.

This system works well for an open domain QA system. WordNet, the ontology used by EphyraQA[1], is a lexical ontology that has been widely used as a semantic resource for QA. WordNet is also an ontology that has made a successful translation to the Semantic Web, thanks in part to OWL, but also in part to the Laboratory for Applied Ontology [52]. To accomplish this, they linked the WordNet ontology to DOLCE, their own foundational ontology. Another such example is a QA system used by the National University of Singapore that has been used and demonstrated in various Text Retrieval Conference (TREC) evaluations [51].
2.3 Question Answering Systems

In reference to Artificial Intelligence, Question Answering is a discipline which focuses on providing an answer to a question using a set of texts, or text corpus. An ideal QA system must be able to accept and interpret a question from any user, then return the correct answer to said question. As a result, QA systems have been an important part of the development of both information retrieval in computers, and natural language processing.

The following sections cover factors of QA that are addressed in the Academy QA system. Section 2.3.1 gives a little background into the problem of natural language processing, and how it pertains to QA systems. Section 2.3.2 describes how QA systems handle different types of domains. Section 2.3.3 begins to delve into the flow of a QA system by discussing the need for classifying natural language input. Section 2.3.4 discusses the next step, which is taking that natural language question and parsing it into a query that the computer is able to understand. Section 2.3.5 discusses how information is retrieved from a text corpus, and the need for an engine that is capable of parsing the query described above. Finally, section 2.3.6 discusses the process of selecting an answer and returning that answer to the user.

2.3.1 Natural Language Processing

Natural language is often described in artificial intelligence as the process of enabling computers to understand human language naturally as if they were human [12]. Natural language itself refers to any sort of unpremeditated language that arises from the human mind. In other words, we understand language, and can process what others say to us either via body expressions, or inflection, or simply talking to each other. Recreating that in computers, as a result, has been an objective of intelligence researchers since the early 1950s[12].
For QA systems, in order to deliver what is perceived as an accurate answer, the system has to not only interpret the question, but the semantics powering the question. An example of this is the following question:

"Which planet is closest to the Earth?"

In this example, the term "closest" can refer to "orbital distance," which results in an answer of Venus, or "relative distance," which is an answer of either Venus or Mars based on the current position of Earth, Mars and Venus. Without context or semantic meaning associated with natural language, it is difficult to derive an answer to this question.

2.3.2 Domain

Regarding QA system, domain describes the questions that are answerable within the system's text corpus [1]. QA systems are divided into closed and open domains. A closed domain contains information on a limited number of subjects, or possibly a limited number of questions can be answered with a large number of descriptors on those questions, while an open domain contains information on everything possible[14]. This leads to a different set of expected objectives for either application: in a closed domain, the subjects covered must be defined as accurately as possible to allow for more accurate answers to a broader range of questions, while an open domain has the additional requirement of ensuring that information is separate to prevent conflicts in query parsing. For example, the question "What is the largest city in California?" may be interpreted as a question of population size, area, level of industry, or any number of factors, but then it may also be interpreted as "What is the largest city in Baja California?" or "What is the largest city in California, Kentucky?" the former of which has a valid answer but be off-topic, and the latter of which is a correct answer at all.

2.3.3 Question Classifier

Part of the solution to the problem discussed in the previous section is in question classification, though this is a subject that is helpful in both open and closed domain systems. A question classifier tells the QA system what type of question the user has asked. At a higher level, questions may be broken down into the following types [1]:

- Factoid questions, which ask for concise answers, often named entities
  ("Who was the lead actor in the movie Back to the Future,")
  "What was the largest country by population in 2008,")
  "When did World War I start")
- List questions, which ask for a list of answers to a factoid question
  ("What are the names of all the actors in the movie Back to the Future,")
  "What were the 5 largest countries by population in 2008,")
  "When did each war involving the United States begin")
- Definitional questions, which ask for relevant information on a given topic ("What
  is Back to the Future," "What is a country," "What is a World War")
Complex questions, which involve composing an answer from multiple sources ("Which actors were asked to participate in filming *Back to the Future* and why didn't they participate,\"
"Which countries build the A380 airplane and how do they contribute,\"
"What were the factors that lead to the first World War, and how did they contribute to its end")

Once a QA system can classify the type of question it is going to answer, it can begin planning a process for answering that question.

### 2.3.4 Query Parsing

The process for query parsing involves taking a natural language question and parsing it in a manner that a computer can understand [1]. Building an accurate query is crucial to receiving a correct answer. As an example, let us consider the Google Search engine and how it interprets the following natural language statement:

What is the temperature in Guatemala?

Client side, we are able to see the first step of Google’s query parser in the query string that is generated:

```
hl=en&site=&source=hp&q=what+is+the+temperature+in+guatemala&pbx=1&oq=what+is+the+%22temperature+in+guatemala%22&aq=f&aqi=g2g-j2&aql=&gs_sm=3&gs_upl=1065l5284l015603l36l19l0l16l16l2l166l1835l10.9l35l0&bav=on.2.or.r_gc.r_pw.cf.osb&fp=a1f7df59a5e58952&biw=980&bih=983
```

In this query, we see parameters to filter the results, such as the client’s language, the site the client wishes to search, and where the search came from. There is also the original query which has been translated to:

"what+is+the+%22temperature+in+guatemala%22"

This statement tells the search engine a few things:

- The original query is not case sensitive
- Split the statement into individual words and phrases
- Search for the phrase within quotations (in this case, %22)

This results in a search which focuses on the key phrase. The results are shown below, with the relevant phrases bolded. Note that both queries provide a correct answer, but for fundamentally different questions.
2.3.5 Search Engine

Search engines serve a similar purpose in QA as they do on the Web; in both cases, the search engine is the system that retrieves information from the corpus based on the query passed in. This requires an understanding of the model used for the corpus, as well as an algorithm for parsing information in that model [1]. To better understand this, it is important to understand a few concepts about graph theory.

A directed graph is an ordered pair \((V, E)\) where \(V\) is a set of vertices and \(E\) is a set of directed edges between the vertices [1]. In an undirected graph, the edges are undirected and can be denoted as binary sets. In either case, we are able to form a path between the vertices, or a sequence of connected vertices, but for a directed graph the path only goes in one direction.

As an example, consider the ontology described in section 2.1.3. The wine class on its own has 8 cardinality restraint specifications, which means there must be at least 8 values for each instance of the wine class. On its own, this can lead to the creation of a directed acyclical graph, in which a pathfinding algorithm as simple as a depth first search may be used to retrieve and present results [21]. When it comes to links between multiple ontologies, such as the one between wine and winery, it is possible that cycles can be formed; for example, if winery were to link back to each individual wine that it created, it forms a cycle. It is because of this that pathfinding algorithms that can detect cycles are necessary in order to perform searches on advanced data models such as ontologies. As a result, we see more efficient searches with fewer redundant results in a shorter amount of time.

2.3.6 Answer Selection

Once a search engine has retrieved relevant information from a corpus, the QA system must then select an answer. This is often done through ranking systems [1], which rates the answers based on the relevance to the context of the original question. How relevance is weighted depends heavily on a system focused on answer selection; for example, a process of scoring answers based on how many of the words match the content of the original question. Likewise, it is possible that the system weighs the answers based on how closely matched the definitions and relations are, based on information from the corpus. In a semantic system, it is
possible to employ a semantic parser, which has defined specific roles that are then searched for and compared within the answers, and a score is generated based on how closely answers fulfill the role of the original question [1]. This is not always as effective, due to semantic parsers often failing to recognize argument boundaries like prefixes, suffixes, articles, and prepositions.

A number of scoring techniques can be employed in determining which answer is the most relevant to the question asked. In the Ephyra framework [1], score normalization techniques such as local scaling based on metasearch systems combining results from multiple systems, global training which takes answers from many different systems into account, and score combination, which is used to analyze and derive conclusions from the prior two systems. All of these lead to a simple rating, the highest of which is chosen for the eventual answer.

2.4 Example QA Systems

This section covers several QA systems of interest. QA systems have been created for use since 1968, for a number of different purposes. Each has pushed natural language processing forward, though they have all been used for specific purposes. Section 2.4.1 discusses Lunar, an early QA system that demonstrates core techniques in QA. Section 2.4.2 discusses ELIZA, and how a QA system can be employed without a large corpus. Section 2.4.3 discusses a modern QA system known as True Knowledge, and some of the assets and limitations of using the Web as a domain. Finally, Section 2.4.4 discusses EphyraQA, a QA system whose question classification techniques are used in this thesis.

2.4.1 Lunar

Lunar is one of the earliest known QA systems. It was developed by NASA during the 1960s to answer questions concerning the geological analysis of rocks gathered by the Apollo 11 mission [2]. This was accomplished by taking a natural language question, parsing the question into a query recognizable by the database, and returning the result of that query. This was a closed domain system, using a loose subset of English for the query parser and a vocabulary consisting of about 3500 words. While the system was capable of providing accurate answers, it required an advanced user capable of interpreting and translating questions into an appropriate query, precise questions through a controlled natural language input, and required computational power that was uncommon for the time [2]. This project demonstrates techniques that are still commonly used in QA systems today, one of which is the interpretation of a natural language question and translation to a machine relatable query.
2.4.2 ELIZA

ELIZA is a QA system developed between 1964 and 1966 to demonstrate the importance of psychology when it comes to natural language [22]. The corpus behind ELIZA was very limited; in fact, the system was designed to side-step the use of a database of knowledge. When a user asked ELIZA a question that was not in this knowledge base, rather than not return an answer, it provides a generic response. The idea was to mimic a doctor, so when someone, for example, say to the system “My head hurts” the system responds with “why does your head hurts?” By providing these answers, the system is able to fool its users into perceiving it as an entity capable of conversation. ELIZA is considered to be the earliest chatterbot, a system that is created for the purpose of responding to its user [22].

