CS 700B Master Project Report

***Automatically Extracting Useful Information from DBpedia into an Ontology to Improve Search Query Suggestions***

Project Report Submitted to   
Department of Computer Science   
College of Computing Sciences  
New Jersey Institute of Technology

In Partial fulfillment of the requirements of the Degree of Master of Science in Computer Science

Submitted By:

Christopher Ochs  
NJIT ID: 215-61-729  
cro3@njit.edu

Project Advisor: Dr. James Geller

May 4, 2011

**Approval by Project Advisor**

Project Advisor: Dr. James Geller

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Approval by MS in CS Committee**

Project Number: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submission Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Project Evaluation: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
(By Graduate Advisor / Committee)

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

I hereby affirm that this report is my own personal work and that all material other than my own is properly referenced.

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Abstract**

In the early 2000s Google introduced the ability to see a small subset of common search queries as a user typed a query of his or her own. This feature, named Google Autocomplete, helped evolve how people use search engines. The search experience was significantly sped up; a user didn’t have to enter an entire search term to find what the query he or she was planning on entering. Additionally, if a user is unsure of what he or she is looking for, Google Autocomplete provides more specific suggestions for an entered query [1]. However there are issues with Google Autocomplete; the suggestions are generated based on other users’ queries, which often results in useless or ambiguous suggestions [1].

To improve query suggestions Dr. Geller’s group has been working on a project named Ontology Supported Web Search (OSWS) to integrate ontologies, a type of structured knowledge base, into search engines. This project was focused on building an ontology of famous people for use in this system. To build this ontology we first extracted names of famous people from Google Autocomplete. Next, I developed a system to extract information from the DBpedia Ontology, a large knowledge base extracted from Wikipedia. After analyzing the data that was collected, I built an ontology out of the 3200 “most famous” people. This ontology was plugged into the OSWS front end, along with a system to dynamically expand the ontology with new knowledge based on user search queries.

Table of Contents

[1 Background 5](#_Toc292314126)

[1.1 Search Engines 5](#_Toc292314127)

[1.2 Automatic Query Suggestions 6](#_Toc292314128)

[1.3 Ontologies 7](#_Toc292314129)

[1.3.1 Concepts 8](#_Toc292314130)

[1.3.2 Relationships 8](#_Toc292314131)

[1.4 DBpedia 10](#_Toc292314132)

[1.4.1 The DBpedia Ontology 11](#_Toc292314133)

[1.4.2 Yet Another Great Ontology (YAGO) 13](#_Toc292314134)

[2 An Overview of the OSWS Project 14](#_Toc292314135)

[3 My Work (Improvements to the OSWS Ontology) 15](#_Toc292314136)

[3.1 Identifying Famous People 16](#_Toc292314137)

[3.2 Classifying Famous People 17](#_Toc292314138)

[3.2.1 DBpedia Ontology 17](#_Toc292314139)

[3.2.2 Mapping from YAGO to DBpedia Ontology 18](#_Toc292314140)

[3.2.3 Mapping from Disambiguation Tags to DBpedia Ontology 20](#_Toc292314141)

[3.2.4 Mapping from Wikipedia Abstracts to DBpedia Ontology 21](#_Toc292314142)

[3.2.5 Choosing the Best Classification 22](#_Toc292314143)

[3.3 Structuring the Relationships and the Attributes 22](#_Toc292314144)

[3.3.1 Identifying Possible Relationships and Attributes 22](#_Toc292314145)

[3.3.2 Organizing the Relationships and Attributes 23](#_Toc292314146)

[3.4 Building the Ontology of Famous People 25](#_Toc292314147)

[3.5 Dynamically Expanding the Ontology of Famous People 25](#_Toc292314148)

[3.6 Generating Search Suggestions 28](#_Toc292314149)

[4 Conclusions & Future Enhancements 29](#_Toc292314150)

[Acknowledgements 31](#_Toc292314151)

[Appendix A: Developer’s Manual 33](#_Toc292314152)

[A.1 Setting Up the Development Environment 33](#_Toc292314153)

[A.2 Retrieving the OSWS Source Code 34](#_Toc292314154)

[A.3 Deploying OSWS to osws.njit.edu 37](#_Toc292314155)

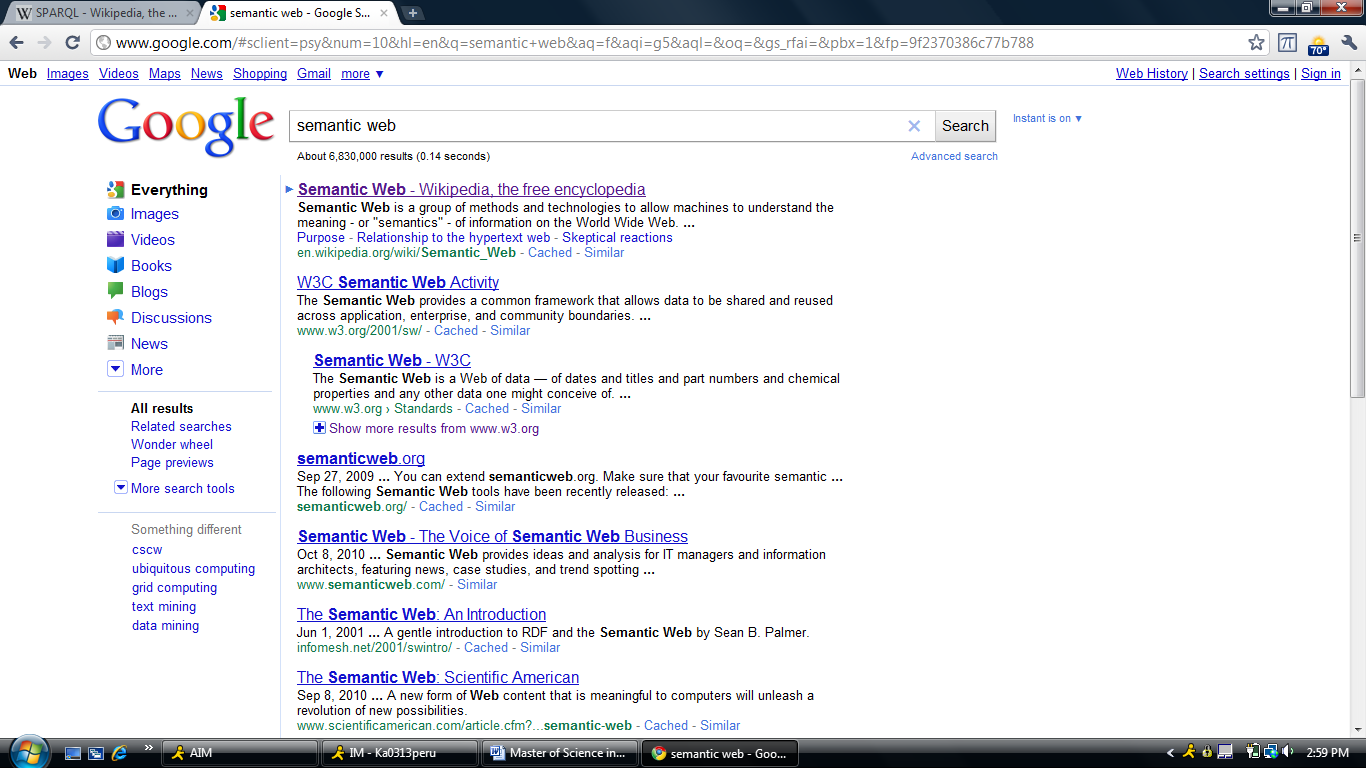
[Appendix B: OSWS Project Source Code 39](#_Toc292314156)

# Background

As a project integrating many facets of the latest Web technology, as well as the latest research into ontologies, there are many subjects that must be understood in order to grasp the entire Ontology Supported Web Search project, as well as the Ontology building system I implemented. Before I introduce the specific implementation details of my project, I will cover all of the major areas that my project touched on in order to provide a better understanding of what was accomplished with my project.

