Software-Developer Semantic Search System

S636 Semantic Web, Dr. Ying Ding

Chaoqun Ni
Bing He
Jiayi Lu

12/11/2009
1 Introduction
Since proposed by Father of Internet-Tim Boners-Lee, the semantic web has seen a growing popularity in standard institutions like W3C, academic community like Stanford University, and individual websites, like swoogle. The semantic web envisions a World Wide Web in which data is described with rich semantics and applications can pose complex queries. Ontologies, a cornerstone of the semantic web, have gained wide popularity as a model of information in a given domain that can be used for many purposes, including enterprise integration, database design, information retrieval and information interchange on the World Wide Web. In addition, reasoning is one of the benefit that obtained from the formal specifications required in ontology; it is also one of the significant support in many implications of semantic web.

In this project, we intends to build an domain ontology representing the skeleton of the conceptual structure of the knowledge domain of software development. In the areas of software development, the professionals who have profound experience are the most valuable assets for commercial technical corporations and academic research institute. The development of semantic search entitled us to a chance to revolutionize the professional management and an open online exhibition of a holistic review of a developer, a company and even a programming language.

This ontology basically contains four major concepts, “Developer”, “Programming Language”, “Software” and “Software Companies”. It serves (1) to make explicit the relationships between the concepts of Software, Companies, and Developers and (2) to find out the advanced professionals who have much experience or can supervise software projects in different programming languages. These functions are basically realized by utilizing reasoning and queries through Jena.

2 Project Steps

2.1 Conceptual Design
The ontology developed in this study does not intend to reflect the real world of software literally; instead we aim to construct the skeleton of the main entities and relationships. Therefore, we have made several assumptions regarding the world:

1. A Developer can develop more than one piece of software; A piece of software can be developed by more than one Developers.
2. A software company can release more than one software; a software can be only be released by one particular software company.
3. A piece of software can only be coded with one particular programming language; a programming language can be used to code more than a piece of software.
4. A developer might have used more than one programming language; a programming language can be used by more than one developers.

5. A developer can only be employed by a particular software company; a software company can employ more than one developers.

The above assumptions have been realized through the facets of properties in Protégé.

### 2.1.1 Data Dictionary for Classes

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
<th>Instance</th>
<th>Object Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer</td>
<td>People who have ever developed a software</td>
<td>Chris</td>
<td>hasDeveloped;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Aaron</td>
<td>isEmployedBy;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jennifer</td>
<td>hasUsedLanguage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.......</td>
<td></td>
</tr>
<tr>
<td>ExperiencedDeveloper (subclass of Developer)</td>
<td>Developers who have developed at least three pieces of software</td>
<td>Inherited</td>
<td>Inherited from its Super Class “Developer”</td>
</tr>
<tr>
<td>LanguageGeneralist (subclass of Developer)</td>
<td>Developers who have used three kinds of programming languages</td>
<td>Inferred</td>
<td>Inferred from its Super Class “Developer”;</td>
</tr>
<tr>
<td>ProgrammingLanguage</td>
<td>The syntax and rules used to develop software.</td>
<td>C</td>
<td>usedInSoftware;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Java</td>
<td>languageUsedBy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Perl</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.......</td>
<td></td>
</tr>
<tr>
<td>Software</td>
<td>computer programs, procedures and documentation play in a computer system</td>
<td>Office</td>
<td>isReleasedBy;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MySQL</td>
<td>isDevelopedBy;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Eclipse</td>
<td>programmedInLanguage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>.......</td>
<td></td>
</tr>
<tr>
<td>SoftwareCompany</td>
<td>Companies which specialize in software development</td>
<td>Microsoft</td>
<td>hasReleased;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sun</td>
<td>hasEmployed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adobe</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>.......</td>
<td></td>
</tr>
</tbody>
</table>
Data dictionary of the object properties of our project is as follows.