Figure 2-6. A sample LUNAR query [2].

Figure 2-7. A picture of ELIZA running in EMacs, provided by Wikipedia [23].
2.4.3 True Knowledge

True Knowledge is one of the first major semantic QA systems to be made widely available on the World Wide Web[7]. Currently, it is not entirely open domain as they continue to establish definitions for certain areas, such as popular culture, but it is considered to be a general purpose QA system.

The True Knowledge Answer Engine works by taking the question posed, and deriving every possible meaning of that question and the phrases within using its knowledge base[7]. The knowledge base is set up in such a way that the computer can understand the facts well enough to associate them with the stated question, allowing the answer selection process to produce an answer to the question via logical deduction. For example, if one were to type in “What is the closest planet to Earth?” as is shown in figure 2-8, below, True Knowledge returns the facts “Earth is a planet,” “a planet is a celestial body that is in orbit around the Sun, has sufficient mass so that it assumes a nearly round shape and has cleared the neighborhood around its orbit,” “Venus has always been the result of the position class and ordering first, planet and [ordering by left relation: [is the astronomical distance to]],” to produce the singular answer, “Venus.” While the answer does not necessarily follow the exact orbital distance between Earth, Venus, and Mars, it does follow the deductive logic surmised by the program.

There are several concerns to be raised for True Knowledge. Currently, it is capable of answering factoid questions via fact association, as was described above. This sort of fact association makes it difficult for the program to handle numerical queries, for instance, the system understands events that occur in the month of June, but it cannot say the difference in months between February and June. This may be a limitation of their deductive process, but it can be addressed over time by extending the system to include numerical constraints in their internal knowledge base.

![Figure 2-8. A sample query run in True Knowledge [7].](image)

2.4.4 EphyraQA

Ephyra provides an open domain framework for QA systems [1]. It integrates several techniques for question analysis, query generation and answer extraction, and it is capable of extracting answers from structured and semi-structured knowledge sources, such as Web documents. By integrating a combination of ontological parsing via WordNet, and returning an
answer based on a combination of statistical techniques and reasoning, Ephyra is able to provide answers to full natural language questions [1].

EphyraQA organizes the structure described in previous sections into a reusable pipeline. This system is also extensible, allowing for additional techniques for question analysis, answer extraction and answer selection, or even adding in additional knowledge sources.

The process for an EphyraQA query is as follows:
- Normalize the question string by modifying tenses and verbs, replace nouns with their lemmas.
- Analyze the question string to determine the expected answer type, named entities and other terms to create a concise interpretation of the question consisting of its key phrases.
- Use the information collected by the analyzer and insert the information into a series of queries used by the query generator. By default, this is a “bag of words” keyword search that matches based on individual keywords.
- Extract the answers from the search engines used for the search - by default, Google and Indri - and filter the answers through a pipeline that rates and ranks each answer based on relevance as defined by the techniques applied.
- Select and display the highest ranked answer, source, and rating.

The results of the default Ephyra framework, as evaluated in TREC 2006, can be found in [1]. What makes this framework interesting is that it is capable of incorporating additional QA techniques, as defined by the developer, to help retrieve more precise answers. This is how Nico Schlaefer, in [1], was able to extend the framework using semantic techniques and the WordNet ontology as a corpus. It is also possible to extend the ontology used with domain specific ontologies, in order to provide a more specific description to a term that WordNet is unable to provide [1]. This form of term expansion fits into the idea of Linked Data, in particular the idea of finding a means of linking information from separate Semantic Web ontologies in order to find an answer to a complicated question [59].
Chapter 3: Analysis of the Academy QA System

This Academy QA system focuses on three primary areas: query parsing, answer selection, and text corpus. This does not necessarily entail a full QA system, but rather that the components are functional to the point that the system can 1) parse the type of fact that the user is searching for, 2) build a query based on that information, and 3) select an optimal answer if the initial query, or set of queries, has multiple results.

The thesis makes the assumption that the user has access to a prebuilt ontology and any ontology that it is built upon. In this case, the Academy ontology is used, which can be found here [29]. This is a sample ontology that keeps a catalog of courses, students, professors, and tutors for universities. Currently within the system is a sample university, based loosely on CSUCI during the Fall and Spring semesters of 2008 and 2009, and a Community College, based loosely around Moorpark College around the same period of time. This is expanded upon in the following chapter.

With an ontology in place, an interface for users to submit their questions is necessary. The focus is to keep as much of the back end out of the mind of the user as possible, such that beyond the basic constraints of entering an interrogative statement, the user is free to ask any question. As a result, there is not much else needed for this system beyond a command line interface, where the user submits a question as input, and the answer is returned to them through the output. If we were to implement this on a Website interface, it might closely resemble the input of a Google search, wherein questions are submitted via a simple text box followed by a button click to signify a post back event, after which the answer is presented in a dialogue box [67].

One complication is that the user may not necessarily ask questions that are relevant to the ontology. For this reason, it is also assumed that the user has enough knowledge of the ontology that the questions being asked are done using terms that exist within the ontology; in other words, the user understands that they are asking a question relating to a university, since that is the ontology they’ve loaded in the first place. Much like how a search engine works, if there is not data within the corpus regarding the question, then no results can be returned. The result is a system that does not parse true natural language, but for all intents and purposes of this system, true natural language is not necessary.

The primary approach taken for each step is described in the subsequent sections. Section 3.1 discusses how questions are classified and parsed into the ontology, using techniques from Schlaefer[1]. It is demonstrated how queries are built, with the result parsed into SPARQL[12]. Section 3.2 discusses the foundational and domain ontologies used in the system, justifying the reason for the use of the DOLCE ontology, and how it is used to augment the Academy ontology.

3.1 Question Answering System Analysis

The demonstration of the feasibility of a QA system based on Semantic Web ontologies for this thesis included the construction of a Academy QA system that illustrates an application of the technology for answering practical questions in a typical domain. In the previous chapter, we discussed areas of consideration for a QA system - natural language, domain, question
classification, query parsing, search engine, and answer selection. For this section of the thesis, each of those areas are examined as they pertain to the Academy QA system, with a focus on question classification in section 3.1.1, query parsing in section 3.1.2, and answer extraction in 3.1.3.

The remainder of the section is focused on the Academy QA system's internal architecture, and how the system pertains to the user. Section 3.1.4 discusses the system architecture along with a basic user flow. Section 3.1.5 expands on the user flow and discusses the use cases of the system.

### 3.1.1 Question Parsing

The system responds to questions generated by a typical academic user with a very simple introduction to the system. When the user accesses this system, they are prompted to ask - type in - a question. The sort of question that is asked by the user can be arbitrary, as long as it begins with an interrogative phrase, and refers to any topic pertaining to the domain; however, it is not expected that all of these lead to a satisfactory answer. This system employs what is known as a relaxed version of a controlled natural language. That is, a subset of natural language that have a restricted vocabulary and grammar to avoid ambiguity or complexity, and improve focus on either human reading or logical context [29]. For this part, it is necessary to create a controlled natural language that relies on a vocabulary of terms common in the academic domain, be human readable, is simple to interpret, and readily translates into a machine readable query.

The system is able to answer factoid questions, list questions, and definition questions, as they were described in Chapter 1. In addition, the system is able to limit the query based on the type of factoid question asked, as is done with EphyraQA[1]. If a user were to ask a question with the interrogative pronoun "who," the target of the question can be assumed to be a sort of person. This provides context for ambiguous natural language questions; for example, the answers to the question "who makes pizza" may exclude the ingredients necessary to make a pizza, as well as lists of pizzas.

The goal for a refined version of this system is to transform arbitrary natural language questions from the user into parseable logical statements. This process has been described by Schlaefer et al [1] as including the following features for transforming a question into a statement:

1) the interrogative pronoun (e.g. who, who is/define, what, when, where, which, how many)
2) the type of adjacent phrases
3) the expected answer type

The interrogative pronoun is not something that is typically "ontological," so, rather than wrestle with significant representation issues, interrogatives are handled within the Java program. The adjacent phrases are parsed via a series of subqueries into the ontology to determine if the words are actually a part of the ontology. Any words that are not in the ontology, that are not also an interrogative pronoun, are discarded for the purposes of this experiment.
From this set of recognized terms it is possible to determine the expected answer type based on the properties and classes contained in the statement, combined with the interrogative pronoun. For example, consider the question “How many computer science courses are available in Fall 2009?” From the pronoun, we can determine that the user is requesting a list count of something, and in the adjacent phrases we can see “computer science courses,” “available in,” “Fall 2009.” From this, we are able to see “computer science” is a type of academic program, “courses” is a reference to academic course, and “Fall 2009” is a reference to the catalogue year “Fall 2009.” The expected answer type, therefore, is an integer number representing the list count of courses within the computer science academic program within Fall 2009.