## Search Engines

As the Web expanded in the early to mid-1990s there were many efforts to aid users search for data. Search Engines automatically visit web pages, saving and indexing the content stored on the pages and following links within each page. Search engines provide users a simple interface to enter a search term and will return pages with content related to that term. Figure 1, below, shows a Google page containing results for the term “Sematic Web.” Throughout the early days of internet companies like Yahoo, Google, Ask Jeeves, Altavista, and Microsoft all fought for their share of the search market [10]. As use of the Web exploded in the late 1990s and early 2000s, billions of web pages had been indexed by all of the major search engines.



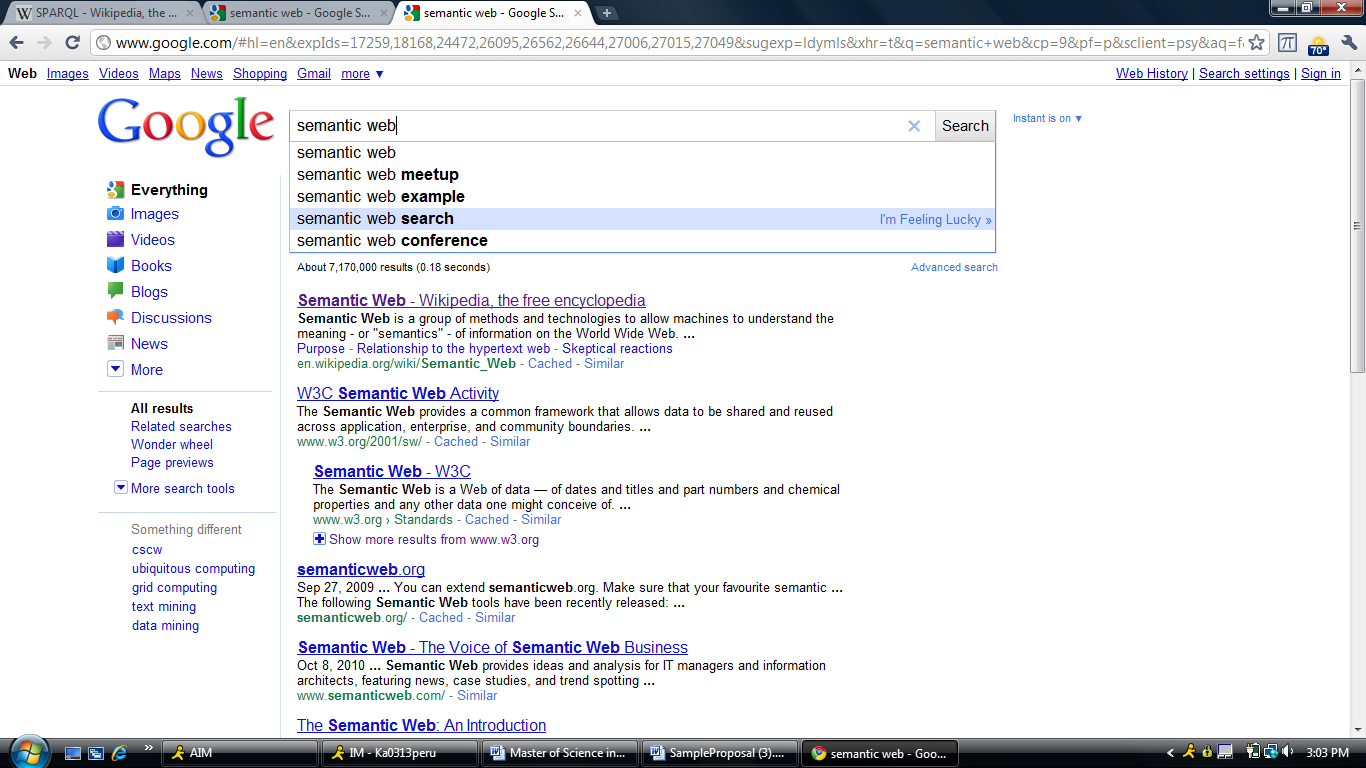
**Figure 1:** A standard Google search results page. This particular results page is for the term “semantic web.” [11]

In the early 2000s Google had established itself as the leading search provided in the world. Google serves nearly 67% of searches on the Web [12]. Their indexing method, named PageRank, aided search indexing by giving a weight to a webpage based on how many other pages link to it [13]. Google has succeeded at providing the best search experience. Their interface is clutter free, giving users only a search box and a button to initiate the search. Results are displayed in an easy to read list with the most relevant results returned first. This is in contrast to companies like Yahoo and AOL, which view themselves as “Web Portals,” offering more information and features outside of search.

Google has spent much of their effort attempting to create the most relevant first page of results. If users can find what they are looking for on the first page then there is little need for them to browse through dozens of search result pages. Google has also introduced new features to further improve the speed of their search engine. In 2004 they introduced Google Autocomplete, described in the next section, and most recently introduced Google Instant, which automatically updates the results page [1].

## Automatic Query Suggestions

In 2004 Google introduced a feature named Google Autocomplete, which presents users with a popup containing a small subset of common search queries. The suggested queries that are returned are based on the search term a user has entered by typing [1]. Google Autocomplete updates this display in real time as a user continues to enter his or her search term. In Figure 2, below, you can see a sample Google Autocomplete popup for the term “semantic web.” Autocomplete has been significant for a number of reasons. Primarily, it sped up the search process by giving users the ability to choose from likely search queries before they had completed their own. Secondly, if a user is not sure of what he or she is searching for, Google Autocomplete provides helpful suggestions based on what the user has currently entered.



**Figure 2:** A Google Autocomplete window for the search term “Semantic Web.” [11]

Google Autocomplete works by providing suggestions based off of other user’s queries [1]. All suggested search queries were at some point entered by another user [1]. In this way, Google Autocomplete will stay up to date with relevant search queries. By providing users with a simple to use and incredible useful feature, Google made searching faster and more accurate. Other search engines soon followed suite, and search suggestions have become a standard feature in almost all search engines [1,3,4]. Additionally, products that include search functionality, such as YouTube, Wikipedia, and Gmail, provide a Google Autocomplete-like experience for search terms.

## Ontologies

In philosophy, ontology is the metaphysical study of the nature of being and existence [9]. In computer science an ontology is much more concrete in nature; it is a rigorous and exhaustive organization of some knowledge domain that is usually hierarchical and contains all the relevant entities and their relations [9]. In layman’s terms, an ontology is a highly organized knowledge base that stores relevant data on some topic. Ontologies are used extensively in artificial intelligence, medical informatics, and semantic web research. Ontologies are made up of two primary pieces; concepts and relationships. Using these building blocks, an ontology can structurally represent many kinds of knowledge.

### Concepts

Ontologies are built from concepts, which are (informally) pieces of information that can be related to other pieces of information through various kinds of relationships [9]. Formally, a concept is an abstract or general idea inferred or derived from specific instances [9]. An oft-used comparison is to Object Oriented Programming; in OOP there are objects being instances of classes. In an ontology concepts are instances of some class. For example, Kurt Cobain is an instance of Musician. Concepts are usually defined within the domain a given ontology covers.

Concepts have some unique identifier that distinguishes them from other concepts, which in many cases may be similar. For example, there are either “famous” people with the name Michael Jordan. There is, of course, the Basketball Player, but there is also the politician and soccer player. A concept may also store characteristics or qualities about itself; these are called attributes. Attributes are values that help to further define a concept.

### Relationships

In an ontology, concepts are linked together using what are called relationships. A relationships represents a link between two concepts; it explicitly shows that two pieces of knowledge are related to each other. Relationships are often hierarchical in nature; they show that one piece of knowledge is more specific than another, much like a class inheritance tree in OOP. The set of hierarchical relationships generally forms either a tree structure or a directed acyclic graph of concepts that go from very specific to non-specific. For example, a Singer is a Musician, a Musician is an Artist, and an Artist is an Entertainer, and an Entertainer is a Person. This hierarchical structure can be used to classify concepts based on where they exist within the hierarchy. Figure 3, below, shows a small subset of the DBpedia Ontology’s class structure.

