DISSERTATION
Configurable Meta-search in the Human Resource Domain

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Univ.-Prof. Dr. Jürgen Dorn
Abteilung für Datenbanken und Artificial Intelligence
E184/2 Institut für Informationssysteme

und

Priv.-Doz. Dr. Nysret Musliu
Abteilung für Datenbanken und Artificial Intelligence
E184/2 Institut für Informationssysteme

ingereicht an der
Technische Universität Wien
Fakultät für Informatik

von
Tabbasum Naz
0527540
Schelleingasse 36/519
A-1040, Wien, Österreich

Wien, im Mai 2009
DISSERTATION

Configurable Meta-search in the Human Resource Domain

Thesis submitted in partial fulfillment of the requirements for the
Degree of Doctor of Technical Sciences
under supervision of

Univ.-Prof. Dr Jürgen Dorn
Database and Artificial Intelligence Group
E184/2 Institute for Information Systems

and

Priv.-Doz. Dr. Nysret Musliu
Database and Artificial Intelligence Group
E184/2 Institute for Information Systems

at Vienna University of Technology
Faculty of Informatics

by

Tabbasum Naz
0527540
Schelleingasse 36/519
A-1040, Vienna, Austria

Vienna, May 2009
CONFIGURABLE META-SEARCH IN THE HUMAN RESOURCE DOMAIN

Abstract

The Web has drastically changed the online availability of data and the amount of electronically exchanged information. However, the volume and heterogeneity of the information that is available online via Websites or databases make it difficult for a user to visit each and every Website that is relevant to the information needed. Primary search tools i.e. search engines, subject directories and social network search engines are not enough to meet the requirements of the information seeker.

Traditional search engines are based on keyword or phrase search, without taking into account the semantics of the word or phrase, and hence may not provide the desired results to the user. Other traditional search tools suffer from low recall and precision. These tools do not provide comprehensive coverage of the Web. To overcome these problems, meta-search engines aim to offer topic-specific search using multiple heterogeneous search engines.

In the human resource domain, traditional methods of job/employee search i.e. newspapers, magazines, advertising at job fairs, employment recruitment agencies and registering with search firms, lack the ability for search in the modern employment market. In this dissertation, we propose a new configurable meta-search engine in the human resource domain to provide an ideal platform for meta-search provider and a job seeker. Our aim is to combine the respective benefits of vertical search engines, meta-search engines and semantic search engines within a domain-specific context, in which there is a well-understood domain ontology.

We are concerned with techniques to support two key aspects of meta-search engines: i) meta-search engine creation by meta-search engine providers and ii) meta-search engine usage for information seekers. One of the important challenges in accessing heterogeneous and distributed data via a meta-search engine is schema/data matching and integration. We describe an approach to schema and data integration for meta-search engines. During the matching and integration process, we need to handle syntactic, semantic and structural heterogeneity between multiple information sources.

In this dissertation, our main objective is to resolve semantic conflicts. Our approach is a hybrid one, in that we use multiple matching criteria and multiple matchers. We employ several element levels, structure levels and ontology based techniques during the integration process. A domain ontology serves as a global ontology and allows us to resolve semantic heterogeneity. Our matching process handles different mapping cardinalities (1:1, 1:n, n:1, m:n). The mappings derived are used to generate an integrated meta-search query interface, to support query processing in the meta-search engine, and to resolve semantic conflicts arising during result extraction from the source search engines. Experiments conducted in the job search domain show that the cumulative use of element-level, structure-level and ontology-based techniques increases the correctness of matching during the automatic integration of source search interfaces.

The system supports meta-search provider in the quick development of meta-search engines and is able to understand and integrate schemas from different job search
engines semantically. Meta-search provider can easily integrate the new search engines in the meta-search engine. The system can help job seekers in the job search without visiting multiple search engines. Job seekers do not need to spend their time to comb through large numbers of job results in searching for the relevant job. The system can semantically understand the job results and rank them for the job seekers.

An important aspect of our meta-search in human resource domain is that it has been designed by applying semantic Web technologies, to solve the problems of meta-search developers and job seekers. We provide the solutions for automatic integration of data, structures and processes in human resource domain into a meta-search by the use of our modelled domain ontology and multiple matchers. We have used HR-XML and different classification schemes in the construction of domain ontology and integrated interface for the meta-search engine. Our modelled domain ontology and HR-XML for the generation of integrated schema and integrated interface are used to understand the meaning of terms and to improve the quality of search interface and search results.

Flexible and re-useable design patterns have been introduced for the creation process, usage process and different components of meta-search engine. Design pattern for the creation process helps the meta-search provider and design pattern for the usage process helps the job seeker. Design patterns for different components of meta-search engine help the new developers to speed up the development process.

Meta-search increases the Web coverage for job seeker by the combination of specialized search engine, multiple search engines and semantic search into one. We hope that new meta-search engine can be helpful in reducing the unemployment rate of a country.
Acknowledgement

“To Him belongs the dominion of the heaven and the earth, it is He who gives life and death and He has power over all things.”

(Al-Quran)

I would like to express my sincere and humble gratitude to Almighty Allah whose blessings, help and guidance has been a real source of all achievements in my life.

I would like to express my gratitude to my supervisor Prof Dr Jürgen Dorn who introduced me to new dimensions of knowledge, for his guidelines, encouragements and valuable suggestions that helped me in completion of my PhD. He always helped me out in every problem during my research. I really wish to express my appreciation to him for his full cooperation, kindness, patience, thoughtfulness and continuous support that made this work a reality.

My sincere thanks to Prof Alexandra Poulovassilis of London Knowledge Lab, School of Computer Science and Information Systems, University of London, UK, who provided me an opportunity to improve my research work at London Knowledge Lab.

I am grateful to Dr Nysret Musliu for encouragements and for his precious time for final readings of manuscript.

I am thankful to Higher Education Commission, Pakistan for the financial support of my study in Austria.

And finally I admit that I completed my PhD due to my loving parents who always pray for my success.

Vienna, May 2009
Tabbasum Naz
# CONFIGURABLE META-SEARCH IN THE HUMAN RESOURCE DOMAIN

## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Acknowledgement</td>
<td>iii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>v</td>
</tr>
<tr>
<td>A- List of Figures</td>
<td>ix</td>
</tr>
<tr>
<td>B- List of Tables</td>
<td>v</td>
</tr>
<tr>
<td>C- Listings</td>
<td>xi</td>
</tr>
<tr>
<td>D- List of Algorithms</td>
<td>xii</td>
</tr>
</tbody>
</table>

## 1 INTRODUCTION

1.1 Online Search Approaches | 1 |
1.2 Meta-search Engines: The Introduction | 3 |
  1.2.1 Meta-search Engine Components | 3 |
  1.2.2 Types of Meta-search Engines | 5 |
  1.2.3 Advantages of Using Meta-search Engines | 5 |
1.3 Challenges | 6 |
1.4 Overview of Dissertation | 10 |

## 2 LITERATURE REVIEW

2.1 Internet Recruitment | 12 |
  2.1.1 Parts of Internet Recruitment | 12 |
    2.1.1.1 Recruiter | 12 |
    2.1.1.2 Job Seeker | 13 |
  2.1.2 Existing application for Internet recruitment: JobOlize | 13 |
  2.1.3 Problems with Internet Recruitment | 13 |
2.2 Internet Technologies | 14 |
  2.2.1 XML Technologies | 14 |
    2.2.1.1 XML | 14 |
    2.2.1.2 W3C XML Schema | 14 |
  2.2.2 HTML | 16 |
  2.2.3 Forms | 17 |
    2.2.3.1 HTML Forms | 17 |
    2.2.3.2 XForms | 18 |
    2.2.3.3 Search Interface Forms | 19 |
2.3 Existing Meta-search Solutions | 19 |
  2.3.1 WISE-Web Interfaces of Search Engines | 20 |
    2.3.1.1 Web Search Interface Extraction | 20 |
      2.3.1.1.1 Attribute Extraction | 20 |
      2.3.1.1.2 Attribute Analysis | 21 |
    2.3.1.2 Interface Schema Matching | 21 |
      2.3.1.2.1 Positive Match Based Clustering | 21 |
      2.3.1.2.2 Predictive Match Based Clustering | 21 |
    2.3.1.3 Global Attribute Generation | 22 |
    2.3.1.4 Unified Search Interface Construction | 23 |
    2.3.1.5 Unified Interface Maintenance | 23 |
2.3.2 Lixto Meta-search.......................................................... 23
2.3.3 Snorri Meta-search....................................................... 24
2.3.4 MetaQuerier............................................................... 24
2.3.5 Wrapper Generation................................................... 25
2.3.6 Result Merging and Ranking........................................ 26
2.3.7 Existing Design Patterns for Domain Specific Search
   Engines........................................................................ 26
2.4 Schema and Ontology Matching/Mapping Approaches......... 26
2.5 Semantic Web................................................................... 30
  2.5.1 What Is an Ontology?.................................................. 30
  2.5.2 Main Components of Ontology.................................... 31
  2.5.3 Kinds of Ontologies.................................................. 31
  2.5.4 Ontology Applications.............................................. 32
  2.5.5 Semantic Web: Languages........................................ 32
    2.5.5.1 RDF............................................................... 32
    2.5.5.2 RDF Schema.................................................... 33
    2.5.5.3 OWL............................................................... 33
  2.5.6 SPARQL................................................................. 34
2.6 Impact of Semantic Web on E-Recruitment......................... 34
2.7 Human Resource Management........................................ 35
  2.7.1 HR-XML.................................................................. 35
    2.7.1.1 Recruiting and Staffing..................................... 36
    2.7.1.2 Competencies.................................................. 36
    2.7.1.3 Resume.......................................................... 36
    2.7.1.4 Staffing Industry Data Exchange Standards 1.0 (SIDES)... 36
  2.7.2 Classification and Standards ..................................... 37
    2.7.2.1 Occupation Classifications................................ 37
    2.7.2.2 Competence/Skill Classifications........................ 38

3 SYSTEM REQUIREMENTS 40
  3.1 Meta-search Engine Provider’s Requirements.................. 40
  3.2 Job Seeker’s Requirements........................................... 41
  3.3 Other Requirements..................................................... 42

4 SYSTEM DESIGN 44
  4.1 Meta-search Architecture.............................................. 44
  4.2 Design Patterns for Meta-search Engines....................... 45
    4.2.1 Design Pattern for Query Interface-Generator............ 49
    4.2.2 Design Pattern for Information Extractor............... 50
    4.2.3 Design Pattern for Result Merger......................... 51
    4.2.4 Design Pattern for Result Ranker......................... 52
  4.3 System Design for Meta-search in Human Resource Domain... 53
    4.3.1 Search Engine Selector.................................... 55
    4.3.2 Interface Extractor........................................ 55
      4.3.2.1 Attribute Extraction.................................. 56
      4.3.2.2 Attribute Analysis.................................... 57
    4.3.3 XML Schema Generator.................................... 58
      4.3.3.1 Schema Elements..................................... 58
      4.3.3.2 XML Schema Equivalents for HTML Elements.... 59
    4.3.4 Meta-search Query Interface Generator................... 59
      4.3.4.1 Schema Integration.................................... 59
4.3.4.2 Query Interface Generation ................................. 61
4.3.4.3 Data Integration .............................................. 61
4.3.4.4 Techniques used for Meta-search Query Interface
  Generation .......................................................... 63
  4.3.4.4.1 Element Level Techniques ........................... 63
  4.3.4.4.2 Structure Level Techniques ......................... 65
  4.3.4.4.3 Ontology Based Techniques ......................... 65
  4.3.4.4.4 Map Cardinality ....................................... 66
  4.3.4.4.5 Schema and Data Integration Algorithm ............ 67
4.3.5 Query Dispatcher .............................................. 69
4.3.6 Information Extractor ....................................... 71
  4.3.6.1 Record Collector ......................................... 71
    4.3.6.1.1 Job Record Section Identification on the Basis of
      Keywords ...................................................... 72
    4.3.6.1.2 Job Record Section Identification on the Basis of
      Dynamic Section ........................................... 73
    4.3.6.1.3 Job Record Section Identification on the Basis of
      Pattern Format ............................................. 73
  4.3.6.2 Result Collector ......................................... 76
    4.3.6.2.1 Label Extraction from Job Result Pages ......... 76
    4.3.6.2.2 Result Identification by Domain Ontology and
      Matchers ....................................................... 76
4.3.7 Result Merger .................................................. 78
4.3.8 Duplicate Result Eliminator ............................... 78
4.3.9 Result Ranker .................................................. 78
4.3.10 Collection of Preferences ................................ 79
4.3.11 Cache Management ......................................... 80
4.4 Ontology Development for Human Resource Management.... 80
  4.4.1 Occupations Sub-Ontology ................................. 81
  4.4.2 Competencies Sub-Ontology ............................... 82
  4.4.3 Levels and Grades for Competencies .................... 83

5 IMPLEMENTATION 85
  5.1 Meta-search Architecture from Implementation View ....... 85
  5.2 Java Servlets .................................................. 86
  5.3 Jena Framework ............................................... 87
  5.4 MySQL Java Connector ....................................... 87
  5.5 MySQL .......................................................... 87
  5.6 Tomcat .......................................................... 88
  5.7 Protégé .......................................................... 88
  5.8 GeoLite Country ............................................... 88
  5.9 Mozilla Xform Plugin ........................................ 89
  5.10 Document Object Model (DOM) API ......................... 89

6 EVALUATION 91
  6.1 Case Study ..................................................... 91
    6.1.1 Query Interface Generation for the Job Meta-search Engine... 91
    6.1.2 Query Processing by the Job Meta-search Engine .......... 93
  6.2 Evaluation of Processes .................................... 93
    6.2.1 Evaluation of Job Meta-search Engine Creation Process .... 93
      6.2.1.1 Evaluation of Interface Extractor .................... 94
6.2.1.2 Evaluation of XML-Schema Generator.............................. 96
6.2.1.3 Evaluation of Query Interface Generator ....................... 98
6.2.2 Evaluation of Job Meta-search Engine Usage Process .......... 100
  6.2.2.1 Evaluation of Query Dispatcher .................................. 100
  6.2.2.2 Evaluation of Information Extractor ............................ 102
    6.2.2.2.1 Evaluation of Record Collector ............................. 102
    6.2.2.2.2 Evaluation of Result Collector ............................. 102
  6.2.2.3 Evaluation of Result Ranker .................................... 103
6.3 Evaluation of Hybrid Approach in Schema Matching .............. 107
6.4 Performance ...................................................................... 108
6.5 Existing Problems and Required Improvements .................... 108

7 CONCLUSION AND FUTURE WORK 111
7.1 Conclusion ...................................................................... 111
7.2 Future Work .................................................................... 113

APPENDIX-A: TABLES AND LISTINGS ................................. 117
APPENDIX-B: SCREENSHOTS ................................................. 126
APPENDIX-C: LIST OF PUBLICATIONS ................................... 127
References ........................................................................... 128
List of Abbreviations .............................................................. 135
Index ..................................................................................... 136
LIST OF FIGURES

1.1 Architecture of a Simple Meta-search Engine .................................................. 04
1.2 a.Canjobs.com, b.careerbuilder.com, c.jobs.net ........................................... 09
4.1 Meta-search Creation Process Components .................................................... 44
4.2 Meta-search Usage Process Components ....................................................... 45
4.3 Abstract Factory Pattern for Meta-search Engine Creation Process .............. 47
4.4 Abstract Factory Pattern for Meta-search Engine Usage Process ................. 48
4.5 Strategy Pattern for Meta-search Query Interface Generator ...................... 49
4.6 Strategy Pattern for Meta-search Information Extractor .................................. 51
4.7 Abstract Factory Pattern for Meta-search Result Merger ............................... 52
4.8 Strategy Pattern for Meta-search Result Ranker ............................................ 52
4.9 Meta-search Creation Process Components for Meta-search Provider .......... 53
4.10 Meta-search Engine Usage Process Components for Job Seeker ................. 54
4.11 Job search Interfaces a. jobs.net; b. aftercollege.com; c. Careerbuilder.com ............... 55
4.12 Job search Interface of jobshejobs.com ..................................................... 56
4.13 a. Generic concept by placementindia.com, b. Specific concept by clickjobs.com, c. Combined concept ......................................................... 62
4.14 Semantic Nearness by Taxonomy Based Matching ....................................... 66
4.15 Job Result Records from http://www.canjobs.com ..................................... 72
4.16 Job Result Records from www.jobserve.com ............................................ 72
4.17 A Single Job Record ...................................................................................... 74
4.18 Ontology for Human Resource Tasks .......................................................... 81
4.19 Ontology with Occupations, Competencies and Learning Objects ............. 83
5.1 Meta-search Architecture from Implementation View ................................... 85
5.2 Basic Servlet Life Cycle ................................................................................. 86
6.2 Extracted Interface of www.jobs.net ............................................................ 94
6.3 Meta-search Integrated Interface .................................................................. 100
6.4 Query Transformation and Dispatching ......................................................... 101
6.5 Job Result Page from jobs.net ....................................................................... 104
6.6 a. Extracted Record Section from jobs.net , b. Extracted URLs and Job Titles from the Record Section of jobs.net ...................................................... 105
6.7 Identification of Job Attributes From jobs.net Result Page With the Help of Domain Ontology ................................................................. 106
6.8 Ranked Jobs from Multiple Search Engines on the Basis of Salary ............. 107
6.9 Evaluation of Hybrid Approach in Schema Matching ................................. 108
6.10 a. Part of Result Page from 6figurejobs.com, b. Part of Result Page from jobs.net ................................................................. 109
B.3 Integrated Interface of Job Meta-search ......................................................... 126
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Classification of Computer and Mathematical Occupations in SOC 2000</td>
<td>38</td>
</tr>
<tr>
<td>4.1</td>
<td>Meta Information for jobshejobs.com</td>
<td>58</td>
</tr>
<tr>
<td>4.2</td>
<td>Common Attributes in Job Domain</td>
<td>60</td>
</tr>
<tr>
<td>4.3</td>
<td>HR-XML Attributes and Schemas for Common Job Portal’s Interfaces</td>
<td>61</td>
</tr>
<tr>
<td>4.4</td>
<td>Levels to Measure Competencies</td>
<td>83</td>
</tr>
<tr>
<td>6.1</td>
<td>Evaluation of LEX Technique</td>
<td>95</td>
</tr>
<tr>
<td>6.2</td>
<td>Evaluation of Relationship Type of Elements</td>
<td>95</td>
</tr>
<tr>
<td>6.3</td>
<td>Evaluation of XML-Schema Generator</td>
<td>98</td>
</tr>
<tr>
<td>6.4</td>
<td>Concept Translation to HR-XML by Domain Ontology and Multiple Matchers</td>
<td>98</td>
</tr>
<tr>
<td>6.5</td>
<td>Evaluation of Record Collector</td>
<td>102</td>
</tr>
<tr>
<td>6.6</td>
<td>Evaluation of Result Collector</td>
<td>103</td>
</tr>
<tr>
<td>6.7</td>
<td>Evaluation of Result Ranker</td>
<td>104</td>
</tr>
<tr>
<td>A.1</td>
<td>List of Job Portals</td>
<td>117</td>
</tr>
</tbody>
</table>
LISTINGS