<table>
<thead>
<tr>
<th>Pronoun</th>
<th>Question Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who</td>
<td>Factoid</td>
<td>“Who composed The 1812 Overture”</td>
</tr>
<tr>
<td>What</td>
<td>Factoid</td>
<td>“What mountain is the tallest on Earth”</td>
</tr>
<tr>
<td>When</td>
<td>Factoid</td>
<td>“When was the Magna Carta issued”</td>
</tr>
<tr>
<td>Where</td>
<td>Factoid</td>
<td>“Where is the largest castle in France”</td>
</tr>
<tr>
<td>Why</td>
<td>Complex</td>
<td>“Why is the sky blue”</td>
</tr>
<tr>
<td>How</td>
<td>Complex</td>
<td>“How does a car work”</td>
</tr>
<tr>
<td>How many</td>
<td>List (count)</td>
<td>“How many buttons are on a standard keyboard”</td>
</tr>
<tr>
<td>Which</td>
<td>List</td>
<td>“Which movies grossed higher than 10 million in 2012”</td>
</tr>
<tr>
<td>Who is</td>
<td>Definition</td>
<td>“Who is George Washington”</td>
</tr>
<tr>
<td>Define</td>
<td>Definition</td>
<td>“Define gravity”</td>
</tr>
</tbody>
</table>

Figure 3-0. Table of interrogative pronouns. Those shaded in gray are not used in the Academy QA system, but may be as future work.

3.1.2 Query requirements

When key phrases and context have been extracted from the initial query, the next step is to form a query to the ontology. To determine this, the system examines the relationships between the classes selected earlier, as well as the properties selected. Based on the associations between the two, the system may be able to filter the number of possible questions, making the eventual query easier to determine. The system also takes into consideration the interrogative pronoun, as this filters the number of relations that are necessary for consideration as well; for instance, if the user asks a “when” question, then the system can filter the number of properties down to those associated with a specific date and time.
The system can run into scenarios where multiple interpretations of the same question are considered to be valid queries by the ontology. Without proper handling, this can result in the system attempting to resolve more than one query, if the terms used by the user are ambiguous. In the Academy QA system, ambiguities are resolved, rather arbitrarily, by the ontology itself. In advanced QA systems, various options exist to these issues. Logical inference engines, or "reasoners" may be employed to infer the proper response, or reaction to such a situation. Based on statistical weighting, it is possible to select a question based on a similar example - i.e. case based reasoning - as seen in the e.g. EphyraQA system[1]. It is possible to add a step in between to give the user a means of clarifying their terminology by selecting a specific query, but this system attempts to resolve the answer with no callbacks from the user, as is performed in EphyraQA[1]. As in EphyraQA, if the user attempts this and receives an incorrect answer, it is simple to ask the question again with more specific terms.

3.1.3 Answer Extraction

Once we’ve run the query on the ontology, we retrieve a list of answers from the ontology. Since the Academy QA system is focused primarily on answering factoid questions, it is likely that only one of these answers is valid. Therefore, it is necessary to have a system to determine which of these answers are correct as well. For the Academy QA system, this is handled arbitrarily by selecting the first answer retrieved by the query.

3.1.4 System Architecture

Consider a scenario in which a user asks the system a question. On the front end, or at the user interface level, the user is prompted to type in a question, then submit that question. The application then splits the question, and evaluates the individual parts of the question to find the interrogative pronoun and adjacent phrases. The interrogative pronoun is parsed at the application level, but in order to determine the relevant phrases, the application then creates a series of SPARQL queries to find classes or properties whose labels match the phrases in the search query. Once these are created, the SPARQL query is then sent to the ontology, and the result returned to the front end is processed such that the result is readable by the user.

Looking a step deeper at when the system has parsed phrases and found terms that indicate their relevant classes, it then constructs another set of SPARQL queries to retrieve an answer to the question from the ontology. The classes and properties returned from the initial question, as well as the interrogative pronoun, are used to form this query. This is similar to the classification based on interrogative pronoun implemented by EphyraQA, with the primary difference being the resulting query system that this is built into uses a more rudimentary type pattern system[1].

If the ontology returns a single answer, then that is returned to the front end so that the user can see the answer that the system has selected. If possible, this answer does not involve the term as is used by the ontology itself, but rather a synonym, jargon, or related phrase represented as a class label or descriptor defined by the ontology to differentiate between the descriptor used
in the ontology, and a human-readable descriptor. An example of this is the “label” class defined by RDF Schema [57].

In figure 3-1, below, we can see the system architecture diagram for the Academy QA system. A more detailed description of the specific frameworks used in this system can be found in Chapter 4. For now, the different Java libraries [30], Jena [38], and ARQ [40], is used for the front end, with the Java IO libraries used to handle the direct interpretation and Java utility libraries to split the user input, Jena to construct a model of the ontology that is usable through Java, and ARQ handles passing SPARQL queries into the ontology and returning a list of responses that are classified through Jena’s model of the ontology. Jena’s model of the ontology are then used to access the human readable resources, which are passed back to the user using the Java IO libraries [30].

![Figure 3-1. System Architecture Diagram for Academy QA System.](image)

### 3.1.5 Use Cases

The 3 primary use cases for the user in the Academy QA system are shown in figure 1 below. These cases are to select the ontologies they want to query, the questions they intend to submit, and to eventually select a query to run against the ontology. This also includes the different question types, which are specified by the interrogative pronoun that the user identifies when they submit a question.

We’ve discussed the concept of feasibility as it applies to the Academy QA system, in that the system must be able to adequately answer questions provided by a user for the ontology. The Academy QA system focuses on a university catalogue, which is discussed in section 3.2.2, so it is reasonable to construct use cases for a student or professor that has access to this ontology. To demonstrate this, a list of sample questions to be used are detailed in chapter 4, and the results of these questions are discussed in chapter 5.

To start, the user needs a way to be able to specify the ontology being entered into the system. In order to answer specific factoid questions, such as “who” and “when,” the ontology must also extend the foundational ontology DOLCE [25]. This ontology is discussed in section 3.2.1.

From here, the user needs to be able to submit their question to the ontology. As discussed in section 3.2.1, this consists of an interrogative pronoun, followed by the pieces necessary to form the remainder of the question type. The users ability to select a query is limited
to the interrogative pronoun that the user chooses in their question, which determines both the type of question they wish to answer, and the result that is returned.

![Use Case Diagram](image)

**Figure 3-2.** Use Case Diagram. Last case is handled by Academy QA system for now.

Figure 3-3 provides a distinctive breakdown of how the user proceeds through the Academy QA system. In order for the system to work, an ontology must be loaded, so a model is constructed for the ontology immediately after loading it into the system; this acts as a means of verifying that the ontology specified by the user exists. From there, the user asks their question, determining which question type is used via decomposing the statement and parsing the key phrases into the system or ontology. From here, the SPARQL query is formed; if there are multiple queries, the first solution is select to find an answer. The ability to specify which query is most appropriate to select an answer for a user is too presumptive of the user’s knowledge of the back end, but may be possible for future work. Afterwards, the results of the query are extracted from the ontology, and, if necessary, an answer is selected, interpreted, and returned to the user. If further questions need to be asked, the system returns to the question step.

![User Flow Diagram](image)

**Figure 3-3.** User Flow Diagram
3.2 Ontologies for the Academy QA System

To demonstrate the use of the Academy QA system, an ontology was constructed and linked to a foundational ontology. In order to ensure that certain concepts, such as determining what persons, places, or things are, classes are referenced directly from the foundational ontology in the system when answering questions, but any lower level ontology references are generated on the fly, such that any ontology using this foundational ontology can be referenced to answer questions. This allows the system to expand its semantic network, providing a possibility for answering a greater number of specific questions [1].

Section 3.2.1 describes the foundational ontology which is used to create the ontology used for the Academy QA system, known as DOLCE. Section 3.2.2 discusses the ontology built for the Academy QA system.

3.2.1 DOLCE

Ontologies in computer science are often components of QA systems and are used for interpreting natural language sentences in other applications. Ontologies also constitute the basic infrastructure for the Semantic Web. With this in mind, we examine a notable Semantic Web foundational ontology, DOLCE, and determine its use for creating a Semantic Web ontology capable of being used in a QA system.

DOLCE is a foundational ontology developed under the auspices of the European Union Wonder Web Project as the initial module of the Wonder Web Foundational Ontology Library, as well as to provide a starting point for building new ontologies [25]. It is a formal ontology, in that is defined by axioms in a formal language, in this case, First Order Logic and represented in both Knowledge Interchange Format (KIF) and OWL DL. A simplified OWL DL version, called DOLCE Ultra Lite, also exists and is suitable for most current Semantic Web applications. A full description of the ontology can be found at [48].

DOLCE is an attractive choice for the Academy ontology, used in this thesis, because DOLCE is an ontology of cognitive artifacts, focused on capturing categories related to natural language and human commonsense [48]. Everything that can be defined through this ontology is something that has an explicit cognitive backbone, something that can be used both by the computer to associate a relative meaning to every term without necessarily understanding what the term itself is, and by humans to more easily provide descriptors to the objects underneath these categories.