**Figure 3:** A subset of the DBpedia Ontology Person Hierarchy. Classes are linked by hierarchical relationships. Concepts are instances of one of these classes [14].

A second type of relationship is the semantic relationship, or associative relationship. These non-hierarchical relationships link one concept to another concept, showing that two concepts are related in some way. This is in contrast to attributes, which are simply named data values stored within a concept. For example, a musician may play some kind of instrument, or play a specific kind of music. A visual example is provided in Figure 4 below. In an ontology of musicians, types of instruments will be stored as concepts. An instance of musician will have a “plays instrument” relationship to some instrument concept. These semantic relationships help further define a concept based on specific information on that concept.

**Michael Jackson**

|  |  |
| --- | --- |
| Description: | Singer |
| Birthdate: | 1958-08-29 |
| Death: | 2009-06-25 |

Musical Artist

Band

**The Jackson 5**

Company

**Motown Records**

record label

associated band

**Figure 4:** An example from the DBpedia Ontology. Michael Jackson is an instance of Musical Artist with attributes Description, Birthdate, and Death. Michael Jackson has semantic relationships “associated band” and “record label” to concepts “The Jackson 5” and “Motown Records,” respectively. [15]

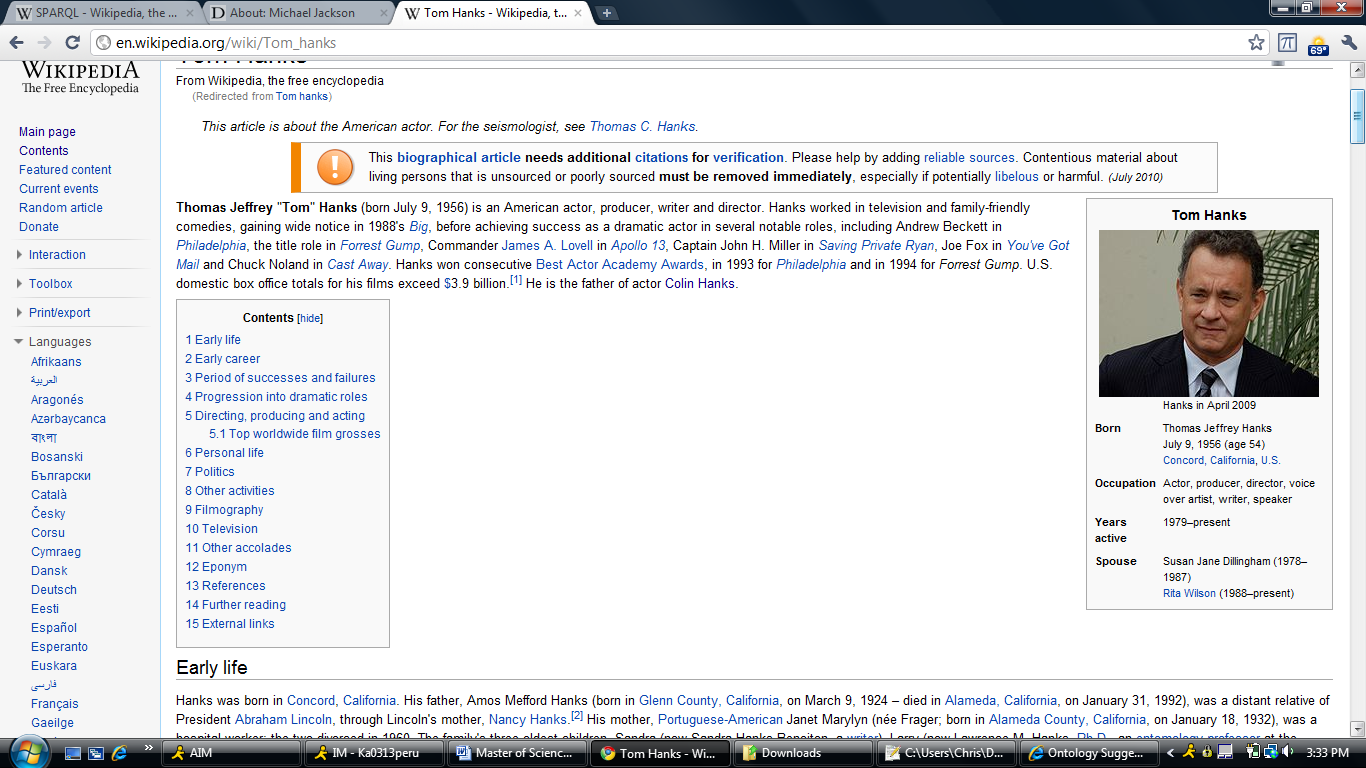
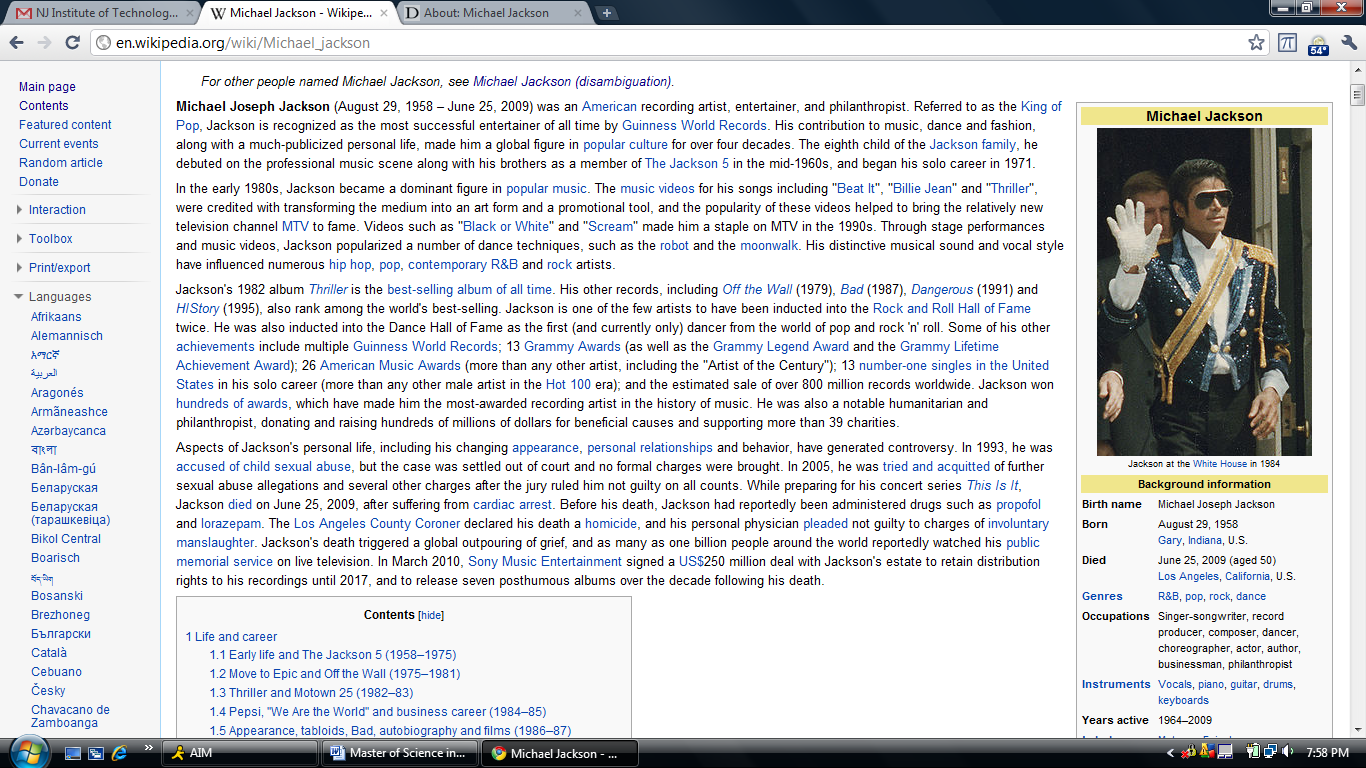
## DBpedia

DBpedia is a massive collection of data extracted from Wikipedia and other sources [6]. The DBpedia project was started in the late 2000s by Free University of Berlin and the University of Leipzig [6]. DBpedia currently stores information on more than 3.4 million things covering many domains, including people, places, and organizations [6]. Data is stored in a format named Resource Description Framework (RDF) [6,7]. RDF is a triple storing two entities and some relationship between those entities. In DBpedia the entities stored in this triple are either concepts or attributes. For example, the triple for the associated band attribute in Figure 4, above, would have the entities dbpedia:Michael\_Jackson and dbpedia:The\_Jackson\_5, and the relationship linking them would be dbpedia-owl:associatedBand [15].