2.1 A Simple XML Schema .......................................................... 16
2.2 A Simple HTML Form .......................................................... 18
4.1 Integrated XL Schema Format for Meta-search Engine ........... 69
6.1 An XML Schema for www.jobs.net ....................................... 96
A.1 Domain Ontology in OWL .................................................... 118
A.2 Individual XML Schema for top-consultant.com ................... 122
A.3 Integrated XML Schema ..................................................... 124
LIST OF ALGORITHMS

4.1 Algorithm for Schema and Data Integration in Integrated XML schema for Job Meta-search Engine by using Domain Ontology and Multiple Matchers…………………………………… 67
4.2 Algorithm for Query Generation and Dispatching………………………………………… 70
4.3 Algorithm for Result Identification and Collection by Domain Ontology and Different Matchers.......................................................................................................................... 77
CHAPTER 1
INTRODUCTION

Unemployment is not only a serious problem of developing nations i.e. Asia, Africa, Latin America but also a problem of developed nations i.e. Europe. In Europe, unemployment rate increases sharply and almost continuously since the early 1970s. It increased further in the 1980s, to reach a plateau in the 1990s. It is still high today and increasing day by day [Ber06] [Bla06]. According to [PHR08] unemployment rates in 2008 are 9.10% in Germany, 7.50% in Pakistan, 4.30% in Austria, 4.60% in United States and 5.40% in United Kingdom. One of the causes of unemployment is problem in searching and distribution of jobs. Job opportunities are available in some areas but people are unable to reach them. In order to drop the unemployment rate there must be a solution so that people can easily search for jobs.

Access to the Internet has proven that the traditional methods of job searching i.e. newspapers, magazines, advertising at job fairs, employment recruitment agencies and registering with search firms are too slow, expensive and lacking in their ability for searching jobs in the shortest possible time in the modern employment market. Now a day other than the newspapers, two other ways of publishing and searching job postings are online job portals and the Web sites of organizations.

“The importance of the Internet for job procurement is increasing for the reason that three quarters of the people in the employment age are online” [FHK+06]. For a certain company, publishing online their job offers is a sign of good economic health, in that way, e-recruitment becomes vector of institutional publicity and ever more companies are publishing their job offers on the Web. Many candidates (job seekers) submit their curriculum vitae (CVs) using the different Web sites. In addition to a broader diffusion, this approach offers the job seeker both reactivity and facility when updating the personal information. Recruiters also like to automate the pre-selection of the candidates and want to save time and transaction cost of the company in the candidate selection process [HLT02].

Traditional job search engines lack the capability of processing structured queries and search is based only on statistical methods without any attempt to understand the meaning of the terms being used. Words can have synonyms, different associations and interrelationships, can be used in many senses and terms appear in different languages that cannot be handled by the traditional job search engines. For job seekers, access to a large number of job search engines is difficult and sometimes existing job portals produce several irrelevant results. As every search portal has different interface style, it can be confusing for the job seeker in giving job search criteria. Recruiting organizations also need some improvements in the existing job search portals and want to construct the search engines according to their own preferences.

1.1 Online Search Approaches

Internet is flooded with the information and contains hundreds of Web sites with thousands of topics that the searchers get lost while searching a topic. Second most popular activity on the Internet after e-mail is “search” [RS05]. One Microsoft researcher says: “Estimates are that information workers spend as much as 30 percent of their time searching for information, at a cost of $18,000 each year per employee in lost productivity” [Gat06]. Varieties of search tools are available for the searchers to
make the search process easy and fast so that searchers can use the information effectively. Primary tools to search the information on the Web are search engines and subject directories.

"With the ubiquity of Internet and Web, search engines have been spouting like mushrooms after a rainfall" [DB03]. Search engines use programs known as Crawlers, Indexing spiders, or Robots to crawl the Web and create search engine indices. Indexing spiders retrieve the documents from the Web servers for automatic indexing and include into their databases. Spiders or crawlers read the contents of Web pages and use their own algorithms to determine what information must be used in the search engine’s database. Spiders are required to frequently update their databases by visiting Web sites. For the successful crawling of Web pages it is required that a Web site is designed according to the needs of the crawler based search engines. Google and Alta Vista are examples of crawler based search engines. Even though a large number of search engines have been developed but searching the Web for a particular topic is difficult. The coverage of single search engine is limited.

Generally search engines cannot locate the information available via invisible Web because of technical limitations or because of exclusion from the indices. A single search engine covers not more than a third of all indexable documents on the Web and it returns long documents as search results. Searchers have to navigate through long documents to find the relevant information from these long documents. One of the Microsoft reports says: “people search an average of 11 minutes before they find what they are looking for” [Pet05]. Crawlers cannot locate the high quality information from the invisible Web stored in Web accessible databases. These Web accessible databases need direct interaction by filling the fields in the forms that cannot be done by the crawlers. Crawlers are designed to request and fetch Web pages only and unable to retrieve information from Web accessible databases that needs input from a form [SP01]. Moreover, some Web pages are customized via cookies, or use java script to access the content and crawlers are unable to reach such type of Web pages. Due to tremendous growth of Internet, it is not possible to box up the entire Web into a single database.

Subject directories are another type of tools for Web search. These directories are organized by subjects on the Internet sites. The users choose a subject of interest from the list of subjects. Human editor categorizes the Web site for Web directories. Short description of the site is submitted during submission of the Web site or human editors write the description for the sites they review. A change in Web pages has no effect on the listing because search looks for matches only in short descriptions. Yahoo is an example of a Web directory. Subject directories also cover only a small fraction of the pages from the Web and are effective only if searcher is looking for general information on popular or scholarly subjects. Subject directories depend on human editors for listings. If short description of the site is not specific enough then search may be unsuccessful.

According to Web coverage, both crawler based search engines and Web directories are further classified as “general search engines” and “specialized search engines”. General search engines are developed to meet the needs of general population and cover wide range of topics. Specialized search engines are developed to meet the needs of particular audience, cover particular topics. Specialized search engines provide greater and relevant coverage of the Web but even then they encounter low recall.
Another type of search tools known as social network search engines is getting mature in these days. Social network services allow people with shared interests, hobbies, or causes to come together online. Social network search engines are a class of search engines that use social networks to organize, prioritize, or filter search results. Social bookmark sites allow the Internet users to share the contents they like the best for others to search and view. Social bookmarking sites help the users to meet others sharing common interests. Examples of social bookmark sites are del.icio.us, digg, reddit etc. Social bookmark sites allow users to save links to Web pages they want to store or share. Users can assign tags to these favourite Web pages. Tags are keywords to describe the Web page. Users can make their bookmarks public and can share the relevant Web pages of interest with other users. The allowed users can view publicly available bookmarks by tags or category or chronologically with the help of search engines or even randomly. Social book marking has several advantages over traditional search engine spiders. In traditional search engine spiders, classification is done by programs who algorithmically determine the meaning of resource but in social networking tag-based classification is done by human beings, who understand the content of resource much better than programs and provide semantically classified tags. Social book marking can be used to enhance Web search results and sometimes can quickly produce relevant results. Some social book marking sites also provide a facility for rating and comments on book marks that can be helpful for the searchers in searching high quality results. But problems with such type of tag-based systems are that there is no set of controlled vocabulary for tags and no standard for tag structures. There also exist spelling errors and multiple meanings of tags. Some users are also misusing tags to make their Web sites more visible [HHL+05].

Internet search still needs improvement. Traditional search engines are based on keyword or phrase search without understanding the meaning of the word and are unable to provide the desired results to the user. Search tools discussed above have different index structures, low recall and precision and do not provide comprehensive coverage of the Web [DB03] [RRS03] [AFS]. To overcome all these problems, there exists another search tool called meta-search engine, considered as an excellent choice for a specific topic search. In this dissertation, we discuss the meta-search engine for jobs.

1.2 Meta-search Engines: The Introduction

Meta-search engines (MSE) also known as multi-threaded engines. They do not necessarily maintain their own listings/databases, but send the seeker’s query simultaneously to other search engines, Web directories or deep Web. They collect the results, merge them, remove the duplicate links, store results in a cache memory, rank them according to their own algorithm in a single list, and then display it to the seeker. Ixquick, Dogpile and Metacrawler are examples of meta-search engines.

1.2.1 Meta-search Engine Components

The main components of a meta-search engine are shown in Figure 1.1.
• **Query Interface**

Seeker can interact with other parts of search engine by query interface by specifying query-terms, parameters, matching type. Query interface of meta-search engine must contain the meaningful labels of elements and semantically unique values for different attributes.

![Architecture of a Simple Meta-Search Engine](image)

- **Cache Manager**

When a seeker sends a request by query interface component to the meta-search engine, it is first observed that required searched items are already available in the cache memory or not. If required items are already available in the cache memory then cache manager of meta-search engine will access and display items from the cache memory. But if required items are not available in the cache memory then the whole process of search starts from scratch. At the end of search process cache manager is responsible to store results in the cache memory for a short time. Accessing items from cache memory is faster as compared to search from scratch.

- **Search Engine Selector**

If required items are not available in the cache memory, search engine selector component of a meta-search engine is activated and the search process starts. It is not recommended to send the user query to all available search engines because some
search engines may produce useless results for a particular query term. This also wastes the time of a user, generates useless network traffic and wastes the resources in the identification of useless documents. Search engine selector is a component of meta-search engine that identifies and selects the potential useful search engines for querying.

- **Query Dispatcher**

Query from the interface of meta-search is required to convert according to each search engine’s format. Query dispatcher is used to convert and send the query of the user to other search engines. It also checks the request methods either GET or POST of each search engine. Then query dispatcher establishes HTTP connection with the server of each search engine and finally data is transferred from different search engines.

- **Result Merger**

Results returned from multiple search engines are combined with the help of result merger for further processing.

- **Duplicates Eliminator**

Results merged from different search engines may contain duplicate links. Duplicate eliminator of meta-search engine eliminates the duplicate links. Duplicate eliminator sends the searched items to the cache manager to store them in a cache memory for a short time.

- **Result Ranker**

Finally, results are ranked according to some algorithm and displayed to the seeker [MYL02] [RRS03].

### 1.2.2 Types of Meta-search Engines

Following are the three different types of meta-search engines:

- **Desktop Meta-search Engine** is a client software utility that needs downloading and installation on the searcher’s computer for meta-searching. They are not free of cost but most of them have free trial version available.

- **All-in-one Meta-search Engine** allows the user to get results from hundreds of different search engines, directories, and indices to query with the help of a single link or search boxes.

- **Hidden Web Meta-search Engine** is a specialized meta-search engine that provides access to search databases missed by other search engines and collect results for the searchers [Not06].

### 1.2.3 Advantages of Using Meta-search Engines

Use of meta-search engines is beneficial as compared to traditional search tools in the following way:
Meta-search engines provide fast and easy access to the desired search from multiple search engines simultaneously and save the precious time of the searcher.

Meta-search engines provide a broader overview of a topic as compared to traditional search engines.

Meta-search engines increase the coverage of the Web by combining the coverage of multiple search engines.

Querying multiple search engines is more scalable than the centralized general purpose search engine.

Meta-search engines make the user’s task much easier by searching and ranking the results from multiple search engines.

Sometimes search by meta-search engine is restricted to special search engines related to the query and this improves the retrieval effectiveness [MYL02].

Meta-search engines require smaller investment in hardware as compared to general search engines i.e. Google which uses thousands of computers and high storage capacity of each computer [MWY+01].

Meta-search engines have ability to search the invisible Web too thus improving the precision, recall and the quality of results [AFS].

1.3 Challenges

There exist many problems in online job search and need to be solved. For example, a large number of online job search portals are available and no one can access all of them. Due to the large number of the openings published online, it is almost impossible for job seekers to gain an overview of the entire employment market. Different job portals have different ways to search for a job. Some portals offer a so-called job agent. After registration the user arrives at the personal starting page to store the search criteria in a personal search profile. The job agent will scan new position openings on a regular basis e.g. two days or every week then check them against the user’s criteria and notify the user of suitable positions by e-mail. Other job portals offer direct searching e.g. quick search or detail search.

Every job search portal has different interface styles and attributes i.e. some job search portals use “Type of Hour” to mention the job type and other use “Employment Type”. Sometimes one term is used to represent different concepts in different search engines. For example, in one search engine “Employment Type” is used to represent “Information Technology, Business or Engineering etc” and in another search engine, it represents “Full time, Part time or Internship etc”. Sometimes it is required to mention religion in CV while in other it is optional. In one job portal salary is displayed in Euros and in other in Dollars. Job portals also vary in the range of services they provide. Some job search portals provide search by generic concepts while other use specific concepts i.e. job category is “Programming language” in case of generic concepts and “Java” in case of specific concepts. Some job portals provide job refinement while others do not. The job seekers provide information about region where they want to work, job category, type of job and relevant keywords etc. As only general data is submitted for job search, the search results returned by the job search engines are of not high quality and very large is number. Job seekers have to spend their time again to comb through these large numbers of results and it is not possible to maintain an overview of all job portals.

Traditional search engines for job search are simply based on combination of locations, list of keywords and sectors. This approach seems poor, ineffective and not beneficial
from a quality point of view, therefore is not helpful for both the job seekers and the recruiters [TE04]. CV or resume can be posted on the Web free of charge. This is done by firms with whom the person has direct contact or over the message box. The CV data is posted anonymously and the user retains the right to decide who will get the access to the complete CV. Job seekers do register their CVs with large number of recruitment companies and job portals therefore they loose the track of their resumes and cannot update them regularly. Job seekers face the problems to emphasize their competencies while writing their CVs and thus have difficulties to find job offers, which ideally correspond to their profiles. When a job seeker writes a CV, it is usually difficult to choose the best sentence (in the natural language) for expressing the competencies acquired during the professional career. Sometimes the sentences or words used to emphasize competencies in the CVs are not very significant and do not include or precisely reflect all the competencies of a person [HLT02]. Job offers lack semantically meaningful annotation therefore search and integration into databases is highly difficult [FHK06].

Traditional job search engines process the queries without understanding the meaning of the term and they are unable to understand the synonyms of words, different associations and interrelationships. Semantic search is an application of the semantic Web to search. The technological foundations of the semantic Web are getting mature. Semantic search engines are equipped with the information about the meaning of terms and also contain interconnections between related terms with the help of ontologies and meta-data. Ontologies have improved the precision and recall of search engines. Ontologies can be used to organize the information resources in a better way. Ontologies also assist users in retrieving the relevant information efficiently. In semantic search engines the term is not treated as a string and result is not based on the frequency of the string but the search engine can exploit the term relationships to get the direct information about the term. The user can jump to the semantically related node for a particular concept via relationships and can examine similar or related concepts [BP02]. Ontologies allow applications to agree on the terms for easy communication and thus increase the interoperability, in the semantic Web. Semantic Web search using ontology provides explicit conceptualization for entities in a specific domain and annotation for the current Web pages. Annotations are meta-data useful for machines to understand the contents of a Web page. Information in semantic Web is understandable by machines as well as human beings [GMM03].

Previous research shows that automatic integration of different search engines into a meta-search is not an easy task. Global attributes and value selection for unified search interface is one of the main problems in automatic integration of schemas extracted from different search engines. Meta-search engines also lack the capability of understanding the terms semantically.

In this dissertation, the aim is to address and solve the problems in job search domain as mentioned above and we address:

- The problems regarding automatic integration of data, structures and processes in human resource domain.
- The automatic integration of the job search interfaces to develop a meta-search engine.
- To solve the problems in the integration by new technologies i.e. Semantic Web.
• The transfer of results from different job search engines into a common structure by using modern Internet technologies.

• The use of HR-XML in the building of HR-XML schemes for meta-search interface.

• The integration of classification schemes for recruitment i.e. Standard Occupational Classification (SOC) System and skills from International Co-operation Europe, Ltd for the data values of meta-search interface.

• The development of design patterns for meta-search engine and its components.

One of the major steps in accessing heterogeneous and distributed data via a meta-search engine is the task of schema and data matching and integration. In this dissertation, we discuss the meta-search engine for jobs, with focus on heterogeneous schema/data matching, mapping and integration.

Much research has focused on developing techniques for schema matching and mapping as it is required in many areas e.g. heterogeneous database integration, e-commerce, data warehousing, semantic query processing, B2B applications, P2P databases, agent communication, Web Service integration. The schema matching process identifies correspondences between elements from different schemas. The schema mapping process defines these correspondences i.e. provides view definitions that link the two schemas [RB01]. Schema matching and mapping may generally be undertaken manually, semi-automatically or automatically. Manual or semi-automatic schema matching and mapping is tedious, complex and time consuming task because of open and heterogeneous nature of Web data. Automatic schema matching and integration is required to reduce manual efforts but require more researcher’s efforts to produce high quality techniques and results. The volumes and heterogeneity of Web data, in particular, mandate the development of automatic schema matching and mapping techniques.