In addition, DOLCE has a long standing association with Princeton’s WordNet Lexical Ontology project. In particular, a current project called OntoWordNet is underway aiming to align the two [52]. Apart from the obvious implications with respect to DOLCE’s suitability for Natural Language Processing systems, this association also make’s WordNet’s extensive collection of synonyms (“synsets”) readily available to systems that use DOLCE. A direct consequence of that is a system that can recognize a wide range of linguistic terms related to precise ontological concept names, and that should result in a more flexible, robust user interface.
Finally, DOLCE is a minimalist foundational ontology that aims to include the most reusable and widely applicable upper-level categories. The minimalist approach is particularly important, because it means there’s less obfuscation when it comes to determining what objects fit under which categories. As an example, consider the table in figure 3-4, below, which describes the categories “non-agentive social object” and “process” in DOLCE. Under the definitions provided for these objects, it is clear that, if we were looking at this from the context of a university, a “non-agentive social object” refers to a university class, while the “process” refers to its sections. This relational approach to defining objects is crucial for building into a specific domain for an ontology, and DOLCE provides an excellent head start.

<table>
<thead>
<tr>
<th>&quot;Leaf&quot; Basic Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Quality</td>
<td>the value of an asset</td>
</tr>
<tr>
<td>Abstract Region</td>
<td>the (conventional) value of 1 Euro</td>
</tr>
<tr>
<td>Accomplishment</td>
<td>a conference, an ascent, a performance</td>
</tr>
<tr>
<td>Achievement</td>
<td>reaching the summit of K2, a departure, a death</td>
</tr>
<tr>
<td>Agentive Physical Object</td>
<td>a human person (as opposed to legal person)</td>
</tr>
<tr>
<td>Amount of Matter</td>
<td>some air, some gold, some cement</td>
</tr>
<tr>
<td>Arbitrary Sum</td>
<td>my left foot and my car</td>
</tr>
<tr>
<td>Feature</td>
<td>a hole, a gulf, an opening, a boundary</td>
</tr>
<tr>
<td>Mental Object</td>
<td>a percept, a sense datum</td>
</tr>
<tr>
<td>Non-agentive Physical Object</td>
<td>a hammer, a house, a computer, a human body</td>
</tr>
<tr>
<td>Non-agentive Social Object</td>
<td>a law, an economic system, a currency, an asset</td>
</tr>
<tr>
<td>Physical Quality</td>
<td>the weight of a pen, the color of an apple</td>
</tr>
<tr>
<td>Physical Region</td>
<td>the physical space, an area in the color spectrum, 80Kg</td>
</tr>
<tr>
<td>Process</td>
<td>running, writing</td>
</tr>
<tr>
<td>Social Agent</td>
<td>a (legal) person, a contractant</td>
</tr>
<tr>
<td>Society</td>
<td>Fiat, Apple, the Bank of Italy</td>
</tr>
<tr>
<td>State</td>
<td>being sitting, being open, being happy, being red</td>
</tr>
<tr>
<td>Temporal Quality</td>
<td>the duration of World War I, the starting time of the 2000 Olympics</td>
</tr>
<tr>
<td>Temporal Region</td>
<td>the time axis, 22 June 2002, one second</td>
</tr>
</tbody>
</table>

Figure 3-4. Description of Tables in DOLCE [48].

3.2.2 Academy Ontology

The Academy ontology [29] is an ontology of concepts related to institutions of higher learning (e.g. universities, colleges). The initial version was intended to support queries related to an individual student’s academic requirements with respect to a particular course of study. This ontology imports, DOLCE Ultra Light (DUL) in expectations of improved interoperability with systems and other ontologies that reference or are aware of DOLCE. In this thesis, we have extended this ontology to suit the needs of the Academy QA system. The Academy ontology is represented in OWL DL, and developed with the Protégé ontology editor. This is detailed further in section 4.1.1.

The ontology provides a simple framework for setting up queries, and also fulfills the basic requirements for question submission on the bases for list questions, definition questions, and factoid questions pertaining to persons, events, and social objects as they pertain to the
university. For example, the classes “professor,” “student,” “tutor” and “administrator” can be queried upon when asking questions with the interrogative pronoun “who.” The classes “course,” “course offering,” “degree,” “major,” “minor” and “record” can be used for the interrogative pronoun “what.” The property “hasEventDate” is referenced when asking questions with the interrogative pronoun “when”[25]. Any of these can be used for the interrogative pronoun “how many,” which can be a reference to a count for a list question, or a sum when referring to a numerical unit of measurement.

This ontology was built in 2009 as a demonstration of Semantic Web concepts and how an ontology is used to retrieve information from a university catalogue using SPARQL. This ontology was built using DOLCE as a base, and was originally built with the assumption that the eventual application be used as a means of checking graduation requirements. As a result, many of the pre-existing classes were oriented around the concept of an academic “course,” something that had specific offerings each semester with a certain number of sections per offering, a number of academic units associated with the course, and specific “required for graduation for major” or “elective for graduation for major” properties. From this, I’ve framed a number of typical questions, which are discussed in detail in chapter 4. In addition, the ontology has been extended to include information regarding professors and students, as is illustrated in figure 3-5 below.

Figure 3-5. A series of queries and results. Top: students participating in courses; Middle: professor teaching specific course; Bottom: Grade student has in courses.
Chapter 4: Implementation of the Academy QA System

This chapter covers the software, tools, and implementation of the Academy QA system. The installation process and the reasons for choosing the software, as well as how it is used in the system, is discussed. Difficulties found in the implementation, and resolutions and workarounds to those difficulties are also discussed.

Finally, this chapter provides an analysis for the code used in the QA program. The control flow from the user’s side is examined, through the business logic, and through the data abstraction. When necessary, snippets of code or markup are included to ground the discussion.

4.1 Tools Used

This section is dedicated to the setup of the Academy QA system and Academy ontology. The applications, libraries, and programming/markup languages used to set up the Academy QA system and Academy ontology is covered. Section 4.1.1 discusses ontology building with Protégé. Section 4.1.2 discusses the choice of Java for building the Academy QA system. Section 4.1.3 details how the Eclipse IDE was used for set up and analysis during experiments. Finally, section 4.1.4 details the usage of Jena, the framework used to extract information from the ontology.

4.1.1 Protégé

Protégé [38] is a free, open source ontology editor and knowledge-base framework. It can be used to create and update OWL ontologies. Protégé was developed at the Stanford Center for Biomedical Informatics Research, initially to support human genome research. Over time it has evolved and become almost a standard ontology editor. Protégé provides a means to create and extend the Academy ontology in a way that allows the developer to specify and immediately visualize classes, properties, and individuals in a graphical interface. It also offers a variety of tools and features to help ensure that the ontology is working properly, including a panel to create and run SPARQL queries on the ontology, a built in reasoner to test logical inferences, consistency checks to ensure the logical integrity of the ontology, and a “to do” list to check notes made in sections that have not been finished yet. It also comes preloaded with essential OWL specification classes, RDFS, and RDF, so associated properties can be accessed as soon as an ontology is created and includes built in features that ensure compliance with the W3C OWL specification.

At the outset of this thesis, OWL 2 had just become a W3C standard, and the technical risk in waiting for a number of Open Source projects to develop stable adaptations to the new standard did not appear to be balanced by the advantages offered to this thesis[15]. Therefore, the version of Protégé that is used for the Academy ontology is 3.4 RC1. Protégé 3.4 RC1 can be found at [38], and along with basic hardware requirements, requires Java 1.5 or higher. Although these are all backward compatible changes, the newer versions do not offer improvements germane to this thesis.
Protégé can be downloaded at [38] and is easily installed on virtually any modern computer. Once Protégé has been installed, loading an existing OWL ontology, such as the Academy ontology, in Protégé can be done by opening the application and selecting the location of the ontology from the “File -> Open” menu. Protégé may access existing OWL ontologies at their specified URI (typically included in the ontology’s full name) or through a local version on the host computer. In this thesis, we operated on a local version of the Academy ontology. Once the ontology has been opened, adding and modifying classes and individuals is straightforward and generally intuitive. Detailed user instructions for Protégé are found in the Protégé User’s Guide available at [38].

4.1.2 Java

The Academy QA system, including the user interface, and the underlying business logic, was developed in the Java programming language. The principal reason for this choice is that the API necessary to access the interface, Jena, was created in Java. Although there are functional extensions of the API available in other programming languages, but the extensions do not receive the same level of support that the original version does, as the extensions are not officially supported by the team that maintains Jena. Protégé is also implemented through the use of Java [40].

The version of Java that is used for the Academy QA system is the Mac OS X version of JavaSE-1.6, and is made available through the Mac OS X developer tools found at [30]. Java is made available for a large number of operating systems, including Windows and Linux, so any system running Java 1.6 or later is capable of running the Academy QA system.

4.1.3 Eclipse

One of the most powerful and easily accessible IDEs for Java is Eclipse SDK, provided by IBM at [31]. The primary benefit for using an IDE for Java development is an area to maintain libraries and build paths without having to set up a separate file or script to run every time a specific project needs them. Other useful features include JavaDoc integration for quickly accessible class descriptions, a graphical debugger to receive immediate feedback on local variables, and modular extensibility to reduce the bloat that normally hinders one’s ability to write code.