DBpedia contains over a billion of these triples in its dataset, a majority of which are non-English in nature, but roughly 257 million are from the English Wikipedia pages [7,8]. DBpedia is made up of various sources of information, including their own ontology (the DBpedia Ontology), YAGO (Yet Another Great Ontology), and Friend of a Friend (FOAF). DBpedia allows users to extract information from their dataset via a web-based SPARQL (SPARQL Protocol and RDF Query Language) interface or by downloading the entire dataset [7].

### The DBpedia Ontology

In order to understand how the DBpedia Ontology was constructed, one needs to first understand how Wikipedia represents data on its pages. On many Wikipedia pages there is a box on the right side of the page, generally with pictures and with several relationship-attribute pairs, these are called Infoboxes. These Infoboxes are standardized in nature, such that a musician will have a Musician Infobox and a basketball player will have a Basketball Player Infobox. Each type of Infobox stores different kinds of information. A Musician Infobox will have information such as genres, record labels, and when they were born, while a Basketball Player Info box will have information such as what team the person plays on. In the DBpedia Ontology these information box types are treated like classes [7]. The DBpedia project organized these Infoboxes into a shallow hierarchical structure [7]. The class structure is only a few levels deep, but there are many instances of each class. Every page that contains an Infobox will be an instance of the class of the Infobox type on the page [7]. For example, Michael Jackson is a musician because he has a Musical Artist Infobox. Tom Hanks is an Actor because he has an Actor Infobox.



**Figure 5a:** The InfoBoxes for Michael Jackson and Tom Hanks [16].

The DBpedia project extracts all of this Infobox data from Wikipedia pages and stores each page, plus the attributes and relationships within the page, in its dataset [7,8,9]. Pages are treated as concepts; since they have a type of infobox, they are an instance of the class associated with their Infobox’s type. Hierarchical data is derived from the class structure created by DBpedia. Semantic Relationships are obtained when an attribute in an Infobox is another Wikipedia page [7,8,9]. For example, in Figure 5 above, Tom Hanks’s has a Spouse attribute with a value Rita Wilson [16]. Rita Wilson has a Wikipedia page, so within the DBpedia Ontology is an instance of a class, in this case Actor [17]. There is a dbpedia-owl:spouse relationship between Tom Hanks and Rita Wilson [17]. Attribute values are obtained when the value of an attribute in an Infobox is not a Wikipedia page, such as birth date. Since the ontology is built off of Wikipedia, the DBpedia Ontology provides a large dataset that is reasonably well structured. In figure 5b you can see the DBpedia entry created from the Tom Hanks Wikipedia page.



**Figure 5b:** The resulting DBpedia entry for Tom Hanks.

### Yet Another Great Ontology (YAGO)

In addition to the DBpedia Ontology, DBpedia also includes the YAGO project’s ontology. YAGO, short for Yet Another Great Ontology, is another ontology project using Wikipedia as its starting point. YAGO is being developed by the Max Planck Institute for Computer Science. The project takes the category names of Wikipedia pages and matches them to instances in WordNet [18]. WordNet is a massive ontology of English language words managed by the Cognitive Science Laboratory at Princeton University, and has been supported by the National Science Foundation [18].

Nearly every page on Wikipedia is given a set of categories. For example, the Tom Hanks page belongs to the categories 1956 Births, Living people, American Film Actors, and American Film Directors [17]. These Wikipedia categories are semi-hierarchical in nature, but YAGO does not use this hierarchy. The YAGO project analyzes the categories of Wikipedia pages and maps them to WordNet classes, essentially making the Wikipedia category a subclass of a WordNet class [19]. Pages are then treated as instances of the classes created from their categories [19]. From the example above, Tom Hanks is an instance of AmericanFilmActors and AmericanTelevisionActors [17]. The categories American Film Actor and American Television Actor were created as subclasses of the WordNet class Actor (Id #109765278) [17].

In DBpedia, the YAGO categories are listed for any DBpedia resource (Wikipedia page) that has been mapped in YAGO. This forms a second ontology within DBpedia that contains information on pages that may not have an Infobox and therefore were not covered by the DBpedia Ontology. The YAGO Ontology is less organized than the DBpedia Ontology. YAGO has a number of issues; most importantly concepts may belong to a relatively large number of classes. This is in sharp contrast to the DBpedia Ontology, where every concept is an instance of exactly one class.

# An Overview of the OSWS Project

My project was an extension of previous work completed by Dr. Geller and his students [5]. The Ontology-Supported Web Search (OSWS) project began in 2009. The goal of the OSWS is to seamlessly integrate ontologies into web search for the purpose of providing improved results [5]. The first version of the project integrated an ontology of five thousand American musicians, Basketball players, and Football players into a Google Autocomplete-like interface (see Figure 6 below) [2,5]. The interface provides users with helpful suggestions for entered search terms, similar to Autocomplete. However, instead of being based off other user’s previous queries, the suggestions were based off knowledge stored in an ontology [5].



**Figure 6:** The Ontology Support Web Search interface [2].

There was also significant research done in advance of this project during the summer of 2010. Research was done on extracting search suggestions directly from Google and matching the suggestions to names. The discovery of DBpedia led to a shift in how the data used to build the OSWS Ontology was obtained. Through the latter half of the summer I performed a significant amount of research into what types of information can be extracted from DBpedia, as well as how we can integrate the data into the Ontology-Supported Web Search system.

# My Work (Improvements to the OSWS Ontology)

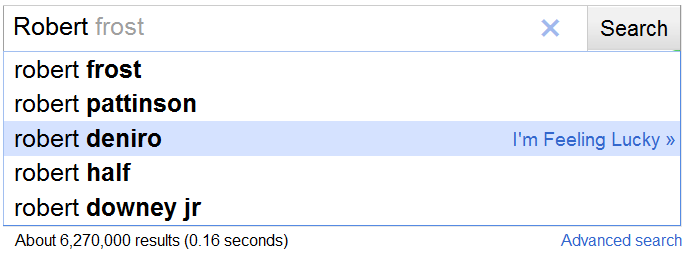
The OSWS system was designed to provide disambiguated search suggestions; therefore our ontology must contain data useful for such a purpose. We began this research by determining how to best expand the domain of our ontology. After exploring a number of options, such as mining Wikipedia data ourselves, we instead chose to use the already well-structured data of DBpedia. Our initial analysis included determining what types of information were available, the structure of the DBpedia ontology, and how to best integrate their system into our OSWS interface. We found that we couldn’t use the DBpedia ontology “out of the box,” instead we would extract subsets of information from their ontology for use within our own.

While planning the ontology, we identified three key focus areas of research. The first area was domain coverage; previously our ontology focused only on American musicians and sports stars (mostly basketball players). This small domain of people is not enough for a real-world application, so we set out to expand our ontology and cover as many famous people as possible. The second key area was assuring that we store up-to-date, relevant information in our ontology. We made the decision, as detailed in Section IV, of designing a dynamically expanding system that will identify what users of the OSWS system are searching for and expanding our ontology to include those people automatically.

The third focus area was determining DBpedia relationships between concepts that are best for use in our ontology. Since famous people are the focus of our ontology, we mainly incorporate relationships from people concepts to other target concepts, such as movies starred in, songs produced, sports teams played on, etc. This gives us a fine granularity for describing people but also provides useful classifications for other concepts they are related to. To develop this ontology we “mined the Google search engine” to identify a small subset of famous people, supported by using government census data available to the public.