Table 2 Data dictionary for object properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Domain</th>
<th>Range</th>
<th>Inverse Property</th>
<th>Facts</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasDeveloped</td>
<td>Developer</td>
<td>Software</td>
<td>isDevelopedBy</td>
<td></td>
</tr>
<tr>
<td>isDevelopedBy</td>
<td>Software</td>
<td>Developer</td>
<td>hasDeveloped</td>
<td></td>
</tr>
<tr>
<td>hasReleased</td>
<td>SoftwareCompany</td>
<td>Software</td>
<td>isReleasedBy</td>
<td>inverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>functional</td>
</tr>
<tr>
<td>isReleasedBy</td>
<td>Software</td>
<td>SoftwareCompany</td>
<td>hasReleased</td>
<td>functional</td>
</tr>
<tr>
<td>programmedInLanguage</td>
<td>Software</td>
<td>Programming Language</td>
<td>usedInSoftware</td>
<td>functional</td>
</tr>
<tr>
<td>usedInSoftware</td>
<td>ProgrammingLanguage</td>
<td>Software</td>
<td>programmedInLanguage</td>
<td>inverse</td>
</tr>
<tr>
<td>hasUsedLanguage</td>
<td>Developer</td>
<td>Programming Language</td>
<td>isProgrammedBy</td>
<td>functional</td>
</tr>
<tr>
<td>usedByDeveloper</td>
<td>ProgrammingLanguage</td>
<td>Developer</td>
<td>hasProgrammedIn</td>
<td></td>
</tr>
<tr>
<td>hasEmployed</td>
<td>Developer</td>
<td>SoftwareCompany</td>
<td>isEmployedBy</td>
<td>functional</td>
</tr>
<tr>
<td>isEmployedBy</td>
<td>SoftwareCompany</td>
<td>Developer</td>
<td>hasEmployed</td>
<td>inverse</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>functional</td>
</tr>
</tbody>
</table>
Figure - 1 Classes and Properties in Protégé

Figure – 2 Individuals in Protégé
2.1.3 Class hierarchy
Our class hierarchy is shown as follows.

![Class Hierarchy Diagram]

Figure 3 Class Hierarchy

2.2 Visualization

We visualize our classes and related object properties in MicroSoft Viso 2003.

![Classes and Object Properties Diagram]

Figure – 4 Classes and Object Properties Diagram
3 Reasoning and Queries Design

Reasoning in ontology means deriving facts that are not expressed in ontology or in knowledge base explicitly. According to W3C, the computational complexity of the most relevant reasoning problems in OWL are as follows:\(^1\)

- **Ontology Consistency**: Check whether a given ontology has at least one model.
- **Concept Satisfiability**: Given an ontology O and a class A, verify whether there is a model of O in which the interpretation of A is a non-empty set.
- **Concept Status**: Given an ontology O and two classes A, B, verify whether the interpretation of A is a subset of the interpretation of B in every model of O
- **Instance Checking**: Given an ontology, an individual \(a\) and a class \(A\), verify whether \(a\) is an instance of \(A\) in every model of the ontology.
- **Conjunctive Query Answering**: Given an ontology O and a conjunctive query q, return the answers of the query with respect to O.

Here in this project, two kinds of reasoning have been designed here: subsumption and class membership.

3.1 Subsumption

We can make inferences about relationships between classes, in particular subsumption between classes; for example, recall that A subsumes B when it is the case that any instance of B must necessarily be an instance of A.

(1) Reasoning & Queries – 1

We defined the class “ExperienceDeveloper”, as a subclass of owl:thing, that it equals developers who have developed at least three pieces of software. By using the reasoning machine - RacerPro\(^2\), the class “ExperienceDeveloper” has been inferred as a subclass of the class “Developer”, which is defined as people who have developed at least one piece of software.

---

1. http://www.w3.org/Submission/owl11-tractable/
Similarly, We defined the class “LanguageGeneralist”, as a subclass of owl:thing, that it equals developers who have used at least three kinds of programming languages. By using the reasoning machine - RacerPro, the class “LanguageGeneralist” has been inferred as a subclass of the class “Developer”, which is defined as people who have used at least one kind of programming language.

3.2 Class membership

We may infer that a certain individual x is a member of a certain class D. Based on the semantic structure we have developed in the protégé, we have designed five sets of reasoning rules and five queries according to each set of rule.

(1) Reasoning & Queries - 1

• Which software company has employed the developer “Aaron”?
  − Rule:
    • ?x software:hasReleased ?y,
    • ?y software:isDevelopedBy ?z

(2) Reasoning & Queries – 2

Figure – 5 Reasoning Process of Submission in protégé
• \( \rightarrow \) ?x software:hasEmployed ?z
  
  – Result:

![Screenshot of interface]

**Figure 6** Result of Reasoning & Queries - 1

(2) **Reasoning & Queries – 2**

• What software have been released by “Microsoft”?
  