The version of Eclipse SDK being used for the Academy QA system is 3.7.0. Hardware requirements aside, the primary requirement to run Eclipse SDK is a Java Virtual Machine, which we already have from the section above. Eclipse is able to run with at least Java SE 1.5. Once the installer is finished and we run Eclipse for the first time, we must begin by specifying an area on the hard drive where Eclipse stores our projects. This is known as our workspace. After that, we can create and import projects, classes, and libraries to our workspace. Packages and folders can be created to organize the project, though packages are handy for not necessarily altering class locations in the file system.
4.1.4 Jena

Apache Jena is a well known, mature, open source Java framework widely used to create Semantic Web applications. Jena was originally developed at the HP Labs, Bristol U.K. in 2000. In 2012, it became an Apache top-level project. Jena provides a collection of tools and Java libraries to develop Semantic Web and linked-data apps, tools, and servers [39].

Any and all access made to an ontology from the Academy QA system are made through libraries available thanks to Jena. Specifically, the libraries we use are the Ontology Model defined within Jena, which is an extension of the base Model class used for RDF models, and ARQ, which is a separate extension of Jena used for sending SPARQL queries to a model and parsing the results.

The version of Jena being used for the Academy QA system is 2.7.0, found at [40] which requires at least JavaSE 1.6. The version of ARQ being used for the Academy QA system is 2.9.0, found at [40], which requires both Jena and at least JavaSE 1.6. In addition, Jena requires the following components:

- Xerces2-j [33] xerces:xercesImpl:jar:2.10.0
- XML Commons External Components XML APIs [34] xml-apis:xml-apis:jar:1.4.01
- IRI [36] org.apache.jena.jena-iri:jar:0.9.0-incubating
- SLF4J API Module [37] org.slf4j:slf4j-api:jar:1.6.4

When the associated JAR files have been acquired and referenced in the Eclipse Project’s libraries, the Academy QA system is then ready to reference the appropriate Jena and ARQ libraries. Javadocs for Jena and ARQ are available to be referenced directly through Eclipse in order to receive immediate descriptions for Jena and ARQ classes and methods. The Javadoc for Jena can be found at [41], and the Javadoc for ARQ can be found at [42].

4.2 Implementation of the Academy QA System

This section describes the development of the Academy QA system. Section 4.2.1 discusses the analysis and development of the extended Academy ontology. Section 4.2.2 looks at the development of questions and corresponding SPARQL queries. Section 4.2.3 details the integration of Jena tools in the Academy QA System. Finally, section 4.1.4 discusses the implementation of the QA processing system.

4.2.1 Analysis and Development of the Academy Ontology

The domain of the Academy QA system was defined in the Academy ontology, and built using Protégé. The newly created Academy ontology imports the DOLCE foundational ontology, so that the creation of the lower level definitions are properly situated with respect to DOLCE.
The ability to add new concepts to an existing model, as opposed to creating a complete, consistent model from scratch, greatly simplifies and improves the development process. Other utility ontologies, such as the Time ontology [70] and toolsets, provided by default by the W3C, are included in every Protégé OWL project by default.

To add specifications for classes, properties, and other resources to the Academy ontology, the definitions of the categories in DOLCE [48] and comments provided by the Laboratory of Applied Ontology [25] were analyzed to determine which categorizations fit best for base classes such as students, professors, courses, catalogues, majors, and universities. From here, additional classes based on the information typically of interest to a university population were built. It was not necessary to, for example, fully emulate an existing course catalogue for this exercise, just to provide a sample rich enough so that answers are provided for the questions asked. To make some of the tests more realistic, the terminology for this ontology was borrowed from the course catalog of California State University, Channel Islands [69]. To indicate a wider scope, such as a difference between course specification for different universities, representative data from other universities was added over time.

4.2.2 Analysis and Development of Questions using SPARQL Queries

Consider a student searching for answers to questions about course requirements and availability for a semester, or a prospective student who just came across one of the universities in this ontology. One question they might ask is “what majors are offered at <university>.” From here, they may narrow their enquiry down to a major they prefer, and ask about the course work necessary, as in: “which courses are necessary for completion,” “which courses are considered to be electives;” and “how many academic units in total are needed?” These questions may be mapped to more formal questions such as: “which courses are requirements of <major>,” “which courses are electives of <major>,” and “how many academic units are required for <major>.”

Once this context is established, the student might start looking at course prerequisites, such as what prerequisites a particular course has, or what prerequisites that course fulfills, and ask the question “Which course is a prerequisite of <course>,” or alternatively, “Which course has the prerequisite of <course>.” For example, consider a course from the CSUCI course catalogue COMP 150, which has a prerequisite of MATH 105 and is a prerequisite for COMP 151[69]. From there, a student may search for what sections were being offered for a course, which involved searching through a specific catalogue year and semester, along with finding out information about those sections such as who the professors were. This was split into two questions: “Which sections are being offered for <course>,” and “Who is teaching <course section>.” A question that may follow is “When is <course section>.” In addition to properties about the course section, it is also necessary to know about properties related to the course section, such as the number of students already enrolled in the course, resulting in the question “How many students are participating in <course section>.”

For an administrator, consider a university administrator attempting to find out as much information as possible about a student, professor, or course. This involved asking a definitional question, rather than the list and factoid questions asked by the sample student, which necessitated questions such as “Define <course>,” “Who is <professor>,” and “Who is
In addition, there is some overlap in what a student is looking for, though there may be some curiosity in what a student is already participating in, resulting in the question “which courses is <student> participating in now.”

For professors, consider a professor attempting to determine if a student was eligible for a course. In searching for this, a professor checks for the courses that student has already taken, asking “what courses has <student> already taken.” In addition to this, the course may have specific grade requirements, which raises the question “what grade does <student> have for <course record>.”

After these questions were formed, it was necessary to form SPARQL queries corresponding to the original questions. The pattern chosen was simple:

1) Determine the type of question needed to be answered from the interrogative. With a factoid question, a single answer needed to be verified against that value in the ontology, while a list question requires that multiple matches were accurate. A self-contained corpus is invaluable in this regard, in that it provides a simple means of fact-checking the responses provided by SPARQL - just look up the necessary references in Protégé. In the case of a list count or summation, associate a secondary variable to ensure that the list count was accurate - e.g. for determining academic units, have the query return the academic course name.

2) Determine the target of the question. For a factoid question like “Who is teaching <course section>” - for sake of this example, assume “<course section>” is “COMP 101” - this is clear from our own perspective, but this is not necessarily the case for a system that would be building this query on its own. In this case, rather than choose to search for an “academic professor” who “is a teacher of” “COMP 101,” search for something that’s a subclass of “a natural Person” who “is a teacher of” “COMP 101” since it is possible to reference the DOLCE category “natural person” in the Academy QA system.

3) Determine how to fit in the remainder of the statement. Each resource type has its own effect on how the eventual SPARQL query can be interpreted. Taking our previous example, “natural person” is a class in the DOLCE ontology, so a statement can be formed saying that the result is something that is a subclass of “natural person.” The next phrase, “is a teacher of,” is a property, and so we can use the same result variable and begin the next statement saying that the result is a teacher of a new variable. In this case, the new variable can be connected to either another class, or individual, while if the next part is another property, then a new statement is formed based off of this new variable instead. In this case, “COMP 101” refers to an individual, so the variable “something” can be replaced, or declared to be, “COMP 101.” If this were to be extended, for example, if someone were to ask “who is a teacher of COMP 101 in 2009,” this chain would continue, linking the new variable to some property containing the date value “2009.”

The final query can be seen in figure 4-0 below. In addition to the above, literal datatypes, such as strings, integers, and dates, can be interpreted using “FILTER” with the result and whatever the filter is shown to be.
4.2.3 Application of Jena tools

Once the ontology has a structure to interpret the representative SPARQL queries, such as the ones listed in section 3.2.2, it is possible to begin construction of the Academy QA system. To use SPARQL in the Academy QA system, Jena has a SPARQL query module, ARQ [42]. The simplest way to determine the transition between the questions presented above, and the SPARQL queries necessary to answer those questions, is to have the program also generate queries in SPARQL. This, along with the interpretation of the interrogative pronoun, determines how answer extraction is handled in the Academy QA system.

It is necessary for the system to load the ontology into the Academy QA system in order to parse items from the user’s question into the ontology. This is handled by the Model Factory built into the Jena libraries, which has a method for reading ontologies into a class instance that can be used without rereading the same file multiple times. Jena can also read the ontology from both local file locations and URIs, allowing access to the ontology from an online resource. To provide the necessary information for loading these resources, a method was created to retrieve the ontology URIs from a file, and set them as parameters for Jena’s Model Factory module. To make queries more readable in the debugger, the Academy QA system also has a parameter to specify the prefix necessary to label specific URIs in SPARQL.

Along with prefixes shown in figure 4-1, it is necessary to establish built-in prefixes for the base W3C RDF, RDFS, and OWL specifications. These are necessary for parsing SPARQL queries in ARQ, since while Protegé provides these queries as prefixes by default, Jena does not. To prevent the user from needing to include these documents, these prefixes are loaded by default in the Academy QA system. To ensure that these are recognizable in the debugger, their acronyms are used, as is demonstrated with RDFS in figure 4-2.
4.2.4 Development of the QA Processing System

The initial step is to provide the user with a means of entering their question into the system. The Java libraries necessary for basic console command input/output are sufficient to retrieve questions from the user[30]. A simple prompt for input is advisable, but not necessary. Once user input has been received, it is necessary to parse information from the query into the ontology. This is accomplishable if the resource has some form of natural language designation, such as RDFS label [57]. The RDFS label class is an effective tool here, because any resource using RDFS can have any number of labels, and labels also have a property to specify languages, providing a means of filtering labels by dialect. Once a property for natural language designation is specified, the next step is to parse the natural language question itself. While it is necessary to have the program parse the first word or phrase in order to extract the interrogative pronoun, the words following that can be extracted one at a time and run through a SPARQL query to retrieve the URI of the object. Using RDFS label as our property, the query looks like this:

```sparql
SELECT ?result WHERE { ?result rdfs:label ?object. FILTER(REGEX(?object, \'<input phrase>\')) }.
```

This returns the URI of the object, which is then located in the ontology and parsed into the Jena model in order to determine what sort of resource it was. Since the resource type was important for generating a statement, a “node” class was created to store relevant information, containing the resource’s URI, SPARQL prefix, resource type, and classes to parse information out of URI tags.