## Identifying Famous People

We began the process of expanding our ontology by mining “famous people” from Google search suggestions. Using publicly available U.S. census data, we queried Google’s public SOAP API for search suggestions with the top 1000 male and female first names from the 2000 census. For example, we passed the name Robert to Google and then extracted the last names Frost, Pattinson, DeNiro, Half and Downey Jr. as people that Google knows are famous right now (Figure 7). We collected the returned results and looked for the ones of the form “*n1 n2 n3,*” where *n1* is the person’s first name, *n2* is the optional middle initial, and *n3* is the last name. We checked the values of the last name against the 5000 most common last names as contained in the census data. Using this method we mined 5286 famous people from Google.



**Figure 7.** Google suggestions for search term “Robert.”

We classified these people as members of the “A-List,” as they are the search suggestions returned by entering only a first name. We repeated this process with “*n1 l1*” style queries, where *n1* is a first name from the census data and *l1* is a letter in the alphabet. This type of query further refines the suggestions by potentially including a specific middle initial or start of a last name. We named the results from this set of queries the B-List, and it is comprised of 132,896 candidates. Finally, we queried with the format “*n1 l1l2,*” where *l1* and *l2* are letters in the alphabet, and mined the returned names, storing them as the C-List, which was composed of nearly a million potentially famous people.

As many properly formatted suggestions were clearly not referring to people (“Sterling Silver,” “Sam Adams,” “John J College”), we next identified what names correlated to an actual person. We took the 5286 names in the A-List and queried DBpedia to determine if any of the names could refer to one or several people. We analyzed the type and Wikipedia subject data stored within DBpedia to make this determination. For example, if a given DBpedia page contained the type “ontology:person” or “yago:person,” we considered this a valid person. Similarly, if the name belonged to the Wikipedia category “living people” or belonged to Wikipedia categories that ended in “births” or “deaths,” i.e. “1986\_births,” we considered the name to be a valid person. Using this method we identified 3241 people among the potential names in the A-List.

## Classifying Famous People

### DBpedia Ontology

Once we extracted the names and correlated them to real people, we inserted them into the OSWS ontology. We based our ontology off of the person hierarchy within the DBpedia ontology. The DBpedia ontology is built from data stored on Wikipedia pages [xx]. Specifically, they use the “Info Boxes” which are included on many Wikipedia pages. Info Boxes are tables of information that are located on the right side of certain Wikipedia pages. These boxes have specific types associated with them, such as Actor Info Box or MusicalArtist Info Box. DBpedia’s ontology is built using these Info Box types as class names, and a page with a specific type of Info Box is assigned that type. For example, Tom Hanks has an Actor Info Box, and he is an instance of the class “Actor” in the DBpedia ontology. We felt that this structure (shown in Figure 8) was adequate for providing the appropriate granularity for use within our ontology.



**Figure 8.** Subset of person hierarchy in the DBpedia ontology in indented format.

The ontology was built by extracting the complete person hierarchy from the DBpedia ontology, and manually adding several other non-person hierarchies. The non-person classes are selected based on their usability as relationship targets for the person classes.

One major issue we faced when using the DBpedia ontology, however, was that within the A-List hundreds of people were found to exist in Wikipedia who did not have DBpedia ontology types associated. We developed a number of ways to augment the DBpedia ontology with our own additions, expanding the domain coverage. For each correlated name in the A-List we had to map it to a class within the hierarchy. While the DBpedia ontology contains over 360,000 categorized people, and continues to expand, there was still a significant number missing, amounting to approximately 520 (17%). Occasionally the DBpedia class was very general and uninformative, such as “Person.” In such cases we attempted to provide more specific classes for these concepts.

### Mapping from YAGO to DBpedia Ontology

YAGO (Yet Another Great Ontology) [18] is an ontology built from Wikipedia page categories being mapped to Wordnet [18] classes. Because of the way the Yago ontology was built, instances often belong to many classes. DBpedia provides mappings to YAGO within their ontology. We first collected the DBpedia ontology types and YAGO rdf:types for all pages correlated to a name in the A-List. We found that the YAGO types were often far too specific for mapping (“AmericanDanceMusicians”), so we went to their broader type (“Musician”), which provided a more useful class name. Using programmed string matching we were able to match roughly 40 YAGO classes to DBpedia classes. Additionally, we used the approximately 450 pages which existed in both DBpedia and YAGO to perform statistical analysis and map a YAGO class to a DBpedia class. For example, if a certain percentage of pages with DBpedia type MusicalArtist had the YAGO types Singer or Soprano, we’d map both to the DBpedia class MusicalArtist. Finally, we sorted through a small number of the YAGO classes by hand and mapped them to DBpedia classes. In total we mapped 85 of the most commonly found YAGO classes to DBpedia classes.

For each page with a set of YAGO classes, we determined the broader YAGO classes and used our mappings to find the DBpedia class for each YAGO class on the page. We then counted the occurrence of each DBpedia class and selected the one with the maximum occurrence as the correct mapping class type. For example, the YAGO ontology classes for Kurt Cobain are “American Diarists,” “Grunge Musicians,” “Musicians From Washington(U.S.State),” “American Musicians Of Irish Descent,” and “People From Olympia, Washington,” among others. We go to the broader types, which are “Diarist,” “Musician,” “Musician,” “Musician,” and “Person” respectively. Or mapping system maps Diarist to Writer, Musician to MusicalArtist, and Person to Person (in DBpedia). In the above example, we find that Kurt Cobain has one mapping to Writer, three to MusicalArtist, and one to Person. We choose the most common mapping, in this case MusicalArtist, and assign the instance, in this case Kurt Cobain, to that type.

Using instances in the A-List that have both YAGO and DBpedia types, this method gave the same classification for 268 out of 401 (67%) instances. Additionally, this mapping system gave a more specific class to 47 people (12%) who were classified as “Person” in the DBpedia ontology. Many other mappings were simply less specific then the DBpedia type given (such as being mapped to Athlete instead of Wrestler), but not unusable.

### Mapping from Disambiguation Tags to DBpedia Ontology

We found that a large number of pages not categorized by the DBpedia ontology were homonyms of more famous people with the same name. As homonyms are a common occurrence in their data, Wikipedia handles homonyms by adding a “disambiguation tag” to the end of a page name. For example, there are a number of Michael Jordan’s on Wikipedia. One Michael Jordan, the famous basketball player, has a page name of “Michael\_Jordan.” Other Michael Jordans have page names such as “Michael\_Jordan\_(footballer)” and “Michael\_Jordan (Irish\_politician);” a soccer player and a politician respectively.

We took advantage of the information between the parentheses and constructed a set of mappings from disambiguation tags to DBpedia classes. We were able to map 50 of the most commonly occurring tags. Many disambiguation tags are in the form of [type]\_born\_[year], or [nationality]\_[type], for example “(footballer\_born\_1984)” or “(american\_singer),” respectively. By matching the type to a mapped tag, we are able to correctly categorize many pages based on their disambiguation tags. Out of 233 people in the A-List who had no DBpedia class and a disambiguation tag, we were able to map 118 people (51%) into the Ontology using the 50 most common tags. By adding more mappings we’re able to increase this to close to 100%.

### Mapping from Wikipedia Abstracts to DBpedia Ontology

When we are not able to identify a person’s type using any of the previously discussed methods, we resort to a less reliable method. A Wikipedia abstract is the paragraph that appears at the top of a Wikipedia page. Many of these abstracts start in the form of “someone is/was something,” which introduces the occupation of a person. For example, Basketball player Michael Jordan has his Wikipedia page introduction starting with “Michael Jeffrey Jordan is a former American professional basketball player.”