Different types of heterogeneity may arise when schemas are matched e.g. syntactic, semantic and structural. a) When in different information systems, different languages are used to represent the elements then it is called syntactic heterogeneity. b) When in different information systems, same meaning of information is represented by different content then it is called semantic heterogeneity. c) When different information systems store their data in different structures then it is called structure heterogeneity.

One of the important challenges in automatic schema matching is to resolve semantic heterogeneity. Different types of semantic conflicts may arise e.g. confounding, scaling, naming. a) When information item seem to have a same meaning but different reality then it is called confounding conflicts. b) When different information systems use different reference systems to measure a value then it is called scaling conflict like currency differences. c) When different information systems use different naming schemes to represent an information item, then it is called naming conflict like homonyms and synonyms [WVV+01].

Another important challenge in automatic schema mapping is mapping cardinalities. There may be different mapping cardinalities between elements from different schemas, 1:1, 1:n, n:1 or n:m [EXD04]. One of the important challenges in automatic schema matching is to resolve such semantic heterogeneities.

Example 1: In the job search domain, one search engine schema may use “career type(s)” (see Figure 1.2 a) and another may use “categories” (see Figure 1.2 b) to
represent the same category of information. This is an example of a 1:1 mapping. Other search engines may use “type of job”, “job category”, “employment type” etc to represent the same category of information.

Example 2: One schema may use “salary” and another combination of “minimum salary” and “maximum salary” to represent salary. This is an example of a 1:n mapping.

Schema matching and integration is a challenging task and our aim is to have automatic schema matching and integration and resolving the semantic conflicts for meta-search engine creation.

The ontology integration, data integration and information integration research communities are addressing similar types of problems in matching and integrating heterogeneous schemas and ontologies. Our research starts from the premise that the techniques developed by these communities are of relevance to schema/data matching in meta-search engines, and that a combination of approaches is required c.f. [MBR01][Noy04]. Ontology integration researchers, database and information integration researchers are facing same type of problems. [MBR01] says: “we believe that the problem of schema matching is so hard, and the useful approaches so diverse, that only by combination of many approaches can we hope to produce truly robust functionality”. Database and information community call the matching process as schema matching and ontology community call it as ontology alignment. It is stated that “database, information integration and ontology community can certainly share and reuse the techniques they have developed, in their respective domain...... We believe that such cross fertilization will improve semantic-integration solutions in both fields” [Noy04].

In our setting, we are concerned with automatic schema and data integration of information arising from different Web portals. Consider, for example, the three search interfaces shown in Fig. 1.2 a, b and c. There is semantic heterogeneity both at the schema level and the data level. At the schema level, we see that three different concepts, “career type”, “categories” and “select a category”, are used to represent the same category of information. At the data level, we see that canjobs.com uses
“Administrative support”, careerbuilder.com uses “Admin – Clerical”, and jobs.net uses “Admin & Clerical” to represent the same item of information. For business-related jobs, careerbuilder.com uses “Business Development” and “Business Opportunity” while jobs.net uses “Business Development” and “General Business”.

Our focus in this dissertation is on automatic schema matching and integration techniques aiming to resolve semantic conflicts between different search engines, in order to support the construction of meta-search engines. Semantic heterogeneity both at the schema and the data level needs to be resolved. We need to discover meanings encoded in schemas from multiple search engines and create an integrated schema. The matching process must handle mappings of different cardinalities (1:1, 1:n, n:1, m:n). Moreover, appropriate integrated terms both at the schema and the data level must be selected for the meta-search interface. The mappings between the source and integrated schemas need to be used by the meta-search engine for query processing. Search engines return a list of Web pages and seekers have to spend a lot of time to navigate through these Web pages for the required contents. We need to extract structured results from the Web pages for the seeker. Semantic conflicts arising during result extraction from multiple search engines need to be resolved too. We aim to develop a configurable approach to meta-search engine construction that aims to meet these requirements.

Problems faced by job seekers and recruiters can be solved with the help of meta-search engine and semantic Web technologies. Use of semantic Web technologies meta-search construction in human resource will provide convenience to the job seeker for job search and can save precious time. Job ontology can help the search engine provider to construct a better job search engine’s interface. We believe that introduction of a new hybrid approach for schema and data integration will solve the problems regarding schema and data integration. Overall, this can also be helpful to decrease the unemployment rate of a country.

1.4 Overview of Dissertation

The rest of the dissertation is organized as follows:

Chapter 2, “Literature Review” contains the state of the art in job search portals. We explain the Internet technologies involved in the meta-search development. Furthermore, we discuss state of the art in the meta-search field from different groups. The existing solutions for query interface development and ontology based recruitment process are also discussed. This chapter also reviews related work in the schema, ontology matching/mapping and Web data Integration systems. Some classification standards, HR-XML and existing design patterns are also discussed in this chapter.

Chapter 3, “System Requirements”, contains the requirements for the meta-search engine from meta-search provider’s and job seeker’s point of views.

Chapter 4, “System Design”, contains the proposed system design to solve the problems discussed in chapter 3. Design patterns for general meta-search engine and its components are presented. Our overall architecture for meta-search engine in human resource domain is also given. This chapter also contains a detailed explanation about the use of modern technologies i.e. Semantic Web for the development of meta-search engine. It also includes the main components of meta-search engine i.e.
interface extractor, XML Schema generator, query interface generator, query dispatcher, result collector, result merger and ranker. This chapter also introduces a “hybrid approach” that consists of different techniques i.e. element level, structure level and ontology based. It is also explained how these techniques are helpful in generating a meta-search query interface. A human resource ontology designed to meet the requirements of meta-search engine is also discussed in detail here.

Chapter 5, “Implementation” discusses the technologies used for the prototype implementation. This chapter also contains the technical issues for the implementation of our configurable meta-search engine in human resource domain.

Chapter 6, “Evaluation” discusses the achievements by our configurable meta-search engine. This chapter contains short case studies helpful in the understanding of interface generation and query processing for meta-search engine. This chapter also explains the results of experiments performed for evaluation of different components of meta-search engine and some experimental results from the case study.

Chapter 7, “Conclusion and Future work” summarizes our contribution and gives direction for the future work.
CHAPTER 2
LITERATURE REVIEW

In the following chapter we discuss the related work in the field of e-recruitment, meta-search, schema matching and integration approaches, Internet technologies, existing classifications and standards for occupations that can be helpful in the development of job meta-search engines.

Section 2.1 describes the Internet recruitment, parts of the Internet recruitment and the existing application for e-recruitment. Section 2.2 is about Internet technologies including XML, XML Schema, HTML, HTML Forms, XForms and search interface forms. Existing meta-search projects including WISE, Lixto, Snorri, MetaQuerier, wrapper generation, result merging and ranking techniques and existing design patterns for domain specific search engines are described in section 2.3. Existing schema matching and integration approaches are described in section 2.4. Semantic Web, ontologies, kinds and parts of ontologies, Semantic Web languages including RDF, RDFS, OWL, SPARQL and impact of Semantic Web in e-recruitment are explained in section 2.5 and 2.6. Finally, section 2.7 contains HR-XML, existing classifications and standards for occupations and competencies.

2.1 Internet Recruitment

One of the most important steps in the recruitment process is to determine the method of recruitment. In the modern employment age there is a trend of Internet recruiting. Internet recruitment means that job is posted on an Internet job sites i.e. http://www.careerbuilder.com. Internet recruiting has many benefits as compared to other traditional methods i.e. internal recruitment, employee referrals, print advertisements, internship placements, executive search firms and unsolicited resumes. Internet recruitment is cost effective, fast and job postings are available 24 hours a day to potential candidates. Moreover, number of unqualified candidates can be minimized by directing people to more information on the organization’s Web site by Internet recruitment [BHM+05] [HLT02] [HRC07].

2.1.1 Parts of Internet Recruitment

Employers and job seekers are involved in the recruitment process. Many Internet recruitment sites serve both the employer who posts and the job seeker who monitors the information on the Web site.

2.1.1.1 Recruiter

Recruiter plays an important role in the recruitment process. He develops selection criteria for the position, determines the recruitment process, timeline, method of recruitment and also initiates the recruitment process. Recruiter posts job offers and gives the brief description of his/her organization, its mission or purpose, duties, responsibilities, qualifications, the supervising authority, salary, benefits attached to the position, salary range, the application deadline, start date, request for references, the format in which candidate would like to receive the information, contact name and the address. He is also responsible to check if resume of job seeker matches the job description or not and takes decision for the final selection of the employee. In this dissertation recruiter and employer will be used interchangeably.
2.1.1.2 Job Seeker

Job seeker is always in the search of better jobs. For the few job search portals he can directly search for the job but for the others he has to register and create CVs/resumes including contact information, education, skills, past professional experiences and preferences for the jobs [TE04] [HRC07].

2.1.2 Existing application for Internet Recruitment: JobOlize

JobOlize is an e-business application that supports job seekers and recruiters. [BPP+08] focused on improving the quality of job extraction process by the combination of existing NLP (Natural Language Processing) techniques with a new form of context driven extraction.

JobOlize deals with layout, structure and content information. JobOlize prototype consists of two core components i) knowledge base with e-recruitment domain ontology and a extraction rule base and ii) pipe line of extraction components.

E-recruitment domain ontology with 15 core concepts of job offers has been developed. Extraction rule base contains 50 extraction rules formulated using the regular-expression-based language JAPE (Java Annotation Pattern Engine). Pipe line consists of different extraction components i.e. initial annotation, page segmentation, block identification and relevance alignment. These components work on the basis of GATE (General Architecture for Text Engineering) infrastructure.

Initial annotation component is responsible for initial annotation of the tokens of the input Web page with concepts from domain ontology. Page segmentation component use Longest-Common-Subsequent (LCS) algorithm to identify the content part of the Web page from top, content and bottom part. Block identification component divide the content part into blocks. Further blocks are categorized into requirements, responsibilities and contact details etc. Relevance assignment component assigns pre-defined relevance values to the initial annotations depending on the block category. Finally, results are available to the user by rich client interface for final assessment.

JobOlize extraction system does not use the DOM tree of Web page but it works directly with the original Web page. JobOlize utilizes different type of context i.e. content, structure and presentation information to produce the resulting annotations and calculates corresponding relevance values. [BPP+08] claims the improvement in the quality of extracted information. Evaluation of JobOlize is based on the test set of 32 randomly chosen online IT job offers from 16 different Austrian recruitment Web sites. Common metrics i.e. precision, recall and F-measure are used for evaluating the quality of information extraction. [BPP+08] claims 64% increase in precision and 38% increase in F-measure regarding operation area. Regarding IT-Skills the precision and F-measure increase by 30% and 20% respectively.

2.1.3 Problems with Internet Recruitment

Postings or searching jobs by the job search portals is recommended but there are some problems. In typical job search engines, job seeker gives very brief information for the job search i.e. region, job category, and keyword. Job search engines treat the keyword
as a string, return the results by string matching and counting the frequency within the document. Due to this, job portals deliver a large number of job offers irrespective of relevancy. Job seeker has to search again through a large number of job offers. For a job seeker it is not possible to keep the overall view of the large number of job portals. To overcome all these problems we have decided to develop a meta-search engine in human resource domain that can combine all benefits of specialized search engine, meta-search engine and semantic search engine.

2.2 Internet Technologies

We face many challenges in the development of configurable meta-search engine in human resource domain. In this section we’ll discuss the Internet technologies that can be used in the development of meta-search engine.

2.2.1 XML Technologies

XML technologies include XML, XML Schema, XPath, XSLT and XQuery. Following is the detail of XML technologies useful in the development of meta-search engine:

2.2.1.1 XML

Extensible Markup Language (XML) is a W3C standard used to describe data. “XML provides a generic syntax used to mark up data with simple, human-readable tags” [HM04]. It does not provide predefined tags and elements but allows developers and writers to define their own tags and the structure of the document. We can create as many tags as we need. Extensible in XML means that the language can be extended and adopted to meet the needs of different types. The Markup in XML describes the structure of the document. [HM04] says: “The markup permitted in a particular XML application can be documented in a schema. Particular document instances can be compared to the schema”. XML documents consist of XML declaration, root element, XML elements, attributes, tags, namespace and entity reference. XML solves the problems due to long term data formats and translation due to cross platforms. XML has been designed to store, carry, and exchange data between incompatible systems. It is extremely flexible format for data.

XML parsers are required to understand XML documents. XML parser can identify elements, attributes and other piece in an XML document. XML use XML Schema languages to describe the schema. Most widely used schema languages are Document Type Definition (DTD) and W3C XML Schema language (see section 2.2.1.2 for W3C XML Schema). XML parsers can also compare the XML document to its schema to check if the document satisfies the specified constrains in the schema. An XML document should be well formed and must be a valid document. By well formed we mean that it obeys the syntax of XML and by valid we mean that it conforms to some semantic rules i.e. valid element sequence and nesting, correct type of attribute values. These rules are user defined or included as DTD or XML Schema [HM04].

2.2.1.2 W3C XML Schema

W3C XML Schema is an XML document that contains powerful and expressive validation method on XML documents. XML Schema is a XML-based alternative to
DTDs but richer and more powerful than DTDs. XML Schema defines elements, child elements, order of child elements, number of child elements, data type of elements and attributes in a document. It is also possible in XML Schema to define default and fixed values for elements. XML Schema also provides namespace support to prevent misunderstandings. In contrast to DTD, W3C XML Schema can define complex restrictions on elements and attributes. [HM04] says: “In addition to a wide range of built-in simple types (such as string, integer, decimal and dateTime), the schema language provides a framework for declaring new data types, deriving new types from the old types, and reusing types from other schemas. Besides, simple data types, schemas can place more restrictions on the number and sequence of child elements that can appear in a given location”

For data communication it is most important that sender and receiver can understand the data in a common format. XML Schemas provide secure data communication and mutual understanding of contents. For example date “02-11-2002” can be interpreted as 2nd November, 2002 or 11th February, 2002. XML Schema element with a date data type <date type="date">2002-02-11</date> ensures a mutual understanding of the content as XML data type "date" requires the format "YYYY-MM-DD".

XML declaration defines an XML version and the character encoding used in a document, at the beginning of XML Schema. Listing 2.1, line 1 shows XML declaration and it means that document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set. XML Schema must also contain a root element. The <schema> element (see listing 2.1, line 3) is the root element of XML Schema. In the XML Schema in listing 2.1, standard namespace (xs) is used and the URI associated with this namespace has the standard value of http://www.w3.org/2001/XMLSchema. This declaration tells the schema validator that all the elements used in this XML document are declared in “http://www.w3.org/2001/XMLSchema” namespace. Remaining parts of XML Schema define the elements of a file. The <sequence> indicator specifies that the child elements must appear in a specific order.

An XML Schema may contain simple and complex type elements.

- **Simple Elements**

A simple element is an XML element that contains only text. It cannot contain any other element or attribute. Text can be one of the types included in an XML Schema definition (Boolean, string, date etc) or it can be of custom type i.e. defined by a user. Listing 2.1, line 7 and 9-18 are examples of XML simple elements. Simple elements may have a default value specified by using default attribute or fixed value specified by using a fixed attribute. Listing 2.1, line 9 represents an example of simple element with default value and line 22 represents an example of simple element with fixed value. XML Schema has many built-in data types i.e. xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time. In order to limit the contents, restrictions (facets) can be added to a data type.

- **Complex Elements**

A complex element is an XML element that contains other elements and/or attributes. There are four kinds of complex elements i.e. empty elements, elements that contain
only other elements, elements that contain only text, elements that contain both other elements and text. Complex elements may contain attributes as well. Listing 2.1, line 20-26 shows an example of complex type element that contains other elements. Complex elements can have a “type” attribute that refers to the name of complex type to use.

Listing 2.1: A Simple XML Schema

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- XML SCHEMA -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="RootJob">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="enter_keywords" type="xs:string"/>
        <xs:element name="enter_a_city" type="xs:string"/>
        <xs:element name="select_a_category" default="all job categories">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="all job categories"/>
              <xs:enumeration value="accounting"/>
              <xs:enumeration value="banking"/>
              <xs:enumeration value="information technology"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="employment_type" type="xs:complexType">
          <xs:sequence>
            <xs:element name="full-time" type="xs:string" fixed="full-time" minOccurs="0"/>
            <xs:element name="contractor" type="xs:string" fixed="contractor" minOccurs="0"/>
            <xs:element name="part-time" type="xs:string" fixed="part-time" minOccurs="0"/>
          </xs:sequence>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

After specifying all elements in the schema, there must be `</xs:schema>` tag to represent an end of schema [Con04a] [HM04].

**2.2.2 HTML**

HyperText Markup Language (HTML) developed by Tim Berners-Lee, is used to publish information on the World Wide Web. HTML is a universally understandable language by all the computers. It is used to publish online documents, retrieve online information and design forms. HTML file contains a markup tags surrounded by angle brackets to tell the Web browser, how to display the page. HTML documents may contain headings, hyperlinks, texts, tables, lists, photos, spread-sheets, video clips, sound clips and other applications.
HTML 4.0 is an extension of HTML and separates the presentation layout from the document contents. It provides a way to move out all the formatting from HTML to a separate style sheet. HTML 4.0 also provides mechanisms for form enhancement, richer tables, style sheets, scripting frames, embedding object and many more features. HTML 4 also brings accessibility improvements to forms. Errors in HTML 4.0 are corrected in revision HTML 4.01.

Hypertext Transfer Protocol (HTTP) is used to transmit HTML documents from a Web server to a Web browser. It is required to send a file format to the Web browser during transmission because Web browser must know the file format of the document to handle it properly [Con99a].