It was then possible to write the logic to translate these phrases into a SPARQL query. As an example, the approach first involved attempting to answer factoid questions that involves checking the answer against the initial statement, then give an answer “Yes” or “No.” This helps to determine if the query logic was capable of parsing an easily interpretable result, but this is also redundant with the debugging tools already built into Eclipse [31].

It is more simple to approach how to answer individual types of queries, then break it down into individual parts from there. In the end, the logic to determine how to answer list queries worked out like so:

1) Check the initial question phrase to determine if it gave a resource. If so, this is designated as the target. Add a statement to pair the result to the query node, then iterate on the query node for the next statement. If not, assume a general answer type.
2) Iterate through each of the remaining phrases. Change logic based on the following:
   1) Class - If the last phrase was a property, then replace the last query node with the class. Otherwise, create a new statement declaring the current query node to be the class, and iterate to the next query node.
   2) Property - Create a statement declaring the current query node to be the next query node.
   3) Individual - Same as the class, except if the last phrase was not a property, filter setting the node equal to the individual instead of declaring the query item to be of type individual. The individual is not a type.
4) Literal - Filter based on the assumed type of the literal. Check against dates if it’s a date, integer if it’s an integer, or run a case sensitive regex match if it’s a string.

3) Construct the query, inserting the information parsed together from the phrases into the where clause.

From here, the result involved taking the label query shown above, only instead of matching on the label, to return the label back. If a label is not found for the object, the base class name was shown instead. This is not necessary in an ontology that has all of its label constructs defined, but since it is possible for ontologies to link to resources that is not defined from within the ontology itself, this allowed for queries from those ontologies to continue working. For example, the RDF and RDFS definitions where label is defined in the first place do not have labels on their own properties.

For list count queries, the logic is consistent with the list query mentioned above, but it is possible to be more specific if a user is posing a question concerning “how many” of an object. For example, if the target result is a number, then it is likely to require the use of a summation, rather than a count. This works well for questions that had an answer type like “academic units,” such as the “how many academic units is required for <major>” question.

For factoid queries that required a specific type in DOLCE to be referenced, like “Who” and “When,” the logic for the list query above also applied, but it was also necessary to add a statement to ensure that the result was also a person for “who,” or an event time for “when.”

For definitional questions, it is assumed that only one item is defined at a time for a definitional question; in other words, this system currently assumes a single individual is asked about. From here, it was necessary to write a query with two results: one for every property associated with the individual, and one for the object that the individual was associated with. The query looked like this:

```sparql
SELECT DISTINCT :result :result2 WHERE { <node> :result :result2. }
```

One concern with the system up to this point is the depth of the query in the current system. For example, this system allows us to add additional conditions after the original ones while creating a syntactically sound SPARQL query, but the current query system does not allow for the initial assumptions of how the system works. The “who” factoid question type described above demonstrates a means to check foundational classes until it either hits the DOLCE class for “NaturalPerson” or “SocialPerson,” but it is also possible to implement this through a

![Figure 4-3](image)

Figure 4-3. The “Who” restriction to ensure that all results were a person.
recursive query call on the individual object until it either finds that the object is a “Person” or an “Entity” - that is to say, not a “Person.” In order to safely do this using SPARQL, it is necessary to implement an algorithm that runs a SPARQL query for every superclass of the resource in question, then every superclass of those classes, until either “Person” or “Entity” was found. This leads to performance issues if the level of depth of the proper solution - or “Entity” if there isn’t one - and the number of superclasses along the way is high.

To avoid this problem, the Academy QA system only implements it to a single level of depth, adding only one additional query to the process. In addition, since the back end was readily accessible, and the objects involved in the questions described did not need multiple descriptors (e.g. “what is the average grade of every student that has taken COMP 150 from Summer 2009 until Spring 2012”), many of the questions described above have established property links between the asked resources. It is more reasonable, and efficient, to establish those relationships explicitly, than it is to traverse the ontology in order to find those relations.

![Figure 4-4. Final implementation of CLASS property.](image-url)
Chapter 5: Analysis of Results

The ontology and Academy QA system have been used and evaluated with respect to question types, and particular examples, generated during the course of the analysis and design of this system. At first, the CSUC1 Course Catalogue was used as a source in populating the ontology. As work proceeded on the Academy QA system, it became clear that information from the partially populated ontology was sufficient, and that a complete representation of the catalogue and university personnel data was not necessary for a demonstration of feasibility.

This constrained development approach is consistent with most software Academy development, and somewhat typical of the emerging practice in the Semantic Web. In the first place, fully populating an ontology can quickly become a distraction from the primary development goal of a Question Answering system. Secondly, “fully” populating any ontology is an lengthy and, arguably, unending task. The art, is, of course, to provide enough structure, logic and data to support development goals and then add to it as the system matures and new needs emerge. When that occurs, if the ontology is well designed, scaling up is straightforward and predictable. The following results were derived from, and can be checked against, the Academy ontology, which can be found at [29].

5.1 Analysis of Experiments

The question categories described in chapter 3, figure 3-0 were used to examine the question analysis techniques discussed in chapter 4. Section 5.1.1 introduces the test set, and how those demonstrate a feasible system. Section 5.1.2 reviews some of the limitations of the system with some questions that did not return a correct answer.

5.1.1 Test Set

The test set was derived from questions that seem typical of a student using the Academy QA system to retrieve course information. The system can be considered feasible - that is technically tractable and useful - if the system is capable of providing answers that can be considered reasonable and accurate answers concerning the natural language questions relating to a particular topic.

In the first set of questions shown in figure 5-0, one of the questions involved the question asked in 4.2.2: “Which majors are part of the CSUC1 curriculum?” Since the system relies on a controlled natural language, rather than a true natural language such as the ones used in the sample question set, it is expected that the eventual questions are not going to match exactly the questions as presented. In the case of the Academy QA system, the Academy ontology does not define certain linguistic concepts, such as adjectives, adverbs, or articles of the English language, so the questions presented to the system are asked without these words. This is in line with other ontology-based QA systems, such as EphyraQA[1], which are able to accept a question in true natural language, but strip articles out of the sentence that do not contribute to the associated phrases.
It was necessary to provide more information about the relation between the major and the university for the ontology to parse the proper information. The system needed information stating that the major was a subset of academic information related to the university, first. Once this was accomplished, it was possible to check which majors were not only a part of the sample CSUCI university, but it was also able to distinguish between majors from different universities, as the query “which academic major is academic information of Moorpark Community College,” was able to demonstrate.

![Figure 5-0. Academic majors.](image)

The result of asking the question of “Which courses are electives for CSUCI Computer Science Majors” is displayed below. As shown, the question does not deviate from the original question proposed, except to remove pluralization from some of the words used in the original question. The result is a query that searches for a result that is a course, and that is an elective of the CSUCI Computer Science Major resource. As a result, this gives an answer that reflects all of the elective courses for the CSUCI Computer Science Major in the academy ontology, and nothing else. This is similar for the question “Which courses are required for CSUCI Computer Science Majors,” which returns all of the courses that are required to graduate for a CSUCI Computer Science Major that are stored in the academy ontology. Both of these questions, and their answers, can be seen in figure 5-1.
Figure 5-1. Is elective, is required.

For the next question set, questions of the form “what are the prerequisites of Comp 102” from earlier is answered. Pairing the relation with its inverse in order to answer a similar question, “what prerequisites do Comp 102 satisfy,” which is also shown in the picture below. In the former’s case, the only prerequisite for COMP 102 is COMP 101, and the only prerequisite that COMP 102 satisfies is the one for COMP 150. This is reflected as such in the ontology, and so the answer is accurate.

Figure 5-2. Is prerequisite of, has prerequisite.

For the question, “How many academic units are required for CSUCI Computer Science Majors,” the “How many” qualifier is taken from the question, and from there parses the “academic units” property. This is, by definition, an integer datatype, so the Academy QA system is able to provide a sum, rather than an integer. When the final answer is given, it is not the same as asking “how many courses are required for CSUCI Computer Science Majors,” which is only able to provide a count of the courses required. This also works at an individual course record level, where if you ask “How many academic units for COMP 101,” it responds with “3.”

Figure 5-3. Numerical queries: How many academic units, how many courses.
In figure 5-4, the question “How many sections are there for COMP 101” is treated differently, because we cannot look at how many sections there are for COMP 101 on its own; the question is too vague. Instead, by searching on the COMP 101 section offering in 2009, which is where our test data is located, we are able to see that there are two COMP 101 sections for the Spring 2009 catalogue offering for COMP 101. With these course sections in mind, we can now ask more in-depth questions concerning the students and professors for those courses.