By analyzing the abstract of a page and extracting the occupation information, we check the occupation against our list of class names and if there is a matching between the occupation and a class we assign the person to that type. In the example above, we find that Michael Jeffrey Jordan is a basketball player by seeing “basketball player” in the abstract and matching it to the BasketballPlayer type in our ontology. We presently do not take into account multiple occupations. Many abstracts have a list of occupations separated by commas or “and.” For example, Martin Scorsese’s occupations are “American film director, screenwriter, producer, actor, and film historian.” For this research we only considered the first class first we were able to match.

Using this method we were able to add 248 new instances out of 473 previously unidentified people into our ontology. In a random sample of 50 of these instances, 44 (88%) were matched correctly. All errors in this sample set were due to the very simplistic way we check occupations against classes. For example, “martial artist” was matched to the class Artist, and “personality” was matched to the class “Person.” Using a more advanced method of string matching or natural language processing would greatly increase not only accuracy, but also increase coverage.

### Choosing the Best Classification

For each name in the A-List, we retrieved the DBpedia Ontology class, YAGO classes, disambiguation tag, and abstract, if each existed. When mapping we give equal weight to the DBpedia class, our mappings from YAGO, and the disambiguation tag. In the event there are multiple possible mappings, we choose the class that is lowest in the hierarchy and assign the instance to that class. In the event there is no DBpedia class, YAGO classes, and disambiguation tag for a person, we resort to using the abstract to classify the person.

## Structuring the Relationships and the Attributes

DBpedia contains a rich set of relationships within their ontology, however for search suggestions many of them are not helpful for the end users. Additionally, information is often not organized well enough for use in search suggestions. We decided to use the existing relationship data within DBpedia for our ontology, but we had to restructure it for use in search suggestions.

### Identifying Possible Relationships and Attributes

After assigning over 3241 people within the A-List to a class, we proceeded to query DBpedia for the types of relationships each instance possessed. We calculated how often each relationship or attributes appeared, shown in (1), where Nr stands for the total number of people in a given class with a particular relationship, while N is the total number of people belonging to the class. Relationships and attributes that appear most often were more likely to be used in our ontology.

 (1)

When building our ontology we chose *p* to be 0.5 (50%). After we filtered out all of the relationships where *p* was less than 0.5, we excluded common relationships that we felt were not useful for search results, such as “subject” (Wikipedia category), “label,” and others. Finally, we did a hand review of the remaining 213 relationships and attributes to make sure they made sense for their assigned class.

### Organizing the Relationships and Attributes

Once we identified the set of relationships and attributes we’d be including in the OSWS ontology, we organized them based on our needs. One problem with DBpedia’s data is there are often redundant relationships, for example Actors have the relationship “dbprop:starring” and “dbpedia-owl:starring,” which mean practically the same thing. When considering relationships such as these for use in our Ontology, we treated them as meaning the same thing when it was appropriate to do so.

A second issue is many relationships in DBpedia are lacking in granularity or have multiple meanings. For example, the relationship “writer” can mean writer of a book, movie, song, or television show. The context of a relationship often depends on the class of the source. We wanted our ontology to distinguish between these semi-ambiguous relationships, so in many cases we split a DBpedia relationship into two or more relationships in our ontology. One example is the “starring” relationship, which links a person to a movie or television show. We replaced it with the “stars in television show” and “stars in film” relationships and use one or the other depending on the type of the target in DBpedia. For each relationship we introduce into our ontology we identify the type of the target in DBpedia by looking at the type in the DBpedia ontology or by looking at the Wikipedia subjects. For example, if there is a relationship in DBpedia “Forest Gump starring Tom Hanks,” while adding this relationship to our ontology we see that Forest Gump has a DBpedia class “Film,” additionally it has the subject “1994\_films.” Knowing that Forest Gump is a film, we introduce the relationship “Tom Hanks stars in film Forest Gump” into our Ontology.

We reversed the order of many relationships that exist in DBpedia. It’s common in DBpedia to see relationships for actors and musicians in the form [Movie] starring [Actor] or [Song] performed by [Musician]. Since our ontology is person-focused we reversed these relationships to make the person the focus of the relationship. In our ontology the relationships would be [Actor] stars in [Movie] and [Musician] performs song [Song]. We also made the design choice of promoting certain DBpedia properties to full relationships where it made sense, such as instrument played by a musician, or comedic genres for comedians. By promoting these attributes to relationships we are able to more explicitly show a linking of concepts, such as two musicians playing the same instrument(s). There are cases where attributes are a comma separated list of terms in DBpedia, in such cases we break the attribute apart at the commas and make each token its own instance.

For non-interpersonal relationships (where the source is a person but the target is not) we organized the targets in a shallow hierarchy based off of DBpedia’s ontology. We removed all unnecessary classes (those in range of no relationships), slightly restructured certain classes, and augmented the ontology with a number of new classes. For interpersonal relationships we had to decide how to handle targets that did not exist in our ontology. If we recursively loaded all information for each target, we would rapidly cover the entirety of DBpedia, which was not our goal. This would also be counter to the idea of our ontology only storing famous people. If the target was not yet searched for, we do not yet consider them famous.

The solution to this problem was the introduction of “stub” instances for such person concepts. Like other target concepts in our ontology, stubs consist of only a name and their assigned type. This prevents the recursion problem while still covering them within our ontology. If a stub is eventually searched for by a user it will be promoted to a full instance and relationship data will be loaded into our ontology. Stubs, only containing the minimum of information about a person, are not returned as search suggestions.

## Building the Ontology of Famous People

Once our ontology’s structure was complete, we utilized the Protégé Java API and built the ontology programmatically. Initially we built the class hierarchy and required relationships. Then, for each name identified in the A-List, we identified its type and added the instance. For each instance added, we queried DBpedia for the necessary relationship and attribute information. For each valid relationship, we added the target as an instance if it did not yet exist. When finally completed, we had an ontology consisting of 3241 people and over 60,000 relationships from those people.

Finally, we integrated the OSWS ontology into our OSWS front end. The OSWS system includes a number of user friendly features, such as displaying longer versions of relationship and class names, class-specific relationships being displayed first, and the ability to filter by class name. To continue to support these features we had to slightly modify our middleware to work with the new ontology’s structure.

## Dynamically Expanding the Ontology of Famous People

Ontologies generally fall into one of two categories; automatically generated and coving a large domain or hand crafted and covering a relatively small domain. Ontologies like DBpedia and YAGO fall into the former. Information stored in these ontologies is generally less organized and often not a good when compared to smaller, hand crafted ontologies.

While the A-List ontology covered 3200 of the most famous people according to Google, we recognize that user’s search interests change on a regular basis. Who is popular and who is not can change overnight. Keeping our ontology of famous people up to date would require significant time and effort if done by hand. Instead, we chose to combine the best attributes of automatically generated ontologies and handcrafted ontologies and developed a system that automatically keeps our Ontology up to date.

Using the A-List as a “training set,” we developed a system to dynamically expand the OSWS ontology based off user searches within the OSWS interface. By plugging the various systems developed for building the A-List ontology into the OSWS front end, we devised a way of expanding our domain with minimal input from either us or the end users.