2.2.3 Forms

Forms are used to get input from the user. In case of job search engines, they are used to get input i.e. job category, country, and job type from the job seeker. Below HTML forms, XForms and search interface forms are explained:

2.2.3.1 HTML Forms

HTML forms are used for searching information, ordering products, making reservations and making transactions with remote services etc on the Web. Forms allow users to select or enter different kinds of input by using form elements (also called control elements) i.e. text field, text area fields, radio buttons, checkboxes, drop down menus etc. A form is defined with the <form> tag with “action” and “method” attribute. The “action” attribute specifies a form processing agent and “method” attribute specifies the HTTP method to be used to submit the form data sets i.e. get or post (see listing 2.2, line 2). The <input> tag with the type attribute is used to represent the control element in the form tag. The most commonly used control elements are explained below:

- Text input is used to get input from the user. Listing 2.2, line 5 shows an example of single line text input control.
- Radio buttons are used when user wants to select one and only one option from the multiple choices. Listing 2.2, lines 7-8 show an example of check boxes with name, value and checked attribute.
- The select element is used to create a menu. ‘Option’ element with select element is used to offer choices and user can select from the list of offered choices. Listing 2.2, lines 11-15 is an example of select element with ‘option’ element with choices of countries.
- Checkboxes are used when user wants to select multiple options from multiple choices. Listing 2.2, lines 18-19 show an example of check boxes with name, value and checked attribute.
- HTML forms can have three types of buttons i.e. a submit button, reset button and a push button. Submit button is used to submit a form. Line 23 of listing 2.2 represents a submit button created by using an ‘input’ element. When submit button is clicked, the contents of the form are submitted to another file i.e. Job_form_action.asp file mentioned in the form’s action attribute (see listing 2.2, line 2).
- The input element with the hidden type is not visible on the form but it is used to submit some technical values with the form. Mostly it is used to store information
between client/server exchanges. Listing 2.2, line 21 shows how hidden control can be used in an HTML form.

Listing 2.2 : A Simple HTML Form

```html
<!-- HTML Form -->
<form name="input" action="Job_form_action.asp" method="get">
  <!-- Text Input -->
  Enter Keywords:
  <input type="text" name="Kwd" size="50"/>
  <!-- Radio Buttons -->
  <input type="radio" name="SelectType" value="any"/>Any of these
  <input type="radio" name="SelectType" value="all" checked="checked"/>All of these
  <!-- Select Element -->
  Select Country:
  <select name="Country">
    <option value="austria">Austria</option>
    <option value="pakistan">Pakistan</option>
    <option value="usa">USA</option>
  </select>
  <!-- Check Boxes -->
  Job Type:
  <input type="checkbox" name="JobType" value="full" checked="checked"/> Full-Time
  <input type="checkbox" name="JobType" value="temp"/> Temporary
  <!-- Hidden Control -->
  <input type="hidden" name="DispalyJobs" value="20"/>
  <!-- Submit Button -->
  <input type="submit" value="Submit">
</form>
```

In the form element, the “method” attribute specifies the method of transferring the form data to the Web server. Form submission method can be either “get” or “post”. In get method the form data is encoded by the browser into the URL. This method should be used when the form is idempotent. By idempotent, we mean that form processing is only for retrieving data and processing causes no changes anywhere except on the user’s screen. If such a form is resubmitted, it might get different data (if the data had been changed meanwhile), but the submission would not cause any update of data or other events. For example, if URL of form in listing 2.2 is http://www.myjobs.com/ and it is submitted then the form data will be encoded into the URL and will become http://www.myjobs.com/Job_form_action.asp?Kwd=java&SelectType=all&Country=austria&JobType=full&DispalyJobs=20.

In post method the form data appears within a body of the form instead of encoding with URL [Con99b].

2.2.3.2 XForms

XForms are platform and device independent forms that separate data and logic from presentation. XForms are made up of separate sections i.e. model and user interface. XForm model describes the data and logic. XForm’s user interface defines the input fields and how these fields should be displayed. By using XForms, same form can be used to be displayed on the Web browsers, handheld devices, and other applications.
XForms define, store and transport data in XML format. XForms provide richer user interface to meet needs of businesses, consumers, and applications. XForms also provide features for form validation.

XForms may contain various device independent controls i.e. input, label, secret, text area, submit, trigger, output, select and upload. The most commonly used XForm controls are input control for text input and submit control for submitting data. XForm leaves it up to the browser to decide how to display the controls. XForms model supports XML Schema data types and also has some predefined functions. It is possible that XForm properties define data restrictions, types and behaviour. XForms can be used to improve authoring, reuse, internationalization, accessibility, usability by separating data and presentation [W3Sa].

2.2.3.3 Search Interface Forms

Users can get required information from different search engines by the search interface form. Many researchers are concentrating on the advanced query interface design. Ill-designed forms can stop the users from using that search engine therefore interface of search engines should be well designed and easier for the users. Six different types of search engine’s interfaces by [PMH+04] are as follows:

If search engine interface supports a search of only one category of product, it is called devoted type i.e. jobs.net.

- If search engine interface supports a search of multiple categories of products by separate child search interface, it is known as divided type i.e. amazon.com.
- If multiple search engines co-exist on the single Web page, it is called co-existing type i.e. airfare.com.
- If single search form is used to search multiple categories of products, it is known merged type. In this type of search interface form, first user has to select the category and then press common search button to start the search. The search fields of different products co-exist in the search form i.e. alldirect.yatego.com.
- If there are multiple child pages for different type of products but each child page share the same search form with one text input field, it is called shared type interface i.e. halfpricebooks.com.
- If multiple pages are used and multiple interactions with the user are required to send a complete search query, it is known as multi-page type search interface. Multi-page interface requires filling and submission of sequence of pages.

According to the analysis of forms in [PMH+04], 82.96% of the search engines are of devoted type.

2.3 Existing Meta-search Solutions

Meta-search engine construction involves interface extraction, interface integration, schema matching, unified query interface generation, query transmission, wrapper generation and result merging and ranking.
Following section contains the existing research work in the field of meta-search:

### 2.3.1 WISE-Web Interfaces of Search Engines

Web Database Meta-search Engine project has been developing technologies for providing integrated access to the Web databases (http://www.cs.binghamton.edu/~meng/DMSE.html). In the development of meta-search engine, important and popular phases are schema extraction, matching and integration. These phases have received much recent attention of researchers. [HMY+03] [HMY+04a] [HMY+05a] [HMY+04b] [HMY+05b] have played a vital role in the development of meta-search engine and have developed WISE: iExtractor for interface extraction and WISE-Integrator for automatic integration of schema, attribute values, format and layout.

WISE-iExtractor and WISE-Integrator deal with e-commerce based search engines. As different search interfaces in the same domain can contain different number of attributes, different names for representing the same type of elements and organize the attributes in different ways so there are problems in the integration of different search interfaces. Search interfaces represent the schemas in different ways. WISE-iExtractor and WISE-Integrator explore special meta-information from the HTML form-based Web search interfaces and use this meta-information to identify the matching attributes from different search interfaces for integration.

[HMY+04b] uses traditional dictionaries along with multiple matching techniques to find semantic similarity between schema elements and values. WISE-Integrator completes the integration process of e-commerce search engines (ESE) in five steps i) Web search interface extraction ii) Interface schema matching iii) Global attribute generation iv) Unified search interface construction v) Unified interface maintenance. Below are the details about the working of WISE-iExtractor and WISE-integrator:

#### 2.3.1.1 Web Search Interface Extraction

HTML Web search interface consists of control elements i.e. text box, checkbox, radio button, select list and label (discussed in section 2.2.3.1). There is no any definition of schema and meta-data for such type of Web search interfaces. First step for the automatic integration is identification and extraction of schema from the HTML search interface. [HMY+04a] [HMY+05a] explain that each interface consists of an ordered list of attributes and each attribute has an associated element. They have worked to extract attributes and meta-information of attributes from search interfaces for the construction of schema model as below:

##### 2.3.1.1.1 Attribute Extraction

During attribute extraction, logic attributes are identified and then related elements and labels are grouped together. Forms are extracted from the group of forms. Further HTML control elements i.e. text fields, text areas, checkboxes, radio buttons, drop down boxes and their labels are extracted. To determine physical location of elements and labels, line separator tags are also extracted. An interface expression (IEXP) representing a search interface is constructed. All the labels and elements that semantically correspond to the same attribute are grouped together by using LEX
(layout-expression-based extraction) technique and an appropriate attribute label/name for each group is selected. Some special features of the search form are used to identify an appropriate label for elements i.e. ending colons, textual similarity of element name and the text, distance of element and text [HMY+04a] [HMY+05a].

2.3.1.1.2 Attribute Analysis

During “Attribute Analysis” phase, information regarding elements i.e. relationship type, domain type, default value, value type and unit is collected. If an attribute has multiple domain elements then a relationship type and semantics of domain have to be identified. Otherwise, if there are no multiple domain elements then relationship type is none. Relationship can be of “Group type”, “Range type” or “Part type”. Meta information is information about attributes i.e. domain type, default value, value type and unit. This information is also used for attributes analysis. Domain type can be range, infinite, Boolean or finite. Value type i.e. date, time, currency etc is determined by the analysis of attribute name. A schema model representing a search interface is constructed by using meta-information about attributes.

2.3.1.2 Interface Schema Matching

Interface schema matching involves identification of i) semantic relationship between attributes and ii) matching attributes. Three types of semantic relationships i.e. synonymy, hypernymy and meronymy between attribute names or element values are identified by using WordNet. Schema matching process is classified into positive match based clustering and predictive match based clustering [HMY+03] [HMY+04b] [HMY+05b].

2.3.1.2.1 Positive Match Based Clustering

Positive match based clustering group the attributes into clusters based on positive matches between attributes. Positive match based clustering considers that values from the two attributes are matched if there is an exact match, synonymy, hypernymy and meronymy or a value based match (exact match, approximate string based match, synonymy, hypernymy). In positive match based clustering, first threshold value is used, next preliminary attribute matching is done and then attributes are grouped into clusters. Furthermore, representative attribute name (RAN) of cluster is identified. RAN of cluster is a candidate of the global attribute name. RAN of cluster is selected on the basis of following i.e. i) generality or majority rule ii) root of hierarchies i.e. hypernymy hierarchies are built on the basis of attribute name in the cluster.

2.3.1.2.2 Predictive Match Based Clustering

Predictive match based clustering states that values from two attributes are matched, if there is an approximate name match, cosine similarity of names, domain type match, value type match, unit match, default value mach and value pattern match. For example, attributes “Publication Year” and “Release Date” cannot be matched by the positive match based clustering because they do not have the same name or matching values. So in this case, predictive match based clustering is used and meta-information may help in matching.
Initially there exists no cluster. When first interface is considered, each attribute $A_i$ in the local interface is considered. Clusters are constructed incrementally and mappings are entered to an attribute-cluster thesaurus. The approach first lookup, if an attribute name of $A_i$ has already mapped to an existing attribute-cluster thesaurus or not. If attribute $A_i$ in the local interface is already mapped to the cluster thesaurus then $A_i$ is placed in the existing cluster. Otherwise, it is checked that RAN ($A_i$) is mapped to an existing cluster. If such a cluster exists then $A_i$ is mapped to the cluster and mapping is added to the attribute-cluster thesaurus. If two lookups fail, then weight between existing cluster $C_k$ and an attribute $A_i$ is calculated to determine that which cluster should include $A_i$. $A_i$ is added to the highest weighted cluster.

Weight between two attributes $A_i$ and $A_j$ is the sum of weights based on seven matching metrics by using $W(A_i,A_j)= W_{am} + W_{vss} + W_{cd} + W_{vtm} + W_{cu} + W_{dv} + W_{vp}$ where $W_{am}$ is approximate string match and substring match, $W_{vss}$ is cosine similarity match, $W_{cd}$ is domain type match, $W_{vtm}$ is value type match, $W_{cu}$ is unit match, $W_{dv}$ is default value match and $W_{vp}$ is value pattern match. As a result of interface schema matching phase, matching attributes from different search interfaces are semantically grouped into clusters.

### 2.3.1.3 Global Attribute Generation

During this phase global attributes for the unified interface are generated by determining i) global attribute name, ii) global attribute domain type, and iii) global value type. Attribute from RAN or any attribute name from the result of predictive match based clustering step is selected for global attribute name. Majority rule is applied for the final selection and attribute mapping table is maintained. Matching attributes may have different domains but it is required to have a single domain for the unified interface.

[HMY+04b] uses following rules to determine the global attribute domain type i) (finite + finite) $\rightarrow$ finite, ii) (range + any type) $\rightarrow$ range iii) (finite + infinite) or (finite + hybrid) + (finite + hybrid) $\rightarrow$ hybrid and iv) (finite + boolean) $\rightarrow$ finite. The unified interface must contain the semantically unique or compatible values from the local interfaces. The values from matching local attributes must be merged to form global values for the global attributes. [HMY+04b] considers the merging of alphabetic domain separately then numeric domains.

If domain type of local attributes is finite and has alphabetic values then it is required to merge these alphabetic domains as follows:

- Matching techniques (exact match, synonymy match, approximate string match, hypernymy match) are applied on the values of attributes in the same cluster during positive match based clustering phase and semantic relationship is identified.
- Relationship between values is used to organize all of them into categories based on approximate string match, cosine similarity match, synonymy match and hypernymy match.
- In case of synonymy relationship, majority rule is applied and most popular among the synonyms or similar are used as the global values.
- In case of hypernymy relationship, both generic and specific concepts from the hierarchy of values are used for the integrated interface.
For numeric domains, WISE-Integrator uses unit relationship dictionary to convert all the units to one compatible unit. Range and non-range numeric domains are treated differently.

### 2.3.1.4 Unified Search Interface Construction

On the unified interface, text boxes are used to represent the infinite global attribute domain type and selection lists are used to represent the finite global attribute domain. Next, attribute layout is determined. From each local interface, layout position of attribute is aggregated and final layout position of attributes at unified interface is determined on the basis of layout aggregation.

### 2.3.1.5 Unified Interface Maintenance

Unified interface maintenance is required because sometimes new local interfaces are added or some existing interfaces are removed from the unified interface. If new interfaces are added, positive match based clustering step incrementally cluster the new attributes into existing clusters. RAN is updated on the basis of current and previous statistical and semantic knowledge. Predictive match based clustering is performed again and mapping is stored. Finally, some new global attributes need to be added to the previous interface. If existing local interface is removed then attribute names, their corresponding values from the clusters, related mappings are removed and new domain types are determined again, if required.

### 2.3.2 Lixto Meta-search

The Lixto Suite (http://www.lixto.com) provides a platform to access the Web sites and extract information from them. The Lixto suite provides a Lixto Meta-search product and develops a hardwired meta-search solution. Lixto suite consists of a visual wrapper and a transformation server. Lixto’s visual wrapper is used for creating a wrapper that extracts the relevant information from HTML documents and translates it into XML, which can be queried and further processed. The Lixto transformation server provides data flow processes like collecting, transforming, concatenating, sending and storing of XML documents [GKB+04] [Jau06]. Lixto technology can be used to define fully automated extraction process from different Web sites, without having programming expertise. Extracted information can be collected and stored in a database. Lixto suite also provides the facility for product comparison and report generation.

Lixto Meta-search uses Lixto visual wrapper for the extraction of relevant information and then integration of different Web sites into a meta-search application. Lixto meta-search product provides automatic navigation, data extraction and data structure mapping. Lixto meta-search uses XML for communication between meta-search engine and the front end of the application. Lixto has implemented meta-search applications in the flight search and hotel search domains.

Meta-search has to handle multiple search requests and it is required to handle these multiple requests in parallel for fast processing. [Jau06] has studied already existing meta-search solutions including Lixto meta-search and stated that Lixto meta-search has some limitations i) synchronous provision of results ii) complicated creation and deployment of scenarios iii) slow URI builder work in sequential mode.
2.3.3 Snorri Meta-search

[Jau06] introduces a special-purpose meta-search named “Snorri” which extends the Lixto meta-search by eliminating limitations such as synchronous provision of results. Snorri also introduces caching, scalability and load balancing. [Jau06] designs the mapping and execution framework for Snorri meta-search engine. Mapping framework design reflects the form mapping theory in the meta-search with the important features i.e. semantic mapping, mapping list support, data and currency conversions. Execution framework design provides scalable, multi-tier and fail-over solutions.

[Jau06] designs and implements Snorri meta-search engine with the following features: i) easier and simpler form mapping framework design. ii) monitoring of whole process is possible. iii) provides scalable solutions to meet the needs of customers. iv) load balancing solutions are available. v) supports source pruning i.e. search that do not provide any result for a similar query will not be searched and thus saves times. vi) supports intelligent pre-loading i.e. advance loading of most requested sources to increase the speed. vii) intermediate results are accessible before the end of the whole search process. viii) caching functionality is available to improve the responsiveness of the system. ix) navigation of multiple search forms with the help of Lixto navigation tool is available. x) provides an option of parallel mode URIbuilder. xi) transformation of results in parallel. xii) handles multiple requests, increased processor load and memory consumption.

2.3.4 MetaQuerier

MetaQuerier project (http://metaquerier.cs.uiuc.edu/) explores and integrates the query databases that are not visible to the traditional crawlers. [CHZ05] claims that large scale integration by MetaQuerier involves dynamic source discovery and on-the-fly query translation. Integrating Web sources of the same domain type is simpler as compared to heterogeneous Web sources. Main components of MetaQuerier are MetaExplorer and MetaIntegrator. MetaExplorer discovers sources on the deep Web and builds a search engine of Web databases. Moreover, MetaExplorer also designs models to represent discovered databases and develops wrappers to automatically extract schema details. MetaIntegrator, integrates the discovered sources into one. The main focus of MetaIntegrator is source selection, query mediation and schema integration. First step is the clustering of interfaces of same type of domain into a subject cluster and then creation of a unified interface for each cluster by discovering semantic matching between attributes.