  – Rule:

    • ?x software:hasEmployed ?y,
    
    • ?y software:hasDeveloped ?z
    
    • \( \rightarrow \) ?x software:hasReleased ?z
  
  – Result (screenshot of interface)
(3) Reasoning & Queries – 3

- Which programming languages have the developer “Chris” used?
  
  - Rule:
    
    - ?x software:hasDeveloped ?y,
    
    - ?y software:isProgrammedIn ?z
    
    - \( \rightarrow \) ?x software:hasUsedLanguage ?z

- Result (screenshot of interface)
(4) Reasoning & Queries – 4

• Who is an experienced developer?
  
  – “ExperiencedDeveloper” equals Developers who have developed at least three pieces of software;
  
  – Rule:
    
    • (?x software:hasDeveloped ?y) owl:minCardinality 3

However, Reasoning & Queries 4 hasn’t been realized through Jena Reasoning. Functions of Jena Reasoning Support used to set the cardinality of a property should further be explored.
4 Interface

4.1 Tool and Work Process
For this project, what we are supposed to is just build ontology in Protégé, realize some reasoning and return some query results in Eclipse. But later we decide to create an interface for this project, since it will be more use-friendly.

The main tools used to create our interface are MySQL, MyEclipse, Tomcat, etc.

MySQL is a relational database management system (RDBMS) that has more than 11 million installations. MySQL stands for "My Structured Query Language". The program runs as a server providing multi-user access to a number of databases. MySQL is free and used here to store our data.

MyEclipse is a multi-language software development environment comprising an IDE and a plug-in system to extend it. It is written primarily in Java and can be used to develop applications in Java and, by means of the various plug-ins, in other languages as well, including C, C++, COBOL, Python, Perl, PHP, and others. The IDE is often called Eclipse ADT for Ada, Eclipse CDT for C, Eclipse JDT for Java and Eclipse PDT for PHP. MyEclipse is used in our project as the software development environment.

Tomcat is a Java Servlet container and web server from the Jakarta project of the Apache software foundation. A web server dishes out web pages in response to requests from a user sitting at a web browser. But web servers are not limited to serving up static HTML pages; they can also run programs in response to user requests and return the dynamic results to the users browser. Tomcat is very good at this because it provides both Java servlet and JavaServerPages (JSP) technologies (in addition to traditional static pages and external CGI programming). Tomcat is used in our project in as a web application server.

4.2 Coding
Coding is the hardest part in our project. We mainly uses JAVA as programming language and aim to create our project as a web search system. Though we call it “interface”, it is kind of independent from what as been done in the earlier parts. The servlet file contains more than 700 hundred rows of code, and is too long to include it in our report. The servlet file will be submitted in a form of simple java file.

4.3 Result
The interface of our project is quite simple, but has realized all the functions needed due to our project. As we can see from the following figure, we can choose any of the four categories in the drop box and input our query. Then our servlet will send our query to a web application server where our codes are deployed. According the query sent by our servlet, the database is connected and the related result will be returned to user by servlet and displayed to users. Here are some pictures of search result in our interface.
Figure 9  Search interface

Figure 10  Result returned by developer query
5 Limitations and future work

Although we have tried our best to work for our project, it is still not perfect and has some limitations. For example, the reasoning in our project is simple, the data is fictitious and the interface display is simple.

As for our reasoning, we have thought about some, but only several of them have been realized in our project, due to some limitations of reasoning tools in Protege and Jena. For example, we want to infer out “Language Generalist”, the developer who has used 3 programming languages to develop software. This reasoning is not realized in our project. Maybe we should try to realize our complex reasoning via some programming work, or consult more data about “rules” in semantic web.

In addition, the data used in our project is kind of fictitious. The data used in our project is just for examplery display and has little real meanings. For example, we assign Tom, a developer employed by MicroSoft Company, has developed Microsoft Office with Language C. But it is not the actual case in real world. The reason why we do this is that it is hard for us to find out who have actually participated in developing a given software, since software development is a huge task and always have more than 10 developers working.
together. Therefore, we just assign some imaginary developers for given software. If we would go further on this topic, we will add actual developer names of software and make it more reliable for users.

Finally, the search interface is kind of simple. Due to our time limitation, we just realize the required functions of our project. If have the chance, we would do some job, such as use CSS technology in our project in order to have a more beautiful interface.

To summarize, we learn a lot from this project. Although we realize the required function as a term project, there are still some limitations. If have the chance, we will go further and try to cover the limitations mentioned above.