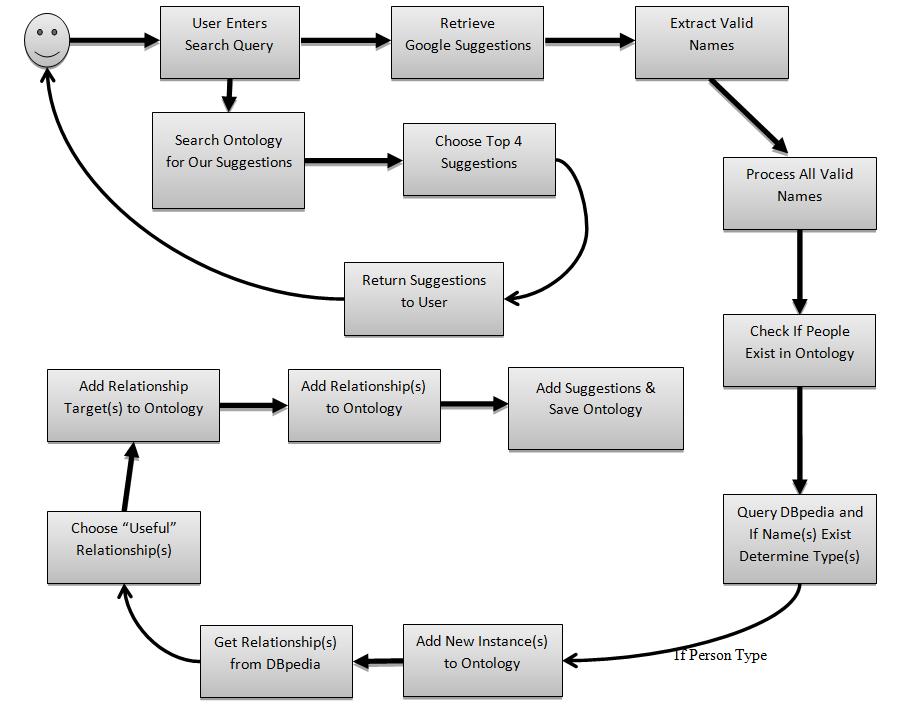
Our system works by analyzing user search queries and then including people who are commonly searched for into our ontology. While our system could be used to extract information on the over 360,000 people covered in the DBpedia ontology, in addition to other people not covered by the DBpedia ontology, we believe that this much knowledge would overwhelm users of our system. We do not feel it is practical to have potentially hundreds of possible search suggestions for each entered query. Instead, we chose to focus on providing suggestions only for people who our users feel are worth searching for. If the users of our system queried for all of the people within the DBpedia ontology our system would eventually cover the whole domain.

When a user enters a name, or a certain sized fragment of a name, into our system we query Google and retrieve their suggested search queries via their SOAP API. We check Google’s suggested queries for valid names, using census data for common names, as described previously. We then query DBpedia and determine if any of the possible names correlate to an actual person. If a name does correlate to a person then we look to see if they already exist in our ontology. If they do not exist we attempt to determine their type and create a new instance in our ontology.

If they exist in our ontology, but only as a stub instance, we promote the stub to a full instance and then treat it the same as a brand new instance. Using the relationships identified and structured previously, we query DBpedia for relationships and targets. If a given target does not exist in our ontology, we create a new instance for the target before we introduce the relationship.

Once the instance has been fully created within our ontology, it is added to the list of valid suggestions and returned to the user, along with any other previously existing instances that may be a search suggestion. This system ensures that the OSWS remains up to date with search suggestions, and the domain of our ontology expands with little input from us and no input from the end-user.

The process of dynamically expanding the ontology of famous people is diagramed in Figure 9.



**Figure 9.** Process of dynamically expanding the ontology of famous people.

## Generating Search Suggestions

The search suggestion mechanism remains largely unchanged from the first iteration of the OSWS. The end-user is still presented with up to twelve search suggestions comprised of at most four person concepts. One change from the previous version is that instead of using hit counts to order the suggestions, we instead use the total number of relationships coming out of a given instance. The assumption is that the more relationships one concept has, the more popular they are since there is more information about them.

A second change to how search suggestions are generated is based on lack of a certain relationship from a concept. If a relationship is within a class’s domain, but a given instance does not have said relationship, we still provide a search suggestion in the form of [Instance Name] [Relationship Name]. Even though no relationship exists in our ontology, this search suggestion should still give valid search results. We do this only for class-specific relationships only to avoid issues such as a “Died in” relationship, which is in the domain of all people, being shown for someone who is very much still alive.

Search suggestions now have more user-friendly versions of the relationships names in our ontology. For example, in our ontology the relationship may be named “starsInFilm” but the user sees “stars in film.” The same is also done for class names. This more verbose form of search suggestions removes the end-user from the underlying structure of our ontology and provides information he or she is more likely understand. Finally, we implemented the tilde operator for many parts of our search suggestions. In Google, the tilde operator searches for synonyms of a given word or phrase. By using this operator we aim to further narrow Google’s search results.

# Conclusions & Future Enhancements

In this report I introduced the concepts of query suggestions, ontologies, and Wikipedia. Next I reviewed the enhancements I made to the Ontology-Supported Web Search system. With my project I designed a system to extract information from DBpedia for use in the OSWS. My system analyzes Google Autocomplete suggestions and searches for valid names within those suggestions. The system then queries DBpedia and extracts information about famous people into the OSWS ontology.

I then integrated the ontology built using this method into the OSWS front end, including a number of functionality enhancements. Finally, I integrated the ontology building system into the front end so our ontology could be dynamically expanded based on end-user search queries. The goal is to keep the ontology up to date with helpful, usable search suggestions.

Aside from the project itself, the most important aspect of this project was turning the research and development described in this report into an academic paper. As of writing this report we are in the process of finalizing a paper entitled “*Google Knows Who Is Famous Today - Building an Ontology from Search Engine Knowledge and DBpedia,”* which will be submitted to ICSC 2011 (Fifth IEEE International Conference on Semantic Computing) at Stanford University.

Since I will be continuing as a PhD student at NJIT I plan to continue my research on the Ontology-Supported Web Search system. Future work involves further refining the OSWS ontology’s class hierarchy to provide better search results and exploring the use of new sources of data to include within our ontology. For the first version of this new ontology we directly used the DBpedia Person hierarchy, which is shallow but provides reasonable granularity for search suggestions.

However, while comparing YAGO types to DBpedia ontology classes, we found that there were quite a few missing class types in the DBpedia Person hierarchy. For example, movie directors are grouped into the Actor class. Martin Scorsese is classified as an actor instead of a director. By further analyzing class names from other sources, we believe we can provide better search suggestions in a number of cases. Additionally, we will continue to improve our domain coverage. By improving on the method of mapping Wikipedia abstract to DBpedia ontology described in Section III we believe we can add many new instances to our ontology.

We also recognized the problem of DBpedia presently being a static data source, when our ontology is trying to dynamically adapt to today’s searching environment. DBpedia updates its ontology roughly twice a year. If we rely on solely DBpedia as a data source we may not find up to date information on certain instances in our ontology, or miss a new instance entirely. DBpedia is working on a live-extraction system, but it is currently in a beta-state. In future iterations I will look for new, more frequently updated data sources to augment the ontology created for this project with new data. By including new sources I will be able to provide up to date and relevant search results to the theoretical end-users of this system.

# Acknowledgements

I would like to personally thank Dr. James Geller for the opportunity to work on this project and for introducing me to his Semantic Web research last summer. His guidance and input on this project was invaluable. This project has been an incredibly interesting research experience and I look forward to continuing research in this area. I would also like to thank Tian Tian for her input and assistance throughout this project. Both Dr. Geller and Tian have been fantastic to work with and I look forward to continuing my work in this area.

**References**

[1] “Autocomplete: Features.” Internet: http://www.google.com/support/websearch/

bin/answer.py?hl=en&answer=106230, [October 15, 2010].

[2] “Ontology Supported Web Search.” Internet: http://osws.njit.edu, June 2010\* [October 15, 2010].

[3] “Yahoo Search Assistant.” Internet: http://tools.search.yahoo.com/newsearch/

searchassist.html. [October 15, 2010].

[4] “Bing Search Suggestions.” Internet: http://help.live.com/help.aspx?project=

wl\_searchv1&querytype=keyword&query=tseggusotua&mkt=en-US. [October 15, 2010].

[5] Tian Tian, James Geller, Soon Ae Chun: “Improving Web Search Results for Homonyms by Suggesting Completions from an Ontology.” In: 2th International Workshop on

Semantic Web Information Management - SWIM 2010 (2010).

[6] Christian Bizer, Jens Lehmann, Georgi Kobilarov, et. al.: DBpedia – A Crystallization Point for the Web of Data. Journal of Web Semantics: Science, Services and Agents on the World Wide Web, Issue 7, Pages 154–165, 2009.

[7] Christian Becker: DBpedia – Extracting structured data from Wikipedia. Presentation at Wikimania 2009, Buenos Aires, Argentinia, August 2009.