[CHZ05] discovers that there may exist simple 1:1 matching or complex m:n matching during schema matching. For example, in a book domain subject and category are synonyms and they only need simple matching. In the air domain passengers= {adults, seniors, children, infants} and in book domain author = {first name, last name} need complex mappings. Semantic matching sub-system of MetaQuerier stores these matchings in the deep Web repository for the construction of unified interface and query translation purposes. [CHZ05] claims automatic extraction of schema models and then matching Web query interfaces. MetaQuerier applies a statistical/probabilistic approach for schema matching but do not use domain ontology for schema matching process. The authors claim that their system fully automates all tasks in streamline to output semantic matching [CHZ05] [HZC05] [HC03].
2.3.5 Wrapper Generation

Information extraction from different Web sites is often performed by using wrappers. “A wrapper can be seen as a procedure that is designed for extracting the content of a particular information source and delivering the content of interest in a self-describing representation” [Eik99]. Wrappers can be constructed manually, semi-automatically and automatically. Manual construction of wrapper requires high cost, maintenance and time. Semi-automatic wrapper generation involves programming by demonstration. For every site, it must be demonstrated that which field should be extracted. So automatic wrapper generation is required that can automatically construct wrappers. Inductive learning methods can be used to generate the extraction rules in the wrapper construction. Researchers from information extraction area have done considerable efforts for “wrapper induction”. Wrapper induction means generation of procedures that can extract required information from the text [Bre03].

[ZMW+05] introduces some techniques to automatically extract search result records (SRR) from dynamically generated HTML result pages and discards other irrelevant information. [ZMW+05] presents a tool called ViNTs (Visual Information aNd Tag Structure) that utilizes visual content features and HTML tag structure of HTML result pages for the automatic wrapper generation of any given search engine.

[ZMW+05] shows the working of wrapper generation process as follows: First dynamically generated result page is rendered. Next for each object a rendering box is created and content lines are extracted. Content line is a group of characters that visually form a horizontal line in the same section. In HTML page content line is a combination of link, text, link text, link-head, text-head, link-text-head, HR-line and break etc. Then x-coordinate of each rendering box is detected, blocks are created and shape of block is studied. Block similarity is checked on the basis of type distance, shape distance and position distance and candidate SRRs are detected. These candidate SRRs are used in the generation of final wrapper. Dynamically generated result page is transformed into a tag tree i.e. tree representation of tags and a tag path is constructed. As result pages are dynamically generated so corresponding group of SRRs have similar tag structure and the roots of all the groups of SRRs must be siblings. On the basis of tag structure and tag path similarity, initial wrapper is generated. Initial wrapper is further refined on the basis of rendering area weight, center distance weight, number of record weight and average number of character weight. Finally, wrappers generated from multiple sample result pages are integrated to build a single wrapper for search engine.

[ZMY06] extends the wrapper generation process by introducing new techniques for the extraction of search result records from the result pages. [ZMY06] proposes an algorithm “Multiple Section Extraction” to solve the problems in the identification of the dynamic sections, SRRs and differentiation between sections and records. Data extraction from the Web result pages involves section extraction, record extraction and data annotation. [ZMY06] defines that a typical result page consists of three sections i.e. static, semi-dynamic and dynamic. Static section remains same while semi-dynamic section is affected by different queries. Dynamic section depends on the query terms and consists of set of result records. SRRs in the dynamic section have same display format. [ZMY06] uses the static and semi dynamic sections to identify the boundaries of dynamic section and then proposes techniques to extract SRRs from identified dynamic section.
[BFG01] introduces some techniques for supervised wrapper generation and automated Web information extraction. [BFG01] implements these techniques in a system called Lixto as explained in section 2.3.2. Lixto provides a visual, interactive and convenient user interface to the user for the creation of semi-automatic wrapper program. The Lixto wrapper generator consists of modules i.e. navigator, extractor and visual developer.

[LGZ03] proposes an algorithm MDR (Mining Data Records in Web Pages) to mine data records in a Web page automatically. Authors claim that their data mining technique is able to mine both contiguous and non-contiguous data records.

2.3.6 Result Merging and Ranking

[LMS+05] investigates result merging algorithms for meta-search engines to merge results from different search engines into a single ranked list. [LMS+05] states that merging based on the titles and snippets of retrieved results can outperform that based on the full document. [LMS+05] presents five algorithms i.e. TopD, TopSRR, SRRSim, SRRRank and SRRSimMF for merging results into a single ranked list. Search result record (SRR) consists of URL, title and summary (snippet) of the extracted document. TopD algorithm uses top document while TopSRR algorithm uses top SRRs to compute search engine score. SRRSim algorithm computes similarities between SRRs and the query. SRRRank algorithm ranks SRRs using more features like location of the occurred query term or total number of occurrences of the query term in the title and snippet etc. SRRSimMF algorithm computes similarities between SRRs and query using more features.

2.3.7 Existing Design Patterns for Domain Specific Search Engines

In object-oriented systems there exist recurring patterns of classes and communicating objects. These patterns provide simple and elegant solutions to specific design problems and make object-oriented designs more flexible. In [GHJ+95], Gof (Gang of four) describe 23 of the most common patterns in detail. Gof’s design patterns can help the developers to craft their own specific applications and give them a common vocabulary to describe design concepts, rather than particular implementations [GHJ+95] [FF04]. To our knowledge, [ZQD+03] is the only published research that describes the framework for domain specific search engines from design patterns perspective. [ZQD+03] also presents design patterns for few components of domain specific search engines.

2.4 Schema and Ontology Matching/Mapping Approaches

In this section, we discuss related work in the area of schema/data matching and mapping, ontology matching and mapping and Web data integration systems that can be helpful in the development of meta-search engine.

Researchers from schema/data matching and mapping area have done considerable efforts [MBR01][RB01][ADM+05][Riz04][SE05][HG02][EXD04][ZP04]. [MBR01] proposes an algorithm Cupid that uses different set of approaches and utilizes name, data types, constraints, schema structure, linguistic matching, structural matching, context dependent matching and leaf structure for schema matching. Cupid also uses
thesaurus to find acronyms, short forms and synonyms. But in some applications, generic thesaurus i.e. WordNet is not enough and domain specific sense of a word is required.

[RB01] investigates seven prototype implementations i.e. SemInt, Learning Source Descriptions, Semantic Knowledge Articulation Tool, TranScm, DIKE, ARTEMIS and CUPID. [RB01] presents a taxonomy for existing schema matching approaches i.e. schema level and instance level, element level and structure level (including top down and bottom up approach), linguistic based and constraint based. Different factors i.e. match cardinality, re-use/auxiliary information, matcher’s combination, manual work/user input etc are also considered for the comparison purposes.

COMA++ (COmbining MAthch) is a schema and ontology matching tool to identify semantic correspondences between meta-data structures or models. COMA++ supports more than 15 matchers that use schema or auxiliary information to find a match between components of schema. COMA++ uses fragment based and a re-use oriented approach and is used as a platform for evaluating different match algorithms. COMA++ supports different type of models including XSD, XML Data Reduced (XDR), OWL, and relational schemas. It also uses taxonomy that acts as an intermediate ontology for schema or ontology matching [ADM+05].

[Riz04] presents an approach to automatically discover semantic relationship between schema elements without utilizing external knowledge i.e. user supplied training data, user defined concept hierarchies, synonym tables, online ontologies and dictionaries. Architecture presented by [Riz04] consists of element name module, data type module, numerical statistical module, non-numerical statistical module, instance module (that use Naive Bayes classifier), number of instance module, precision module, length module and existence module. [Riz04] utilizes several types of information and uses bidirectional comparison of the elements instances and meta-data to discover five types of semantic relationship i.e. incompatible, disjoint, intersection, subsumption and equivalent between elements.

[SE05] has classified elementary schema based matchers into “element level” and “structure level” matching techniques. [SE05] distinguishes between syntactic, semantic and external techniques at element and structure levels. Structure level techniques presented by [SE05] are based on graph-based techniques, taxonomy based techniques and model based techniques. Repository of structures is used to find the pair wise similarities with the help of stored schemas or ontologies and their fragments.

[HG02] presents an approach to integrate different schemas from different communities into a single global schema for federated database systems. As each community uses different formal ontologies therefore the concepts in these ontologies are merged by the means of similarity relations. Project like SHOE and Ontobroker also use ontologies to improve the searching abilities on the Web. [EXD04] introduces two matchers i.e. object-set and structure matchers to improve the matching process. Object-set and structure matcher both use instance-level and schema-level matchers too. Object-set matcher presented in [EXD04] is composed of three matchers i.e. name matcher, value characteristic matcher and data frame matcher. Structure-matcher introduced in [EXD04] differs from the structure matcher presented in [RB01]. Structure matchers presented in [EXD04] not only use structural similarity of two
schemas but also use domain ontology snippets to find mappings. Structure matcher that uses domain ontology snippet increases the mapping accuracy [EXD04].

[ZP04] introduces a framework AutoMed - “Automatic Generation of Mediator Tools for Heterogeneous Data Integration” for investigating virtual, materialised and hybrid data integration. AutoMed (http://www.doc.ic.ac.uk/automed/) guides the users and developers to translate and integrate data from various data sources. AutoMed is based on a BAV (both as view) data integration approach. Transformation pathways are defined between schemas for data transformation or integration, for example between a set of data source schemas and a virtual integrated schema [ZP04].

Different research communities use string distances for matching and clustering list of entity names. String distances can also be utilized while schema matching. [CRF03] compares string distance metrics for matching entity names. [CRF03] considers Edit-distance like functions (Levenshtein distance and Monga Elkan) and Token based distance functions (Cosine Similarity and Jaccard Similarity functions) for the comparison. [Cha06] investigates string metrics and developed an open source java library “SimMetrics”.

“Levenshtein distance (LD) is a measure of the similarity between two strings, source string (s) and the target string (t). The distance is the number of deletions, insertions, or substitutions required to transform s into t. The greater the Levenshtein distance, the more different the strings are [Gil07]”.

“Cosine similarity is a common vector based similarity measure. In Cosine similarity function input string is transformed into vector space so that the Euclidean cosine rule can be used to determine similarity [Cha06]”. Below is a formula to measure cosine similarity between q and r:

$$\cos(q, r) = \frac{\sum_y q(y)r(y)}{\sqrt{\sum_y q(y)^2 \sum_y r(y)^2}}$$

“Euclidean distance approach again works in vector space, however the similarity measure is not judged from the angle as in cosine rule but rather the direct Euclidean distance between the vector inputs [Cha06]”. Below is a formula to measure Euclidean distance between vectors q and r:

$$e_{\text{uc}}(q, r) = \left(\sum_y (q(y) - r(y))^2\right)^{1/2}$$

There also exist some stemming algorithms i.e. porter stemmer algorithm that convert word to their related form i.e. root, stem, or base. The stemming process is useful to solve the problems in search engines, natural language processing, and text processing [Por06].

[Lin07b] describes the schema matching process and ontology based schema matching process in detail. During ontology based schema matching process, one of the critical requirements is the availability of the ontologies that describe the data sources. During schema mapping, is-a, equivalence and disjointness relationships are considered. There can be two cases while ontology based schema matching process. In first case, a single ontology describes all the sources. In second case, multiple ontologies describe all the sources and these multiple ontologies are merged together to form a new single shared
ontology. [Lin07b] studies different ontology merging methods i.e. Chimaera, PROMPT, FCA-MERGE and stated that “Methods for matching in the field of ontology merging or ontology alignment are of similar principles to the methods for schema matching. That is, because ontologies and data schemas are closely related…. Methods of ontology alignment or ontology merging are performed, as methods for schema matching at different levels: instance, element and structure, and use syntactic and semantic approach”.

Research on a very large scale is in progress by the ontology community in the field of ontology matching and mapping [WVV+01][Lin07a][Noy04][XC06]. [WVV+01] has analyzed 25 approaches using ontologies as a solution to semantic heterogeneity problem and information integration. Single ontology approach, multiple ontology approach or hybrid ontology approach is used for the identification and association of semantically corresponding information concepts. Single ontology approach uses single global ontology that provides a shared vocabulary for resolving semantic heterogeneity. This global ontology can be a combination of multiple specialized ontologies. Single ontology approach is used when all the information sources to be integrated provide almost same view on the domain. Multiple ontology approach uses multiple ontologies where each information source is described by its own ontology. In hybrid ontology approach, the semantics of each source is described by its own ontology and these ontologies are built upon a global shared vocabulary to make the source ontologies comparable. During integration process, mappings are defined to connect the contents of an information source with ontology. In case of multiple ontologies, mapping between ontologies are defined. Ontologies are also used to define an integrated global query model. The user formulates a query in terms of ontology and later on this query is decomposed according to different sources and sent to the multiple sources. In some cases, ontology is also used to verify the mapping between local and global ontology.

[Lin07a] also uses an ontological approach to solve the problem of data integration by using ontologies of sources. Each source schema is described by ontology. The system is based on multiple ontologies of sources, ontology for global integrated view and also ontology of the integration system for mapping description between elements of the global view and the local sources. In a survey of approaches to semantic integration by the ontology community, [Noy04] proposes two approaches i.e. “shared ontology approach” and “heuristics and machine learning approach” to discover mappings between two ontologies. [XC06] proposes ontology based framework for integrating and exchanging XML data by using ontologies. The frame work involves data integration and query processing. For data integration, a global RDF ontology is built by merging the local ontologies. The local ontologies are generated from XML Schemas. The mappings between global ontology and local XML Schemas are defined manually.

[HK08] develops a prototype for Web data integration system in the domain of Malaysian universities. The prototype assumes that each Web source has an underlying pre-existing local ontology on the Web but if ontology does not exist for a university then it must be created before. [HK08] resolves semantic schema heterogeneities between the Web source and a user query through semantic mapping between the domain ontology and local ontology. The system displays the GUI that is composed of super-concepts of domain ontology to a user. First, a user selects super concepts for query and then sub-concepts, attributes or relationships are selected. User cannot create
complex query. The ontology server selects the related local ontology with a user query domain and sends it to mapping modules. Mapping algorithm is used to map the user query to the relevant Web source query language. The algorithm fails, if similar terms of query concept and query attributes are not found in the local ontology.

2.5 Semantic Web

It is essential to improve the schema matching process. The schema matching process needs an improvement to enable computers and people to work in cooperation with the Semantic Web. The improvement can be done with the help of machine understandable data. Semantic Web aims to use the Web as a distributed knowledge store and to make the Web contents more accessible to the automated processes.

One of the important activities of online people is “search”. In traditional Information (IR) Technology, mostly search is based on the occurrence of words in the documents. Traditional search engines are not enough to meet the needs of searchers. Semantic Web offers some opportunities to improve the traditional search by representing the Web semantically. It can also increase the precision and recall of the search results [LDF02]. [GMM03] says: “the addition of explicit semantics can improve search. Semantic Search attempts to augment and improve traditional search results (based on Information Retrieval technology) by using data from the Semantic Web.”

Ontologies are used in different fields’ i.e. semantic Web, software engineering, artificial intelligence and information architecture for knowledge representation. Ontologies provide a key factor for enabling interoperability in the semantic Web by allowing applications to agree on the terms that they use when communicating. Ontologies furnish the semantics for the Semantic Web. When we talk about application integration domains, it is required to achieve semantic interoperability between very different systems. Problems in interoperability can be solved by developing ontologies and controlling the semantics.

[Lin04] explains the ontology’s value within application integrations and states that “The use of ontologies concept within modern application integration techniques and technologies seems to be a good match”.

2.5.1 What Is an Ontology?

“In the context of computer and information sciences, an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members)” [Gru08]. Ontology includes machine-interpretable definitions of basic concepts in the domain and relations among them.

Ontology establishes and share terminology between members of a community of interest (human or automated agent) and helps in knowledge sharing between people and computers. Ontology also provides other benefits like re-use of knowledge and easy way for domain knowledge analysis. Ontology provides an explicit conceptualization for entities in a specific domain and annotation for the current Web pages. Annotations are meta-data useful for machines to understand the content of a
Web page. In short, ontology in the context of computer science helps to understand the concepts to share information in a domain [NM01].

2.5.2 Main Components of Ontology

Ontology consists of following main components:

- **Individuals**

   Individuals (also known as instances) represent objects in the domain that we are interested in. Individuals can be concrete objects i.e. people, animals, tables or abstract individuals i.e. words.

- **Classes**

   Set of individuals form classes (also known as concepts). Class may contain other classes too and might be organized into super-class or sub-class hierarchy. For example, class “Elephant” might be a sub-class of “Animal” saying that all the elephants are animals (or “Animal” is super-class of “Elephant”).

- **Attributes**

   Attributes are used to describe the objects in ontology. Each attribute has a name and value to store specific information about object. For example, object elephant has attribute “age” with value 5.

- **Relationships**

   Attributes are used to describe the relationship between object in the ontology. Important types of relationships are Is-a, is-subtype-of, is-subclass-of, part-of and made-in. For example, Vienna is-in Austria and Austria is-a country of Europe.

2.5.3 Kinds of Ontologies

Following are the main kinds of ontologies [AH04]:

- **Domain ontologies**

   Domain ontologies describe specific field of endeavor and used to model specific domain, or part of the world. Domain ontologies are used to represent the particular meanings of the terms as they apply to the domain. For example, the word “apple” has different meaning in ontology for computer domain as compared to ontology for fruit domain. Domain ontologies represent very specific concepts as compared to upper level ontologies.

- **Upper-Level Ontologies**

   Upper level ontologies describe the basic concepts and relationships about any domain. Upper ontologies (also called foundation ontologies) are used to model common objects that are generally applicable to wide range of domain ontologies. For example, Dublin core, WordNet, ResearchCyc etc.
• **Integrated vocabularies**

Integrated vocabularies are developed when a number of independently developed vocabularies are merged into a single large resource.

• **Topic Hierarchies**

Topic hierarchies are simply a set of terms, loosely organized in a specialization hierarchy. Topic hierarchies are not strict taxonomy but mix different specialization relations.

2.5.4 **Ontology Applications**

Ontologies can be used to enhance the functionality of the Web. Resources can be represented semantically in ontologies and improve the performance and accuracy of search engines. Ontologies and Web search engines are built on domain basis which provide a foundation for a meta-search engine to be developed on domain basis as well. These meta-search engines can be proved very helpful by providing an interface with multiple remote search engines. Meta-search engines can select and the rank remote search engines intelligently [LDF02].