The next questions, “Who teaches COMP 150 Section 1” and “Who teaches COMP 150 Section 3,” demonstrate a particular factoid question that only provides a single result for each answer, as there is only one professor for COMP 150 section 1, and a different professor for COMP 150 Section 3. Likewise, this is a factoid question that filters based on people. Since we specified we are asking about a person with the “who” qualifier, even with a vague question, the answer is based on people only. As a result, this question can be asked in a number of different ways, while returning the same answer. In the figure, the question “Who is required for CSUCI Computer Science Major” is asked. With the “which” interrogative, this question returned a list, but with the “who” interrogative, it is empty.

There is also a limited system for definitional answers, as demonstrated in the “Who is AJ Bieszczad” and “Who is William Wolfe” questions. When these questions are asked, a full list of all information related to these two individuals, as defined in the ontology, is returned. This is not as apparent for the professors, since there is a limited working set of information related to course sections in the ontology as of this writing; however, when the same question is applied to
a student, like “Who is Joseph Donuts,” it is apparent that a lot of information can be returned by this query. Clearly, with a simple design effort, composite results, such as these can generate a more readable summary of the result set, and, perhaps offer users an option to view the entire list.

![Image](image_url)

**Figure 5-6. Who is/Define.**

Much like questions that use the interrogative of “Who,” questions of the type “When is COMP 101 Section 1” returns event information tied directly to a span of time. In particular, a question concerning COMP 101 has a lot of information related to it, as the “Who is” definitional question for COMP 101 demonstrates. This query can be effective for getting specific date time information from the ontology. To demonstrate this, figure 5-7 splits the above question into three queries: one to retrieve the course period’s start time, the end time, and the weekly meeting days. Note that a “when” query does not retrieve information concerning a person. Again, it must be noted that a “when” query is geared towards providing a single answer, and since weekly
meeting days is a list of week days, it can only be retrieved using a list query such as “Which” and “How many.” This does not mean that the “when” query is unable to return results that are week days, as the last part of figure 5-7 demonstrates.

Figure 5-7. When.

Figure 5-8 demonstrates how to retrieve grades for students in answering the questions “which course records belong to Joseph Donuts,” and “What is Joseph Donut’s grade from his COMP 150 Course record.” While the naming convention for records remain consistent, it requires the name of the record itself in order to find a grade.

Figure 5-8. Has grade.

The query “Which courses is Joseph Donuts participating in” makes use of a phrase used by the foundational ontology, not to see what courses Joseph Donuts has, but the courses that Joseph Donuts is actively participating in. As is demonstrated by the figure, Joseph Donuts is
only participating in Comp 101 for the moment. The query used here follows the original natural language question closely, and gives us a correct answer based on what is in the ontology.

![Figure 5-9. Is participant in.](image)

Figure 5-10 demonstrates the query “How many students are enrolled in Comp 101 section 1,” which also returns the correct number of students enrolled only in that particular course section. In addition, figure 5-9 also demonstrates the use of a synset. “Is participant in,” “is taking,” and “is enrolled in” are all synonymous to each other with respect to the question asked, allowing for a greater number of options in the type of question asked. Though it is not apparent in figure 5-9, “has student” is also synonymous with “has participant” in the ontology with respect to the question asked.

![Figure 5-10. Sysnets, how many is participant in.](image)

5.1.2 Limitations

Some of the difficulties with the ontology involve the lack of defined synonyms for arriving at a specific answer. For example, in the question “which majors are part of the CSUCI curriculum” above, the question returns the expected results, but if we were to ask a similar understandable question of “Which majors are offered by CSUCI,” the system does not understand what to do with it. The words correspond with the previously described interpretations, in that by typing in “which,” the type of question was determined as a list count question, and the remaining words were used as conditions in the resulting answer statement.
The ontology does not contain a property to properly represent “offered by” yet, so the query parser is unable to interpret what this phrase means. In the end, it is possible to retrieve this listing through the ontology, and it certainly is feasible to do using other phrases in the system, but with the current system implementation it is not likely that someone, without training on the system, understands the proper question formation protocol necessary to receive an accurate answer. This difficulty can be overcome when the ontology begins to introduce additional means of identification for university objects.

It is also clear that there are certain areas that can be directly expanded upon from the questions generated. For example, there is not currently a course section capacity datatype defined, but in the future it is reasonable for a student to ask a question like those in figure 5-10, along with a question of “what is the capacity of Comp 101 Section 1” and receive a response to determine course availability. Outside of the Academy ontology, the default properties for RDF and RDFS do not provide labels through our ontology. This is problematic for results to “who is” or “define” questions. To avoid lost answers, the system is built to provide a label if one is found, or the direct resource name otherwise. This is resolvable without assistance from the Academy QA system with newer versions of OWL[15].

There are a number of situations that have come up where a similar telling of a specific question are not answered; for example, swapping the question from “which courses are required for CSUCI Computer Science Majors” to “which CSUCI Computer Science Major courses are required” does not work without breaking the sentence structure. Part of this is a result of a lack of proper labels for the resources; as it is, many of these labels have been set up to properly demonstrate the system. In the future, this part of the ontology can be extended to suit additional phrases, also allowing other languages to be interpretable as well.

The situation described above can lead to another limitation of the current system, wherein the system may be confused if a phrase references two objects with a similar label. This can easily be corrected at the ontology level by ensuring the phrases do not match, but it is more important to ensure that these changes are corrected at the system level since similar phrases with separate semantic meanings can occur in natural language all the time. At the system level, this can be resolved with a method more effective than label matching to parse natural language phrases into the ontology; in particular, Schlaefer mentions the use of “synsets,” or groups of terms that are semantically equivalent, to help establish similarities in words and, in turn, eliminating ambiguities through simple keyword matching [1].

5.2 Comparison to Other Work

In addition to the discussion of the system and ontology’s benefits and drawbacks, there is included a comparison of the system to other QA systems available for public use. Section 5.2.1 compares the Academy QA system and Academy ontology to OpenEphyra 0.1.2 [65], using a question answered correctly by the Academy QA system to demonstrate the strengths of the system’s question analysis. Section 5.2.2 compares the system to EAGLi, and demonstrate the similarities of the controlled natural language techniques. Finally, section 5.2.3 compares the academy ontology to existing university ontologies, and describe how the ontologies define academic institutions differently.
5.2.1 OpenEphyra

OpenEphyra is an open source version of the EphyraQA system discussed in section 2.4.6, and provides a statistical machine-based approach to question analysis, as well as answer extraction [1]. However, it also uses an ontology to interpret and translate words to better fit its query system in a manner similar to the approach used in the Academy QA system. Due to EphyraQA’s dependence on statistical methods, it is likely that a considerable investment in acquiring or creating a textual corpus for the university domain and concomitant refinements to the system design are required to generate acceptable QA performance. Without this, based on simple answer correctness, it does not seem that OpenEphyra is capable of answering relevant questions concerning the CSUCI catalogue, nor is it likely possible for OpenEphyra to return answers relevant to a specific student without a specific source document to cite [1]. It stands to reason that neither does this system, since the ontology used does not emulate the full CSUCI course catalogue either, but it stands to reason that, with the ontology already in place, it is possible to make the academic ontology more like the CSUCI course catalogue, which makes the system more accurate than the OpenEphyra system for answering university related questions. Likewise, it is also possible to adjust the weights of the OpenEphyra statistical word analysis to emphasize the course catalogue at CSUCI; unfortunately, this still does not guarantee an accurate answer. This is the same case with other open domain QA systems, such as True Knowledge [7].

It is clear that there are areas that the Academy ontology is more proficient at providing an answer than the variation of WordNet that OpenEphyra operates under. In figure 5-11, the question “what course is a prerequisite of COMP 151 at California State University Channel Islands” is presented. Thanks to WordNet, OpenEphyra is more capable of interpreting and analyzing a full sentence than the Academy ontology. Questions can be formed in true natural language, and it is then able to interpret and parse the entire sentence. While the system is able to derive the correct property type of the answer, it then incorrectly asserts that the target answer is a “prerequisite,” rather than a “course.” Attempting this question using “curriculum,” “class,” and stand-alone, returns the same result each time. This incorrect assertion leads to a result set of 0, leaving the system unable to answer the question.
5.2.2 EAGLi

EAGLi is a closed-domain QA system that focuses on Genomics Literature [64]. It adopts a similar approach to processing user input as the Academy QA system, as it uses a controlled natural language for user input. Much like the Academy QA system, questions are qualified by an interrogative pronoun, but unlike the Academy QA system requires that the next item in the input sequence be the target of the search. This ensures that the user is able to specify the target themselves, unlike EphyraQA, but it also limits the range of questions that can be asked of the system - for example, the vague question “who is teacher of COMP 101” lacks an expression stating that we are searching for a teacher, thus it can not be asked in EAGLi; however, it would answer the question “which teacher is teacher of COMP 101,” or some equivalent.

This distinction of only being permitted to use a single ontology for the system gives EAGLi an edge on question analysis capabilities. While limiting the corpus to a foundational ontology allows users to ask questions of ontologies greater than its own, this broad domain also makes it impossible to analyze word meanings from the lower levels of each individual ontology without a pre-built assertion for those words in the program, or a built-in reasoner, of which EAGLi has both [64].