[8] Christian Bizer et al.: DBpedia – Querying Wikipedia like a Database. Presentation at WWW2007.

[9] “WordNet Search – 3.0.” Internet: http://wordnetweb.princeton.edu/perl/

webwn?s=ontology. [October 15, 2010].

[10] “A Brief History of Search Engines.” Lee Underwoord. Internet: http://www.webreference.com/authoring/search\_history/. August 18, 2004\* [October 15, 2010].

[11] “Google.” Internet: http://www.google.com/. [October 15, 2010].

[12] “Google Search Share Up After Instance, Yahoo Down After Bing Transition.” Danny Sullivan. October 13, 2010\* [October 15, 2010].

[13] “Corporate Information – Google.” Internet: http://www.google.com/corporate/

tech.html. [October 15, 2010].

[14] ”DBpedia Mappings.” Internet: http://mappings.dbpedia.org/server/ontology/classes. [October 15, 2010].

[15] ”About: Michel Jackson.” Internet: http://dbpedia.org/page/Michael\_Jackson. [October 15, 2010].

[16] “Wikipedia.” Internet: http://www.wikipedia.org/. [October 15, 2010].

[17] ”About: Tom Hanks.” Internet: http://dbpedia.org/page/Tom\_Hanks. [October 15, 2010].

[18] Fabian Suchanek, Gjergji Kasneci, Gerhard Weikum: “YAGO: A Core of Semantic Knowledge Unifying WordNet and Wikipedia.” In: WWW2007 / Semantic Web Track. May 12, 2007\* [October 15, 2010].

# Appendix A: Developer’s Manual

The Ontology-Supported Web Search interface and backend are a complex software system composed of a number of inter-connected systems. The frontend has been written entirely in Java using the latest version of Google Web Toolkit (GWT), an SDK produced by Google for developing JavaScript programs in Java. The middleware of the system is queried using the Remote Procedure Call system provided by GWT. The middleware is composed of a number of subsystems which were implemented as modules into the system.

The purpose of this appendix is to provide future developers with a short reference guide on how to work within OSWS. Over the following pages I will describe how to access the source code for the OSWS project, run the OSWS project, and how to deploy the OSWS project onto NJIT’s AFS servers such that it is accessible from <http://osws.njit.edu>. By the end of this appendix the developer should be able to perform all of the tasks mentioned above.

## A.1 Setting Up the Development Environment

The OSWS system was developed using the latest technologies for Java. I developed OSWS using the Netbeans Java IDE, available here: <http://netbeans.org/downloads/index.html>. You must download the version that includes Apache Tomcat since OSWS uses Java Servlets. Netbeans isn’t a strict requirement; a fellow developer who worked on a separate module of OSWS was able to successfully run the project in Eclipse by copying the source files into an Eclipse project.

In addition to a Java IDE, you will need to install Google Web Toolkit, which is available here: <http://code.google.com/webtoolkit/>. If you choose to use Netbeans you will need to install the GWT4NB plugin from: <http://plugins.netbeans.org/plugin/716/gwt4nb>. There is an equivalent plugin by Google for Eclipse, available at the Google Web Toolkit link provided above.

Once you have installed all of the required software, start Netbeans and install the GWT4NB plugin and link it to where you extracted the Google Web Toolkit SDK.

## A.2 Retrieving the OSWS Source Code

The source code for the OSWS system, as well as the ontology-building code used to generate the A-List ontology, are both stored on NJIT’s CVS (Concurrent Versioning System) server. To access this server NJIT’s system administrator must give you access. If you ask Dr. Geller he will be able to contact the current system administrator and have you added to the access list. If you are outside of NJIT’s network, such as accessing from home, you must be connected to NJIT with their recommended VPN client.

To access the server, start the Netbeans IDE and navigate to the Team -> CVS -> Checkout menu item.



Next you’ll need to enter our server’s information. Enter the following:



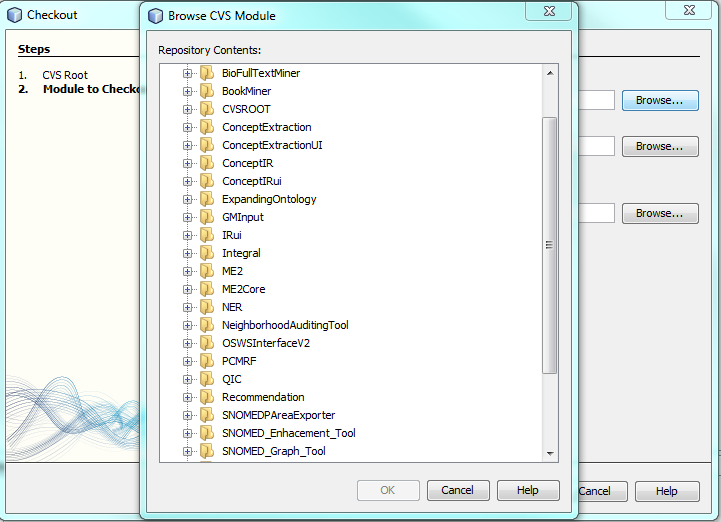
Under CVS root enter the following:

:**ext:yourUCID@cvs.njit.edu:/cvs**

Where “yourUCID” is the ID you use to access resources within NJIT network, such as logging into the AFS machines. Make sure “Use Internal SSH” is checked, and then enter your UCID password. You’ll be brought to the “Module to Checkout” screen.



Under Module select “Browse,” then expand the /cvs folder. Select both the   
“Expanding Ontology” and the “OSWSInterface2” folders then click OK. After clicking OK, click “Finish” on the Module to Checkout window. Then, when prompted, choose to open both projects into Netbeans.



The projects contain all of the required libraries for the Protégé API used to build the ontology, as well as the ontology files. The OSWSInterface2 project is the main OSWS system that can be deployed onto NJIT’s AFS servers. To run the project locally, simply right click on the project and choose “Debug.” This will launch Apache Tomcat and the Google Web Toolkit hosted code interpreter/debugger. The OSWS interface will start in your default browser.

## A.3 Deploying OSWS to osws.njit.edu

Deploying servlets is notoriously difficult on NJIT’s AFS servers. Since you do not have access to the Apache Tomcat logs, and the configuration is slightly different from a default Tomcat install, it may take a few attempts to get this working, but I’ve found a few simple steps to avoid the most common problems.

First, you’ll need access to the OSWS system’s directory. Either Dr. Geller or the current system administrator can add you to the OSWS’s user group. Once you have access you will be able to upload files into the OSWS web server, located at:

**/afs/cad/research/ccs/geller/osws/webapps**

The next step is making sure the OSWS front end is configured to find the GWT RPC endpoint. GWT assumes a default Tomcat installation, which will result in the front end not detecting the middleware. In the MainEntryPoint method, located in the MainEntryPoint.java as part of the OSWSInterface2 project, make sure the first couple of lines are set to the following:

*ServiceDefTarget endpoint = (ServiceDefTarget) getService();*

*endpoint.setServiceEntryPoint("http://osws.njit.edu/ExpandingOntology/ontologysuggestservice");*

This will allow the OSWS to function properly. After these lines of code are properly set, right click on the OSWSInterface2 project in Netbeans and select “Clean and Build.” Using your favorite secure file transfer program (either SSH or Putty) connect to any of NJIT’s AFS machines and navigate to the OSWS directory provided above. Next, on your local system, navigate to where you saved the OSWS projects and open the OSWSInterface2 folder. Open the “web” folder and then transfer the entire contents into the OSWS folder.

The OSWS system should now be available by accessing <http://osws.njit.edu>.

# Appendix B: OSWS Project Source Code

The source code for the entire OSWS project, including the Ontology Builder and related utilities, can be accessed from the /SourceCode folder on the accompanying compact disc. Due to the number of lines of code (over 6000) in this project is it not practical to include the source code within this document.

The source code has been separated into two different subfolders within the SourceCode folder. OntologyBuilder/src contains all of the code used to build the A-List ontology. OSWSInterface/src contains all of the source code for the OSWS Interface, along with the code for dynamically expanding the ontology.