In application integration scenarios, terms defined in ontology help in fully understanding the meaning and context of information, no matter where the information resides [Lin04].

Scientists, researchers and regularity authorities demand the integration of diverse and heterogeneous data sets that originate from distinct communities of scientists in separate subfields. This integration is achieved in a large part through the adoption of common conceptualizations referred to as ontologies [SBH06].

2.5.5 **Semantic Web: Languages**

Ontology language standardization is an important issue of the World Wide Web Consortium (W3C). It is required to have powerful, simple and extensible ontology language that can capture the most desired information to allow agents to perform inferences. Ontology language should have well defined syntax, formal semantics to describe the meaning of knowledge, convenience of expression, sufficient expressive power and efficient reasoning support. Now a day, there is direct growth to support for RDF and various species of OWL [AH04]. Following is the detail about few ontology languages with their features:

2.5.5.1 **RDF**

The Resource Description Framework (RDF) is W3C standard for describing and interchanging metadata using XML. RDF can be used to describe a collection of books, jobs, competencies or a collection of Web pages to create machine readable summaries of Web sites. RDF describes resources with property and property values. Resource is anything that can have Uniform Resource Identifier (URI), including all the Web pages as well as individual elements of an XML document i.e. http://www.w3schools.com/RDF. Property is a resource that has name i.e. author.
Property value is a value of property i.e. ‘Jan Egil Refsnes’. The RDF data model combines the resource, a property and a property value making a statement in the form of subject-predicate-object (also known as triple in RDF terminology). For example, in triple

“The author of http://www.w3schools.com/Rdf is Jan Egil Refsnes”,

http://www.w3schools.com/Rdf is a subject, author is a predicate and Jan Egil Refsnes is an object. In triple “Tina has skill of leadership”, “Tina” is a subject that can have URI http://www.tuwien.ac.at/staff/tina, “has skill of” is a predicate and “leadership” is an object.

As RDF documents are written in XML, so RDF information can easily be exchanged between different type of computers using different types of operating systems and application languages [Con04b] [W3Sb] [AH04].

2.5.5.2 RDF Schema

RDF Schema (RDFS) specifies the data schema for defining RDF statements. As we need a schema or a DTD to define the grammar of the domain specific language, in the same way in RDF we need a schema to define the type hierarchy of the language. RDFS is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes. RDFS is an extension of RDF but limited to a subclass hierarchy and a property hierarchy with domain and range definitions of these properties.

RDFS allows the representation of some ontological knowledge but lacks some features i.e. local scope of properties, disjointness of classes, boolean combination of classes, cardinality restrictions, transitive properties and inverse properties [Con04c][W3Sc][AH04].

2.5.5.3 OWL

Web Ontology Language (OWL) supports richer expressiveness than RDF Schema. OWL offers more features and considered as an extension of RDF Schema. OWL is aimed to be the standardized and broadly accepted ontology language of the Semantic Web. OWL adds more vocabulary for describing properties and classes, relation between classes (e.g. disjointness), cardinality (e.g. exactly one), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

OWL ontologies may be categorised into three species or sub-languages i.e. OWL-Lite, OWL-DL and OWL-Full. A defining feature of each sub-language is its expressiveness.

OWL-Lite is the least expressive sub-language and syntactically simplest sub-language. It must be used in situations when a simple class hierarchy and simple constraints are required.

OWL-DL based on Description logic, is more expressive than OWL-Lite and considered as an extension of OWL-Lite. The expressiveness of OWL-DL falls between that of OWL-Lite and OWL-Full. It is possible to check inconsistencies and automatically compute the classification hierarchy on OWL-DL ontologies
OWL-Full is the most expressive sub-language and considered as an extension of OWL-DL. OWL-Full must be used in situations where very high expressiveness is important. It is not possible to perform automated reasoning on OWL-Full ontologies. OWL-Full is fully upward compatible with RDF both syntactically and semantically [AH04] [HKR+04].

2.5.6 SPARQL

Simple Protocol and RDF Query Language (SPARQL) provides the means of conveying a query to a query processor service, and the XML format in which query results will be returned. [Xml05a] says: “SPARQL is based on matching graph patterns. Graph patterns contain triple patterns. Triple patterns are like RDF triples, but with the option of query variables in place of RDF terms in the subject, predicate or object positions. Combining triple patterns gives a basic graph pattern, where an exact match to a graph is needed”. SPARQL actually consists of following three separate specifications:

- The query language specification
- The query results XML format which describes an XML format for serializing the results of a SPARQL SELECT (and ASK) query.
- The data access protocol which uses WSDL 2.0 (Web Services Description Language 2.0) to define simple HTTP and SOAP (Simple Object Access Protocol) protocols for querying any data repository that can be mapped to the RDF model [Con07a] [Xml05a].

2.6 Impact of Semantic Web on E-Recruitment

The KnowledgeNets project [BHM+05] [MOH04] introduces an approach to solve the problems in recruitment process by technologies from the Semantic Web. Using Semantic Web technologies, the data exchange between employers and job portals can be based on a set of controlled vocabularies which provide shared terms for describing occupations, required skills and educational background to perform semantic matching [BHM+05]. All job portals can operate on the same information and postings would reach more applicants, resulting in higher market transparency. Job portals could offer semantic matching services, which would calculate the semantic similarity between job postings and applicant’s profiles.

[BHM+05] emphasizes that both job portals and employers must use a set of controlled vocabularies that can provide shared terms for the description of occupations, required skills and educational background. [BHM+05] claims the development of human resource ontology by the integration of existing standards and classifications. This ontology is divided into sub-ontologies i.e. JobPositionSeeker, JobPositionPosting, Skills, Person, Education and Organization.

[BHM+05] uses existing taxonomies Standard Occupation Classification scheme (SOC), BKZ (German Version of SOC) HR-BA-XML (German version of HR-XML), NACIS (North American Industry Classification System), WZ2003 (German classification of Industry sector) and KOWIEN ontology for the development of HR-Ontology. HR-XML and few of these classifications are explained in section 2.7. HR-ontology in [BHM+05] can be used in both job posting and job application
descriptions. [BHM+05] also develops skill sub-ontology that defines concepts representing competencies, derived from KOWIEN ontology and defines the competence level. [BHM+05] states that “We focus on how vocabularies can be derived from standards already in use within the recruitment domain, on how the data integration infrastructure can be coupled with existing non-RDF human-resource systems, and on matching algorithms for calculating the similarity of job requirements and applicant’s profile”.

Due to large number of applications, recruiters face the problems in choosing the best person for their company. Job seekers also face the problems to emphasize their competencies in CVs. Due to these problems both recruiters and job seekers have to face difficulties in employee or job search. [HLT02] explains the importance of Web talking to machines with the help of Semantic Web and emphasizes to benefit from it in different fields of life including e-recruitment. [HLT02] works on the project CommOnCv to transform the current Web site into real Career Networks i.e. virtual places where both job seekers and employers can find relevant and efficient services to meet their respective needs. [HLT02] proposes a competency model with a formal representation of competencies as a part of the curriculum vitae or a job offers. It is also defined that identification and representation of a competency can be based on different ontologies i.e. sector ontology, enterprise ontology and behaviour ontology. The competency model can be helpful in the matching of CVs and job offers. [HLT02] uses RDF to represent the competencies in a CV.

Application of Semantic Web in the recruitment domain can be beneficial for the job seekers as well as employers. Job seekers will be able to find more relevant jobs for them and employers will be able to have potential applicants. Moreover the cost on job publication for recruiting company will also be reduced [BHM+05].

2.7 Human Resource Management

Human resources management includes a variety of business processes such as advertising staffing needs, recruiting and training the best employees, enrolling employees and their dependents within benefit plans, filing reports and notifications to government agencies, performance issues etc. Many processes are related to each other and there are problems while data exchange between processes because it require data format decisions. HR-XML consortium is working to solve these problems in data interchange [AP01].

2.7.1 HR-XML

HR-XML organization (www.hr-xml.org) is dedicated to the development and promotion of a standard suite of XML specifications to enable e-business and the automation of human resources-related data exchanges. By developing and publishing open data exchange standards based on XML, the Consortium provides the means for any company to transact with other companies without having to establish, engineer, and implement many separate interchange mechanisms. XML Schemas define the data elements for particular HR transactions, as well as options and constraints governing the use of those elements. The HR-XML Consortium has 120 member organizations all over the world. It has produced schemas covering major processes, as well as component schemas that are used across multiple business processes [AP01].
The following section contains the introduction to Recruiting and Staffing workgroup, Competency workgroup, Resume Workgroup and Staffing Industry Data Exchange Standards 1.0 (SIDES) of HR-XML with brief description of deliverables.

2.7.1.1 Recruiting and Staffing

Recruiting and Staffing workgroup develops Staffing Exchange Protocol (SEP) to support common types of Internet based recruitment and staffing data exchanges. SEP is a set of XML specifications. [HRX06a] says: “Transactions supported by the Staffing Exchange Protocol include the posting of job or position opportunities to job boards and the exchange of a candidate resume and/or profile data independent of or related to those postings to other recruiting and sourcing venues”.

2.7.1.2 Competencies

The Competencies Workgroup develops competencies model that can be used in any process for easy comparison, ranking and evaluation of competencies. It can also solve the problems faced due to multiple skill taxonomies. HR-XML’s competency model can provide an easy mapping between multiple skill taxonomies. The schema is capable of referencing competency taxonomies from which competency descriptions have been used [HRX03a].

2.7.1.3 Resume

Resume is used to interchange data between different recruitment companies. It is emphasized that resume module should contain distributed guidelines, executive summary, objective, employment history, educational history, licence and certifications, military history, patent history, publication history, speaking event history, qualifications (competencies), languages, achievements, associations, references, security credentials and resume additional items [HRX01a].

2.7.1.4 Staffing Industry Data Exchange Standards 1.0 (SIDES)

The SIDES document describes the business processes involved in typical staffing transactions in general terms. The eight major SIDES schemas include

- StaffingOrder to describe staffing customer’s servicing need.
- HumanResource to describe a staffing resource employed by a staffing supplier.
- Assignment for information confirming the actual work agreed upon to be done.
- Staffing Supplier to describe a staffing company.
- Staffing Customer to describe a customer.
- Staffing Action for request and response schema, e.g. interview request and response.
- Extended TimeCard to show the actual amount of time spent by the staffing resource performing work on the assignment over a particular time period.
- Invoice for information associating the actual time expended as shown via the Timecard, with the bill rate, as confirmed in the Assignment [HRX03b].

HR-BA-XML is an extension of HR-XML. HR-BA-XML is to improve the data exchange for a job seeker and employer. HR-BA-XML is also known as German
version of HR-XML. HR-XML-SE contains some original parts of HR-XML and adds some more schemas with Swedish extensions [TE04].

2.7.2 Classification and Standards

There exist some efforts on taxonomies for describing occupations and competencies. Some well known classification schemes for occupations and competencies/skills are given in section 2.7.2.1 and 2.7.2.2. These classification schemes, standards and their combinations can be used in the development of ontology for human resource management.

2.7.2.1 Occupation Classifications

Following are some occupation classification schemes developed by different agencies or groups in different parts of the world to facilitate international communication about occupations. Occupational classifications can be used at the unified interface of the job meta-search engine to guide the job seeker in the selection of appropriate job category for him.

US Federal agencies have classified occupational categories in order to meet the need of occupational classification system (http://www.bls.gov/soc). The Standard Occupational Classification Scheme (SOC) is used for the purpose of collecting, calculating or disseminating data. Federal agencies use it for collecting occupational data and providing the means to compare occupational data across agencies. In SOC 2000, all workers are classified into one of over 820 occupations according to their occupational definition. The 2000 SOC classifies workers at four levels of aggregation i) major group ii) minor group iii) broad occupation and iv) detailed occupation.

All occupations are combined to form 23 major groups, 96 minor groups, 449 broad occupations and 821 detailed occupations. Occupations with similar skills or work activities are grouped at four level of hierarchy to facilitate comparisons. Table 2.1 shows the classification of “Computer and Mathematical Occupations” at four levels. All SOC major groups are available at http://www.bls.gov/soc/soc_majo.htm in English language. Next major review and revision of the SOC will be available for use in 2010 Decennial Census.

The International Standard Classification of Occupations (ISCO-88) is another tool for classification of jobs into a clearly defined set of groups according to the tasks and duties undertaken in the job. The main aim of ISCO-88 is to facilitate international communication about occupations. ISCO-88 has been based on two main concepts i.e. job and skill. ISCO-88 uses two dimensions of skill concepts i.e. skill level and skill-specialisation. The focus in ISCO-88 is on the skills required to carry out the tasks and the duties of an occupation and not on whether a worker in a particular occupation is more or less skilled than another worker in the same or other occupations. ISCO-88 has 10 major groups, subdivided into 28 sub-major groups, 116 minor groups, and 390 unit groups. The current version of the ISCO-88 is available in English, French and Spanish. ISCO-88 is available at http://www.ilo.org/public/english/bureau/stat/isco/isco88/index.htm

contains the results collected from twelve EU countries with their knowledge experts in occupational classification domain. ISCO-88 (COM) is a result of coordinated effort by National Statistical Institutes to implement ISCO-88 for census and survey coding purposes.

ISCO-88 (COM) is available at http://www.warwick.ac.uk/ier/isco/brit/intro.html and its grouping is available at http://www.warwick.ac.uk/ier/isco/brit/grplist.html. It is available in English, French and German language.

Table 2.1: Classification of Computer and Mathematical Occupations in SOC 2000

<table>
<thead>
<tr>
<th>Major</th>
<th>Minor</th>
<th>Broad</th>
<th>Detailed</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-0000 Computer and Mathematical Occupations</td>
<td>15-1000 Computer Specialists</td>
<td>15-1010 Computer and Information Scientists, Research</td>
<td>15-1031 Computer Software Engineers, Applications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15-1030 Computer Software Engineers</td>
<td></td>
</tr>
</tbody>
</table>

International Classification of Status in Employment (ICSE-93) classifies with respect to the type of explicit or implicit contract of employment of the person with other persons or organizations. The basic criteria used to define the groups of the classification by the ICSE-93 are, the type of economic risk, an element of which is the strength of the attachment between the person and the job, and the type of authority over establishments and other workers which the job incumbents have or will have. ICSE-93 has six groups; i) employees ii) employers iii) own-account workers iv) members of producer’s cooperatives v) contributing family workers vi) workers not classifiable by status. ISCE-93 is available in English language at http://www.ilo.org/public/english/bureau/stat/download/res/icse.pdf

2.7.2.2 Competence/Skill Classifications

Following are some competence or skill classifications schemes developed for different purposes i.e. to determine skill trends, for skill identifications or for helping job seekers and recruiters.

Information and Communications Technology (ICT) European classification describes the skills and competencies required by the ICT industry in Europe. ICT European classification has been developed with the support of the European Commission, a consortium of eleven major ICT companies (BT, Cisco Systems, IBM Europe, Intel, Microsoft Europe, Nokia, Nortel Networks, Philips Semiconductors, Siemens AG, Telefonica S.A., Thales), and EICTA, the European Information and Communications Technology Industry Association with the co-ordination of International Co-operation Europe Ltd (ICEL 2001). The framework has been developed for students, education institutions and governments. The main aim of the project is to determine the ICT trends in Europe to develop a methodology which would lead to a better quantification of the resources required by the industry. It has 13 major groups that are further divided into sub-groups.
The 13 major groups of ICT European classification are as follows:

- Radio Frequency (RF) Engineering
- Digital Design
- Data Communications Engineering
- Digital Signal Processing Application Design
- Communications Network Design
- Software and Application Development
- Architecture and Design
- Multimedia Design
- IT Business Consultancy
- Technical Support
- Product Design
- Integration and Test/Implementation
- Systems Specialist

ICT European classification is available at http://duda.imag.fr/edu/supplydemand.pdf

*The Skills Framework for the Information Age (SFIA)* is a common reference model for the identification of the skills needed to develop effective Information Systems (IS) by making use of Information Communications Technologies (ICT). SFIA can be helpful for the employers of IT professionals and is available at http://www.sfia.org.uk in English language.


*Employer type classification, Classification of employment types and Position Classification* are to provide assistance to job seekers for job search and guidelines to include employment classification information in job advertisements. These classification schemes are available at http://www.sigmod.org/dbjobs/posting-guide.html in English language.
CHAPTER 3  
SYSTEM REQUIREMENTS

In chapter 2, we have explained existing researches in the construction of meta-search engines for different domains but these techniques need some improvements. Compared to WISE and MetaQuerier, our approach to schema matching aims to use a domain ontology to resolve semantic conflicts. Compared to Lixto, we aim to provide an automatic and simple construction process for information extraction. Moreover, no one of them is specifically for e-recruitment domain. Compared to vertical search engines (such as http://www.kayak.com and http://www.skyscanner.net), which provide hard-wired solutions, we are aiming for a configurable and extensible approach. Compared to JobOlize, we aim to provide not a job analyser that can analyse the job offers from the Web pages but also a system that can integrate different job search engines into one. We distinguish our work from Web-scale architectures such as PAYGO [MJC+07], in that we are aiming to develop techniques to support the construction of domain-specific meta-search engine, rather than Web-scale search of the deep Web. Our aim is to combine the respective benefits of all vertical search engines, meta-search engines and semantic search engines within a domain-specific context, in which there is a well-understood domain ontology.

In the recruitment domain, both recruiters and job seekers face the problems during the recruitment process. In the current recruitment process, applications or CVs are submitted in different ways often in unstructured hard copy format, via email or company’s Web site. The recruiters face the problems in searching most suitable candidate from the rain of CVs. The recruiters also have to optimize their handling and administration processes. Job seekers have to monitor a large number of job search engines and are unable to keep the track of employment market. So it is required to design a job meta-search engine that can solve the problems of both parties involved and can meet the various demands that the modern companies and job seekers expect from current online job market.