Since the corpus for EAGLi exists in the domain of genomics [64], it does not share enough in common with an ontology focused on a university ontology to provide a direct comparison on answer correctness, nor does this system allow outside users to choose an ontology for upload. However, as the query in the figure above demonstrates, the QA system does demonstrate advantages over the Academy QA system in answer selection techniques, being able to provide a “favorable” answer among multiple selectable answers across multiple documents.
5.2.3 Existing Ontologies

There are a number of university ontologies that have been made using OWL, including the HERO ontology [54]. In fact, building a university ontology is one of the tutorials for using Protégé [43]. Focusing on the HERO ontology, there are conceptual similarities in how the ontologies are built, such as the use of courses, the separation of course definitions from course offerings, and the splitting of these based on a semester, trimester, or quarterly system, based on whatever the course is associated with. The primary difference, however, is that the HERO ontology is not built with a foundational ontology; it exists standalone. As a result, it cannot be easily extended to be linked to other ontologies using a foundational ontology, such as DOLCE. In order for this to happen, the ontology needs to be rebuilt, from the ground up, to accommodate the definitions and terms that are provided by DOLCE. This ontology is unable to function with the current system; in order for the ontology to work, there needs to be some basis for establishing what a person is, or how to distinguish events. It is possible to extract information from this the HERO ontology using SPARQL queries, but the Academy ontology provides a greater number of choices and classifications.
Chapter 6: Conclusion

In the title of this work we allude to an assessment of the feasibility of such a system in a practical, that is, non-trivial, domain. That is, we imply a judgement of both technical tractability and potential utility of such a system. In this effort, we have demonstrated a design and development method that, we believe, can be extended to support a practical implementation and eventual fielding of such a system. We have also demonstrated the potential utility of such a system, albeit in a limited way, in an academic environment. In particular, we have indicated the utility of Semantic Web technologies (OWL ontologies, SPARQL, Jena, etc.) in the development of the Academy QA System.

We discussed the necessary concepts for the creation of the Academy QA system, and how Semantic Web ontologies can be used to create a QA system. We discussed standards for additional ontologies that can be used for QA systems, and why foundational ontologies are necessary for this. To further understand this, we discussed the Semantic Web in detail, and the technologies necessary to build and navigate an ontology in the Semantic Web. We discussed domain and foundational ontologies, and their use in QA systems. We saw demonstrations of other QA systems that made use of ontologies to introduce a semantic approach to QA [1].

To demonstrate the use of a Semantic Web ontology, we have seen the creation of a new ontology, the Academy ontology, which extends the foundational ontology, DOLCE[25]. This ontology, built upon the concepts discussed and described by a university catalogue, has been used to demonstrate the uses of Semantic Web technologies, such as OWL, RDF, and SPARQL. From this demonstration, we were able to build a QA system capable of interpreting questions given in a controlled natural language, parse them into a SPARQL query, and return answers that match the context of the question.

The questions from chapter 4 were used to evaluate the semantic question analysis and query extraction techniques discussed in chapters 2 and 3. Section 4.2 describes, along with the Academy QA system’s setup, the test set to be used in chapter 5, and the reason for its use. Chapter 5 summarizes the answers given by the Academy QA system to those questions, and compares the approach to other systems and ontologies. In it, we see that the questions were interpretable, and that the answers were precise and reflected the results displayed in the Academy ontology.

At this time, there already exist a number of extensive, general purpose OWL ontologies, such as DBpedia[71], UMBEL[72] and GoodRelations[50], available on the Web via SPARQL endpoints that can easily be used to rapidly broaden and extend the range of an ontology-based QA system. As more and more focused domain ontologies are developed and published on the Web, those also may be used to extend the system. At the same time, a number of relatively new general purpose QA systems, such as Watson[5], Wolfram Alpha[9] and Apple’s Siri[26], have emerged and are enjoying notable popular success. While not all of these systems are explicitly using Semantic Web technologies, they all depend on an appreciation of semantics. From this we conclude that the type of Semantic Web QA system we have demonstrated would be deemed useful by a substantial group of future users, and that the type of system indicated by the Academy QA system is both possible and useful to construct.
6.1 Future Work

One of the primary bottlenecks for the Academy QA system, as it was implemented, was the lack of resolvable depth for many of the questions. It was possible to extract information from the ontology for statements that had explicit relations in many cases, or at least if those relations were a level of depth from each other, but it is not likely to give answers for complex questions. This is solvable with direct integration of the Academy ontology with the query extraction techniques used, but the objective was not to look at creating a system capable of only answering questions for the Academy ontology. More robust machine learning techniques, such as the statistical approach combined with the WordNet ontology that Schlaefer used in EphyraQA, may have allowed for a greater level of depth when searching for answers[1]. These concepts could also be used to enable the system to pick a “favorable” answer instead of the first available answer, where “favorable” can be, among other things, the mode of the result set[64].

Another bottleneck for this system was that the number of candidates for a particular question was limited. The empirical approach to resolving this issue is at the ontology level, by adding labels, synonyms, and subclasses that better define the various words in as many ways as possible. Looking at examples from EphyraQA, this may have also been partially resolved with a semantic parser to handle integration to the ontology, and an implementation of synsets to broaden the number of ways a question is asked [1]. For example, one area that was largely unexplored in the Academy ontology is the use of additional language specifications for labels. With an extender to filter results to specific languages, it might have been feasible to make the Academy QA system multilingual.

The ontology was able to demonstrate feasibility using the ontology as a corpus, but it has potential for providing more effective answers had the ontology linked to a Web resource, such as DBPedia, to provide real-time definitions and specifications for courses, majors, and professors. It may be worth approaching in a separate ontology to consider the possibilities of extending the university catalog to DBPedia, in a similar manner that Ms. Krishna describes adding Semantic Extensions to Wikipedia [20].

In some cases, additional ontologies can be linked to better parse user input. The Laboratory for Applied Ontology, responsible for the Wonder Web Foundational Ontologies Library and DOLCE, have been able to use DOLCE to implement an OWL version of WordNet. If this ontology were directly integrated with the Academy QA system, as DOLCE was for the Academy ontology, it is possible to create a QA system able to more closely parse true natural language. This can be extended further by enabling users to integrate with internet search engines, such as Google’s search engine[67], to find specific answers from documents on the Web. This more closely follows the approach used in EphyraQA[1].

This builds into another approach that may have increased the number of options available for the Academy QA system: using the ontology - or collection of related ontologies - as a system knowledge base to support a wide range of intelligent functionality. For example, reviewing the Academy ontology with a reasoner would result in a larger, logically robust ontology. This is one of the areas of the Semantic Web stack [56] that was left undiscovered in chapter 2: the use of a rule language may have been helped for assessing some of the layers not
accomplishable through empirical entry via OWL DL. For example, in early stages of the ontology where certain properties were expressed as the combination of other properties, such as determining a candidate for graduation for fulfilling certain requirements. In addition to allowing a direct connection to other ontologies, such as those implemented through other universities, or DBPedia, a reasoner would result in a dramatic expansion of the Academy ontology's existing definitions. This requires a pragmatic approach; while it is feasible for a rule engine to reason with an ontology written in OWL Lite, a rule engine working on an ontology in OWL DL has a higher worst-complexity, and a rule engine can take infeasible amount of time to run through an ontology in OWL Full [69]. It is possible to build a set of rules on the ontology using one of the W3C standard rule languages, such as the Rule Interchange Format(RIF), or the Semantic Web Rule Language(SWRL), making the process of reasoning much simpler; however, these languages are not always sufficient for OWL DL [60].

The ontology and system may have been improved by making full use of more up to date software. The version of Protégé used for this thesis is 3.4, but as of this writing, there is a more up to date version of Protégé available [40]. This latest version of the Protégé is not compatible with the Academy ontology, which was created using an earlier version of OWL(1.1); as of Protégé 4.0, all projects have been integrated to use OWL 2.0. It was possible to update the ontology to use OWL 2.0, but it was not necessary for the scope of this thesis. In the future, such an update would give users seeking to improve the ontology a broad range of options, as well as more effective reasoning utilities should they choose to use them.

Other foundational ontologies may have better suited the needs of this system than DOLCE. One example of this is Dublin Core [61]; unfortunately, the simple Dublin Core metadata element set does not meet the needs for defining an organization, and is suited more for the organization of Web resources. There are other ontologies, such as the Organization ontology [62], that may have also been used in the context of a university; unfortunately, this ontology is not suited for a QA system attempting to resolve multiple domains beyond those defining organizations. It is possible to use these ontologies in conjunction with DOLCE, and it would be worth exploring some of the more established ontologies, such as the Semantic Web version of Cyc[63].

The Academy ontology can be extended by implementing the categories “SpaceRegion” and “Place” to define universities as physical and political geographic entities. Once implemented, this may allow us to create additional sub-areas necessary to answer questions related to specific locations, such as “where is COMP 101 Section 1 being taught,” or “where is CSUCI” giving filtered location information for any property related to CSUCI.

There are many areas that have a class defined in the Academy ontology, but have no associated instances. Advisors, Administrators, Researchers, and Tutors are all available, with no entries. There are academic actions, such as enrolling in a course, completing a course, even instances for describing when an exam is within a course, that could have all been specified as members of specific courses to give us a means of asking when any of those occur, but were left empty. By adding instances to these classes, it would be possible to convert the Academy ontology into a fully functional university catalogue, if necessary.
References