In this dissertation, we have proposed a new configurable job meta-search engine that would be able to meet the requirements of a meta-search engine provider and a job seeker. The recruiters can ask the meta-search providers to develop a job meta-search engine according to their demands. Job meta-search provider needs to construct a configurable meta-search engine, which would be used by the job seeker to search a job. Proposed configurable job meta-search engine will be able to provide an ideal platform for the construction of a meta-search engine and job tracking processes. The rest of the chapter discusses the requirements for a new configurable meta-search engine from job meta-search provider’s and job seeker’s point of views. Requirements to improve the efficiency of job meta-search engine have also been identified. We have considered the development of meta-search engine as different process than the deployment process. We have also identified the requirements for both processes separately.

3.1 Meta-search Engine Provider’s Requirements

Meta-search engine providers want to have a job meta-search engine that fulfills the following requirements:
• Need to construct a job meta-search engine automatically without any human intervention. It must support automatic integration of interfaces, data, structures and processes.

• Need for an automatic and quick development of meta-search engine.

• Minimal manual efforts for schema matching and integration for meta-search engine construction.

• Automatic extraction, matching and integration of individual schemas from different job search engines into one. It also needs translation of individual extracted schemas from individual search engines into a common schema.

• Able to handle the heterogeneous nature of Web data, understand and integrate schemas semantically.

• Capable of resolving semantic heterogeneity/conflicts both at schema level and data level during meta-search engine construction.

• Able to handle mapping cardinality problems during schema matching and integration and solve direct as well as indirect mapping problems.

• Need for a standardized and easier query interface making framework for the meta-search engine.

• Ability of automatic source selection according to the preferences of the meta-search provider. For example, if meta-search provider wants to offer jobs only for country Austria, then the search engines supporting jobs only in Austria must be automatically chosen for the job meta-search engine.

• Need for a configurable meta-search engine so that meta-search provider can set his preferences. For example, meta-search provider can set the currency for salary. Meta-search provider can also set salary-range to show it on weekly, monthly or yearly basis.

• Need to support fully automatic wrapper generation for the extraction of job results from the result pages returned by multiple search engines.

• Able to automate the pre-selection of candidates by adding more job sites in the meta-search engine. This process of selection of candidates helps in decreasing the time to hire and transaction cost from the company’s point of view.

3.2 Job Seeker’s Requirements

Job seeker wants to have a meta-search engine that fulfills the following requirements:

• Job meta-search engine must be capable of automatic source selection according to the preferences of job seekers. So that when job seeker submits a query, the meta-search engine checks which job search engine may have relevant answers to the query. For example, job seeker wants to have all the jobs related to “IT” therefore
meta-search engine should exclude the job search engines that do not support “IT” jobs to save time and system resources.

- Easy to understand interface of meta-search engine. Meta-search interface must contain standardized attributes and their values i.e. job category, type of hour etc so that job seekers can easily construct the job search criteria for multiple search engines.

- An option to give short or long search criteria for a job search.

- Need to give their preferences by job meta-search interface. For example, job seeker wants to rank the jobs according to their preferences i.e. by salary or geographical area.

- Job meta-search engine must be configurable for the job seeker too. For example, job seeker may want to have salary in some particular currency or salary-range.

- Able to semantically understand the job results and convert them into a single format for quick and easy navigation of the jobs to the job seekers.

- Meta search engine needs to have most up-to-date job offers from multiple job sites.

- There must be a standardized way provided by the proposed system so that job seekers can emphasize their competencies during job search.

- Job postings must be available 24 hour a day.

- Need to expand the job search to increase the chances of identifying job openings and to break into the hidden job market not just locally and regionally, but nationally and internationally as well.

- Need to have a job agent where job seekers can create their CVs and mention their job preferences. Job agent can search jobs for them according to the job preferences and send them via email.

### 3.3 Other Requirements

Some of the overall requirements to improve the efficiency of meta-search engine are given below:

- Design patterns for meta-search engine and its components are required to be constructed to accelerate the meta-search engine development process.

- There is a need to develop a set of controlled vocabularies for job portals and employers to provide shared terms for the description of jobs, occupations and required skills.

- Unlike traditional search engine, new meta-search engine must be able to understand the meaning of terms. Job categories or competencies given in the
search criteria by the job seekers must not be treated as string but new meta-search engine must be able to understand the meaning of them. Meta-search engine must also understand the synonym and hypernym relationships of these terms given as job categories or competencies.

- Proposed system must solve the problem of selection of generic or specific concepts related to the job categories for the meta-search interface.

- Meta-search engine must be fast enough and must keep up-to-date jobs. Meta search engine must scan the jobs from the job search engines on regular basis and should store them in advance for future use to save the precious time of the user.

- Job results from the meta-search engine should be stored in the cache memory for further use.

- Meta-search engine must be able to semantically understand the job search criteria especially job keywords and if jobs are already available in the cache memory, then it must not start the search from scratch but must display job results from already searched jobs.

- Meta-search engine must keep the record of frequently searched jobs and these jobs should be searched in advance for the job seekers.

- New system must be easy to operate for both job seekers and meta-search providers.

- Meta-search engines must increase the Web coverage by combining multiple search engines. It must also have an ability to search the invisible Web to increase the precision, recall and the quality of results.

- Meta-search engine must also be able to identify the jobs according to geographic location. There must be a geographical model to decide the place of a job.

- Meta-search engine must be able to identify different regularities according to geographic location. For example, in some countries extra salary is given at Christmas event and in other countries extra bonus is given at religious events.
CHAPTER 4
SYSTEM DESIGN

The following chapter contains the meta-search architecture for general meta-search engine, design patterns for general meta-search engine, architecture and system design for job meta-search engine and ontology development for human resource management.

4.1 Meta-Search Architecture

This section presents an overall architecture for meta-search engine construction. We have identified that there can be two processes involved for the construction of meta-search engine: i) Meta-search engine creation process and ii) Meta-search engine usage process. Figure 4.1 and 4.2 show the components of meta-search engine architecture that support these two processes.

The meta-search engine creation process (see Figure 4.1) is as follows. First, the meta-search provider submits its preferences via the Preferences Collector. Preferences may be for which countries or geographical areas meta-search engine is required. The Search Engine Selector is then activated, and search engines that meet the preferences of meta-search provider are selected from an already known set of URLs of candidate search engines. This can be done by analyzing the URL, IP address and country of the search engine. Next Interface Extractor derives attributes from those search engine’s interfaces. The process of interface extraction has two phases: a) attribute extraction, and ii) attribute analysis. The XML Schema Generator then creates an XML scheme corresponding to each search interface. Finally, Query Interface Generator matches and integrates different XML-Schemes to have a single query interface of meta-search engine.

Figure 4.1: Meta-Search Creation Process Components
The meta-search engine usage process (see Fig. 4.2) is as follows. An information seeker can access and use the meta-search query interface generated by the meta-search creation process. Queries submitted to the query interface are rewritten by the Query Dispatcher in order to target the individual source search engines. The Query dispatcher submits the rewritten queries to the individual search engines. The result pages from various search engines are passed to the Information Extractor (IE). IE is responsible for extraction of records and results from the result pages. The extracted results are merged by the Result Merger component. Duplicate results are removed by Duplicate Result Eliminator and stored in a database for further use. Finally, the results are ranked by the Result Ranker according to the preferences of the information seeker and displayed to the seeker.

![Figure 4.2: Meta-Search Usage Process Components](image)

**4.2 Design Patterns for Meta-Search Engines**

Research on a large scale is in progress in the field of developing configurable meta-search engines in different domains i.e. jobs, hotels, flights, news, research papers and real estate etc. It has been observed that developing a configurable meta-search engine in any domain is a tedious and time consuming task. Every time developers have to start the development process from scratch. It is required to have a reusable and flexible design for meta-search engine. We have proposed design patterns for meta-search engines construction and their components so that they can be reused in future several times after some modifications. Presented design patterns for meta-search engines and their components are extendable and flexible. The design patterns of meta-search engine i.e. result ranker can also be reused in some other application domains too. These design patterns can help the developers in accelerating the development process in meta-search domain and other application domains. These design patterns also provide developers with shared vocabulary for easy communication.
This section contains the designed patterns for the above processes and main components of meta-search engines. Figure 4.3 and 4.4 describe our meta-search engine creation and usage processes by using abstract factory design pattern. “Abstract factory pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes. It defines the interface that all concrete factories must implement, which consists of a set of methods for producing products” [GHJ+95] [FF04]. Meta-search abstract factory can produce any type of meta-search engine at a time, as long as it gets proper set of directions called factories.
Figure 4.3: Abstract Factory Pattern for Meta-Search Engine Creation Process
Figure 4.4: Abstract Factory Pattern for Meta-Search Engine Usage Process
A factory will build a certain type of meta-search engine, depending upon the type of factory. It is clear from the abstract factory pattern in figures 4.3 and 4.4, that family of job meta-search object or any meta-search object can be created from developer or user perspective. Each type of meta-search engine has same overall structure that all meta-search engines share in common i.e. interface extractor, XML-schema generator, query interface generator, information extractor, result merger, duplicate result eliminator and result ranker etc. Factories for hotels, flights or real estate’s search etc can be added easily. Using meta-search abstract factory pattern, we do not need to worry about what kind of meta-search we’re building. An abstract class may contain a default method i.e. a simple ranking algorithm, that can be used by every concrete meta-search engine, but will be refined if more specific ranking is derived. Job-MSC-Factory in figure 4.3 represents Job-Meta-Search-Creation-Factory and X-MSC-Factory in figure 4.3 represents a Meta-Search-Creation-Factory for any domain i.e. flight, real estate, hotel etc. Job-MSU-Factory in figure 4.4 represents Job-Meta-Search-Usage-Factory and X-MSU-Factory in figure 4.4 represents Meta-Search-Usage-Factory for any domain [DNo8].

4.2.1 Design Pattern for Query Interface-Generator

Query interface generator component of meta-search engine is responsible for schema integration, data integration and then production of query interface for meta-search engine. Integration of interface schemes is divided into two parts i.e. schema matching and data integration. During schema matching, semantic correspondence between interface attributes is identified and each scheme is translated into a single schema for the query interface. During data integration, the values of different attributes for the user interface are determined. It is required that values are semantically unique and compatible with the local values. Different methods have been discovered for schema and data integration by using i) hybrid approach that uses domain ontology (see Section 4.3.4), ii) clustering approach [HMY+04b] or iii) holistic schema matching approach [CHZ05] [HZC05]. Different approaches for schema matching meet specific requirements according to the specific context. So user must have a facility to choose one of the above mentioned approaches according to the specific requirement. It is required that new algorithms for schema and data integration comply with the same interface can be easily introduced with less efforts and without changing the other code.

![Figure 4.5: Strategy Pattern for Meta-Search Query Interface Generator](image-url)
To meet above requirements, the strategy design pattern as shown in figure 4.5 is used for query interface generator. “Strategy design pattern defines a family of algorithms, encapsulate each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it” [FF04].

Design pattern in figure 4.5 represents that Meta-Search-Query-Interface-Generator is a class that is responsible for schema and data integration of different search engines and Schema-And-Data-Integrator is an interface. Integrate strategies are not implemented by the class Meta-Search-Query-Interface-Generator. Instead, they are implemented separately by sub-classes of abstract Schema-And-Data-Integrator class. Sub-classes of abstract Schema-And-Data-Integrator class implement different integrate strategies i.e. Integrate-By-Hybrid, Integrate-By-Clustering and Integrate-By-Holistic-Schema-Matching. To switch schema and data integrator strategies, each meta-search engine calls the integrate method. If user wants to add new schema and data integration algorithm into the system then it can be done easily by implementing a new class using the Schema-And-Data-Integrator interface.

For meta-search in human resource domain, we have used our “Integrate-By-Hybrid” approach during query interface generation (see Section 4.3.4). Compared to Integrate-By-Clustering approach and Integrate-By-Holistic-Schema-Matching approach, our Integrate-By-Hybrid approach also uses multiple matchers and a domain ontology to resolve semantic conflicts during scheme/data matching and integration.

4.2.2 Design Pattern for Information Extractor

Information extractor component is responsible for the extraction of results from the result pages. It consists of Record collector component and Result Collector component. Record collector component is responsible for the identification of the record section from the result page i.e. list of jobs, table with flights etc.

Result collector component is responsible to extract the exact fields i.e. job salary, hotel price etc from the identified record section. Information extraction research shows that information extraction from different Web sites is often performed by using wrappers. Wrappers can be constructed manually, semi-automatically and automatically for record section identification. For the identification of record section, different approaches i.e. i) automatic wrapper generation as ViNTs [ZMW+05] or our hybrid approach (see Section 4.3.6) ii) semi-automatic (supervised) wrapper generation as Lixto [BFG01] or iii) data mining approaches [LGZ03] can be utilized. After the identification of record section, meta-search result identifier component is utilized for the extraction of exact results by using i) hybrid approach (see Section 4.3.6) or ii) semi-automatic wrapper generation as Lixto information extraction tool again [BFG01].

For schema matching and merging it is required to normalize the terms like “Posted Date” into “Post Date” or “Job Types” to “Job Type”. Stemming algorithms i.e. porter stemming algorithm can be utilized for term normalization process.
Strategy design pattern is utilized for information extraction component of meta-search engine. Design pattern in figure 4.6 shows that Meta-Search-Information-Extraction class is responsible for information extraction from different result pages. Result-Collector and Record-Collector are interfaces. Sub-classes of Record-Collector (abstract class) implement different record identification strategies i.e. Record-Identifier-By-Automatic-Wrapper-Generation-Techniques, Record-Identifier-By-Semi-Automatic-Wrapper-Generation-Techniques and Record-Identifier-By-Data-Mining-Techniques. Sub-classes of Result-Collector (abstract class) implement two result identification strategies i.e. Result-Identifier-By-Hybrid-Approach and Result-Identifier-By-Semi-Automatic-Wrapper-Generation-Techniques.

For meta-search in human resource domain, we have used “Record-Identifier-By-Automatic-Wrapper-Generation-Techniques” for record extraction and “Result-Identifier-By-Hybrid-Approach” for result extraction. Our hybrid approach during result identification uses multiple matchers and a domain ontology. Section 4.3.6 includes the details about how the records can be collected by analyzing pattern format for an automatic wrapper. It also elaborates that how the results can be extracted by using multiple matchers and a domain ontology.

### 4.2.3 Design Pattern for Result Merger

Results extracted from different search engines need to be merged and stored for future use. Results can be stored in database or in XML format. We have re-used result merger design pattern for result merger component from [ZQD+03] with little changes.

Abstract factory pattern is used for the result-merger component of meta-search engine (see figure 4.7). Meta-Search-Result-Merger (abstract factory) defines the interface that all concrete factories i.e. Database-Factory and XML-Factory must implement, which consists of a set of methods for merging results. The concrete factories i.e. Database-Factory and XML-Factory implement the different product families i.e.
create, add, update, delete and query. To merge results, the client uses one of the factories and each factory knows how to create the right object for the right merging process. Few parts in figure 4.7 are not drawn for the sake of saving space.

For meta-search in human resource domain, we have used “Database Factory” i.e. MySQL database server to merge the results from different search engines.

4.2.4 Design Pattern for Result Ranker

Result ranker component can rank the results according to the preferences of the seeker. Rank preferences of the seekers can vary according to their personal choice or

meta-search engine type i.e. job, hotel. For example, the seeker may want to rank the flight results according to the price, hotel according to the nearest location or job
according to the salary. For merging results into a single ranked list according to the query relevance anyone of the algorithm from TopD, TopSRR, SRRSim, SRRRank, SRRSimMF [LMS+05] can be utilized.

Strategy design pattern is utilized for result-ranker component of meta-search engine (see figure 4.8). Meta-Search-Result-Ranker class is responsible for ranking of results. Sub-classes of Rank-Algorithm (abstract class) implement different ranking strategies i.e. Rank-By-Price, Rank-By-Location, and Rank-By-Relevance.

For meta-search in human resource domain, we have used “Rank-By-Price” ranking strategy to rank the jobs according to the offered salary from various search engines.

4.3 System Design for Meta-Search in Human Resource Domain

According to the identified processes in the construction of “general meta-search engine”, there are two processes involved in the construction of job meta-search engine i) meta-search engine creation process for job meta-search provider and ii) meta-search engine usage process for job seeker. Figure 4.9 and 4.10 show the main components of our meta-search architecture that support these two processes and meet the requirements of job meta-search provider and job seeker.

In human resource domain, the meta-search engine creation process (see Figure 4.9) is as follows. First, the job meta-search provider submits its preferences via the Preference Collector. Currently, preferences may be for which countries or geographical areas meta-search is required. In the longer term, we plan to extend this

Figure 4.9: Meta-Search Creation Process Components for Meta-search Provider

to capture also information about preferred units of data types (e.g. preferred currency and preferred periodicity for salaries e.g. yearly salary, monthly salary, weekly salary
etc). The Job Search Engine Selector is then activated and job search engines that meet the preferences of meta-search provider are selected from already known set of URLs of candidate job search engines. Next Interface Extractor derives attributes from job search engine’s interfaces. The process of interface extraction has two phases: i) attribute extraction, and ii) attribute analysis. The XML Schema Generator then creates an XML Schema corresponding to each search interface. Job ontology is available for the job meta-search engine. This ontology is used by the Job Meta-search Query Interface Generator to create mappings between the XML Schemas and the ontology, and hence indirectly mappings between the different XML Schemas via the ontology. The Job Meta-search Query Interface Generator undertakes three main steps: i) schema matching and integration, ii) data integration and iii) generation of single query interface for the meta-search. Different search engines use different representation concepts, data structures and different granularities of knowledge, so our hybrid approach is used for translation between concepts. Our hybrid approach consists of element, structure and ontology based techniques with multiple matchers for schema and data integration (we give more details of all the components in the next part of this chapter).

In human resource domain, meta-search engine usage process (see Figure 4.10) is as follows. Job seeker can access and use the job meta-search interface generated by the job meta-search creation process. Queries submitted to query interface are re-written by the Query Dispatcher in order to target the individual source search engines, using the mapping rules. The query dispatcher submits the re-written queries to the individual search engines. The result pages from various search engines are passed to

**Fig. 4.10: Meta-Search Engine Usage Process Components for Job Seeker**

*information Extractor (IE) component. Automatic wrapper generation techniques are used for the implementation of this component. In particular, the IE consists of Record Collector and Result Collector sub-components. Record Collector component is responsible for identification of the record section from each job result page i.e. a list or table containing jobs. It also identifies the required URL and title of each identified record. Result Collector visits the identified URL and is responsible for extracting the*
job description, and the record fields, e.g. job salary, job start date, job requirement, from the result page. Since different job search engines use different concepts and data structures for job description in the result pages; Result Collector utilizes again the domain ontology and variety of matching techniques in order to conform the different concepts and data structures of result descriptions and result attributes, and to convert them to a single common format for presentation to the job seeker. The conformed results are merged by the Result Merger component. Duplicate results are removed by Duplicate Result Eliminator and stored in a database for further use. Finally, the results are ranked by the Result Ranker according to the preferences of the job seeker and displayed to the job seeker. The system solves problems of different representational concepts in different job search engines.

As we use Semantic Web technologies in the construction of our meta-search engine therefore we call it Semi-JIN “Semantic Job search engine INtegrator”. Occupation schemes and HR-XML are involved in the construction of domain ontology in HR domain. Domain ontology and multiple matchers with stemming algorithm and string distance functions are used in the proposed system during the integration process. Following is the design of main components of job meta-search engine as explained above:

4.3.1 Search Engine Selector

Search engine selector is designed to meet the requirements of the meta-search provider. Search engine selector is responsible to choose the job search engines to be included in the job meta-search engine. These job search engines are selected from already known set of URLs of candidate job search engines according to the preferences of the meta-search provider. For example, job meta-search provider wants to develop a meta-search for a country i.e. UK then all the job search engines from already stored list of job search engines that support country UK will be selected for inclusion in the meta-search engine.

4.3.2 Interface Extractor

The integration of schemes in a certain domain seems to be easier as between different domains because sources are more homogeneous and are thus easier to be integrated [CHZ05]. However, different search interfaces in the same domain can contain different number of attributes, different names for representing the same type of elements and organize the attributes in a different way as shown in figure 4.11.

Figure 4.11: Job Search Interfaces a. jobs.net; b. aftercollege.com; c. careerbuilder.com
Jobs.net and aftercollege.com have interfaces with different attributes (see figure 4.11). Jobs.net has an attribute “Employment Type” and aftercollege.com has an attribute “TYPE OF WORK” to represent the same concept. Jobs.net and aftercollege.com do not ask about “Degree” of a job seeker but careerbuilder.com asks about it. So there are problems in the integration of different job search portal’s interfaces in the job domain as interfaces are representing the schemas in different ways.

First step in the construction of meta-search engine is to construct a machine understandable format of the search interface of each portal. For machine understandable format, we should extract interface of each job search engine by the interface extractor component of meta-search. Interface extraction is designed and implemented almost in the same way as given in [HMY+04a] and [HMY+05a] but it has been developed specifically for job search portals with some extensions.

In the “Interface extraction” phase we identify logic attributes by grouping related labels and elements and then derive meta information of these attributes. The process of interface extraction is further divided into two phases i) Attribute Extraction ii) Attribute Analysis.

4.3.2.1 Attribute Extraction

The aim of attribute extraction is the identification of labels and elements and grouping of related elements. First, the URL of a portal is given to the interface extractor. The HTML form may be of devoted type or co-existing type [PMH+04]. If the interface is of devoted type, the program can take the form as a target form otherwise meta-search provider has to select the target form. <FORM> and </FORM> tags of HTML are used to identify a form.

![Figure 4.12: Job Search Interface of jobshejobs.com](image)
Secondly, individual labels and form control elements (e.g. input fields, checkboxes, and radio buttons) are extracted. Text between elements is extracted to determine the labels for control elements. Moreover <BR>, <P> and </TR> tags are also extracted to determine the physical location of elements and labels. Extra scripting and styling information i.e. font sizes, styles is ignored. Logically, elements and their associated labels together form different attributes. Attributes can have one or more labels and elements. For example, the “Skills/Keywords” attribute in figure 4.12 has four elements including a text area and three radio buttons.

Thirdly, an interface expression (IEXP) to provide a physical layout of a search interface is constructed. ‘t’ is used to represent a text or label, ‘e’ to represent an element and ‘|’ to represent line separation. IEXP for the interface in figure 4.12 is t|e|t|tt|tt|tt|ttt|teee|tt|tt|te|ee|ee|ee|t|tt|tt|t|e|t|e|t|e|t|e|e|t|t|e|t|e|t|e where first “t” represents a label “Job Category”, first ‘e’ represents a select list element next to a label “Job Category”, first “|” represents a line separator and so on.

Fourth step is to group the labels and elements that semantically correspond to the same attribute, to find an appropriate attribute label/name for each group. In figure 4.12, a label “Skills/Keywords”, a text area, three radio buttons and their values all belong to the same attribute. For grouping labels and elements, a LEX (layout-expression-based extraction) technique described in [HMY+05a] is used. LEX finds an appropriate attribute label for each element, either in the same row or above the current row. During the grouping process, if the neighbour text of an element is not considered as an attribute label of e, then it is assumed to be an element label of e. For example in figure 4.12, “From” and “To” are recognized as element labels for text boxes instead of attribute label. Our interface extractor uses heuristics measures i.e. ‘Ending Colon”, ‘Distance of element and text’, ‘Vertical Alignment’, ‘Priority of current row’ from [HMY+05a] with minor changes to identify an appropriate label for elements. One heuristic measure is introduced to count number of labels with colons and without colons from an IEXP. If a number of labels with colons is greater than the number of labels without colons then apply the ending colon heuristic otherwise choose closest label for an element.

4.3.2.2 Attribute Analysis

Aim of the “Attribute Analysis” phase is to collect information about elements i.e. relationship type (RT), domain type (DT), default value (DV), value type (VT) and unit. Semantics of domain elements and meta information is also identified.

RT can be of “Group type”, “Range type” or “Part type”. Check if all elements of attributes are check boxes or radio buttons and are greater than one in number then RT is of group type. In figure 4.12, “Specify Employment Type” attribute is an example of group type elements. Check if labels contain some keywords or patterns e.g. “between”, “from”, “to”, then RT is of range type. Few job search portals contain “from”, “to” labels with “Salary Range” attribute as in Fig 4.12. Elements that are not of group and range type are treated as a part type.

Next step is to extract meta information attributes i.e. domain type (DT), default value (DV), value type (VT) and unit. DT indicates how many values can be specified on an attribute for each query. DT can be range, infinite, Boolean or finite. If RT type of
elements is of range type then DT is recognized as range. If relationship type is not
range and a textbox or a text area is involved then DT is infinite. If attribute has a
single checkbox, then DT is Boolean. Otherwise, if selection list is involved then DT is
finite. DV only occurs if there is a selection list, radio buttons or checkboxes and is
always marked as checked or selected in an element. If an attribute contains just
textboxes or text areas then attribute has no default value. VT can be determined by the
analysis of attribute name. It can detect date, time, currency, integer and string data
type. If an attribute name contains “range” or “number” then value type is numeric.
Otherwise if the value type is not date, currency, and number then it is considered as a
string. In job search portals, sometimes salary attribute contains a unit in label or
values of some attribute. Interface extractor can detect unit if label contains “EUR”,
“€” etc. Table 4.1 represents the schema model including attribute names and meta
information collected during interface extraction and attribute analysis phase for a job
interface shown in figure 4.12.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>RT</th>
<th>DT</th>
<th>DV</th>
<th>VT</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>job_category</td>
<td>None</td>
<td>Finite</td>
<td>All categories</td>
<td>String</td>
<td>Nil</td>
</tr>
<tr>
<td>job_location</td>
<td>None</td>
<td>Finite</td>
<td>All locations</td>
<td>String</td>
<td>Nil</td>
</tr>
<tr>
<td>or_province</td>
<td>None</td>
<td>Finite</td>
<td>-Select Province-</td>
<td>String</td>
<td>Nil</td>
</tr>
<tr>
<td>skills_keywords</td>
<td>Group</td>
<td>Infinite</td>
<td>Any of These</td>
<td>String</td>
<td>Nil</td>
</tr>
<tr>
<td>salary_range</td>
<td>Range</td>
<td>Range</td>
<td>Nil</td>
<td>Integer</td>
<td>Nil</td>
</tr>
<tr>
<td>specify_employment_type</td>
<td>Group</td>
<td>Finite</td>
<td>All Types</td>
<td>String</td>
<td>Nil</td>
</tr>
<tr>
<td>job_posted_in_last</td>
<td>None</td>
<td>Finite</td>
<td>All Periods</td>
<td>String</td>
<td>Nil</td>
</tr>
<tr>
<td>sort_jobs_by</td>
<td>None</td>
<td>Finite</td>
<td>Post Date</td>
<td>String</td>
<td>Nil</td>
</tr>
<tr>
<td>Show_jobspage</td>
<td>None</td>
<td>Finite</td>
<td>10</td>
<td>String</td>
<td>Nil</td>
</tr>
</tbody>
</table>

4.3.3 XML Schema Generator

Schema model developed by interface extractor is used by XML Schema generator to
define the legal building blocks of an XML document. An XML Schema defines the
elements, attributes, child elements, order of child elements. It also defines the data
types, default and fixed values for elements and attributes [W3C04]. Character set for
schema is collected from the HTML page; if there is “charset” attribute otherwise
consider “iso-8859-1” as a default character encoding. During this process schema
elements and XML Schema equivalents for HTML elements are identified and are given
in more detail in our work [Naz06].

4.3.3.1 Schema Elements

<RootJob> is automatically created with a sequence indicator as the root element of
schema and it contains all other elements from the search interface as child elements.
An XML Schema may contain simple and complex elements.
All simple and complex elements from the schema model are identified for XML Schema. As explained in chapter 2, simple elements contain only text and cannot contain any other element or attribute, while complex elements contain other elements and may contain attributes as well. For example, in figure 4.12, “Job Location” is a simple element and “Salary Range” is a complex element that contains “from” and “to” as child elements. If labels “from” and “to” are on the HTML page for textboxes, these labels are used as name of elements. Sometimes when labels are not available, internal names of elements can be used. Elements can have a “type” attribute that refers to the name of complex type to use.

### 4.3.3.2 XML Schema Equivalents for HTML Elements

In this section, we explain how HTML elements i.e. text filed, text area, radio button, check box and select list from the HTML search interface can be represented in XML Schema.

i. Text boxes and text areas on the search interface are represented as simple elements.

ii. A group of multiple radio buttons on search interface is also a simple element having a default value, restriction and enumeration list. Text/label associated with radio button is taken as a value for that radio button.

iii. Multiple checkboxes on a search interface with domain type “group” are treated as complex type element with attributes “fixed” and “minOccurs”. The <minOccur> specifies, how many values are selected for a checkbox.

iv. If select element in HTML does not contain “multiple” attribute then the select list is a single-select list otherwise it is multiple select list. A single-select list on search interface is treated as a simple element having a default value, restriction and enumeration list in the same way as radio buttons. But multiple-select list on the search interface is treated as complex type element and it must contain a “type” attribute that refers to an element of complex type. In figure 4.12, “Jobs Posted in last” is a single-select list and “Job Category” is a multiple-select list [Naz06][W3C04].

A complete XML Schema for search interface can be developed by combining XML equivalents for each HTML element. A complete XML Schema generated by our XML-schema generator component of meta-search engine is given in Appendix A.

### 4.3.4 Meta-search Query Interface Generator

Query interface generator of meta-search engine is responsible for integration of meta-data in three steps: i) schema integration ii) query interface generation and iii) data integration. Query interface generation process and all the techniques used for meta-search query interface generation are given in detail in this section.

#### 4.3.4.1 Schema Integration

During schema integration, schemes generated for different job portal’s interfaces are translated into a HR-XML Schema for a meta-search of jobs.

Table 4.2 shows a list of common attributes, available in different job portals. An asterisk (*) in a cell marks the presence of the attribute. “Job Category” represents a grouping
of jobs under one or more classification schemes that is meaningful to an organization i.e. IT, Accounting, Education. “Job Type” represents the type of hours i.e. full time, part time. In some portals “Job Type” also represents the nature of the position i.e. contract, temporary, volunteer. Some job portals provide more specific concepts for job category and represent it as “Industry”. Some other attributes found are “Travel”, “Experience” and “Posted within”. “Travel” means that if the person is willing to travel, “Experience” means information about work experience or education in years, “Posted within” means when job was being posted. Some attributes are related to decide on the ranking.

Table 4.2: Common Attributes in Job Domain

<table>
<thead>
<tr>
<th>Job Web sites</th>
<th>Attributes</th>
<th>Keyword</th>
<th>Location</th>
<th>State</th>
<th>Country</th>
<th>Province</th>
<th>City</th>
<th>Job Category</th>
<th>Job Type</th>
<th>Industry</th>
<th>Degree</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jobs.net</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aftercollege.com</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Clearchannel.com</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jobmonkey.com</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Careerbuilder.com</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Techjobsonline.com</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Promotions.</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brightspyre.com</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Careerscafe.com</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jobinterviewonline.com</td>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jobshejobs.com</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search2.workopolis.co</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Directjobs.com</td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alljobsearch.com</td>
<td></td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Integration of interface schemes is divided into two parts: schema matching and schema merging. During schema matching, semantic correspondence is identified between interface attributes and each schema is translated to the HR-XML Schema. The used HR-XML Schema is derived from “Job and Position Header”, “Worksites”, “Educational History”, “Postal Address” and “HR” schemes. Table 4.3 represents the name of HR-XML schemes and attributes in HR-XML schemes that are used for capturing common attributes of job search portals.

During schema merging, a single scheme is derived for the meta-search user interface. A domain-ontology contains the HR-XML attributes and attributes from job portals. For an attribute of the job portal interface, a corresponding attribute from HR-XML is obtained by using the similarity relation of the ontology. The HR-XML attributes are further used in the construction of the unified user interface.
4.3.4.2 Query Interface Generation

We need to construct a user interface for meta-search engine which contains all distinct fields. [DYM06] emphasizes the importance of well-designed unified query interface and meaningful labeling of elements. It also states that the labels assigned to their elements must be carefully chosen to convey the meaning of each individual element. For example, one job portal uses “Employment Type” to represent the job category and other may use “Type of work” or “Job Type”. From different attribute names, the user interface must contain the most meaningful and appropriate one. The problem of carefully choosing the meaningful label is solved by HR-XML schemes. During the query interface generation, a unified form given the above discovered schema matching and merging is constructed.

Many job search engines have been considered for the elements for meta-search query interface. The order of elements in the user interface has also some importance. There was possibility to have the intersection or union of all attributes from different job search portals. We have considered union of all attributes to have a maximal interface for the meta-search query interface. Upper part of meta-search query interface can contain competency, job category and geographical area related attributes while lower part can contain attributes like type of hours, education type, salary, preferences, travel, posted within, show number of jobs per page, sort job results by, create profile, and change language etc. The number of attributes on the meta-search query interface depends upon the type of job search interfaces to be integrated.

The form generation is supported by XForms which enables the generation of the form from an XML Schema and also the easy adaptation to different user clients [RDH04].

4.3.4.3 Data Integration

The aim of the data integration is to determine the values of different attributes for the user interface. We have to choose the values that are semantically unique. These values should also be compatible with the local values. Since the search engines use different data, different concepts and different granularities of knowledge we use a domain-ontology to translate between concepts.

After analyzing job search portals, we have discovered that there are two types of semantic relations: synonymy and hyponymy between concepts. Synonymy means that two terms $x_1$ and $x_2$ are synonyms, if they have same meaning. For example, “programmer” is a synonym for a “coder”. Hyponymy means that a term $x_2$ is

Table 4.3: HR-XML Attributes and Schemas for Common Job Portal’s Interfaces

<table>
<thead>
<tr>
<th>Job Portal’s Attributes</th>
<th>HR-XML Schema</th>
<th>Attributes in HR-XML Schemes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Category, Industry</td>
<td>Job and Position Header</td>
<td>JobCategory</td>
</tr>
<tr>
<td>Job Type</td>
<td>Job and Position Header</td>
<td>PoistionType, TypeOfHours</td>
</tr>
<tr>
<td>Qualification</td>
<td>Education History</td>
<td>SchoolName, Degree</td>
</tr>
<tr>
<td>Location, State, Country, Province, City</td>
<td>Postal Address</td>
<td>Region, Municipality, CountryCode</td>
</tr>
<tr>
<td>Travel</td>
<td>Human Resource</td>
<td>Preferences</td>
</tr>
</tbody>
</table>
hypernym of \( x_1 \), if \( x_1 \) is more generic concept of \( x_2 \). For example, “IT” is hypernym of “IT-Hardware”.

For finding synonyms and hypernyms in job portals, again our domain ontology is utilized. Normally, hypernymy relationships exist for attribute “Job Category” or “Industry” in job search interfaces. In the job domain, most of the attributes take alphabetic values and are of finite type, so we have focused on the merging of alphabetic domains. Only “Salary” and “Experience” attributes can take numeric values and salary-display also requires currency conversion. If alphabetic values are synonyms, then we have to choose which to represent in the user interface. To solve this problem, we get help of our domain ontology. We maintain a list of synonyms and then choose corresponding attribute i.e. class name from the domain ontology.

If values are hypernymy (mostly in case of job category), then we find a semantic relationship between values by using the taxonomy for job categories. Global values for the user interface may represent a generic concept or a specific concept. Both of them have their pros and cons. If generic concepts are chosen then query against the unified interface may need to be mapped to multiple values in some local interfaces. If we keep specific concepts, then for users who are interested in more generic concepts may have to submit multiple queries using the more specific concepts, resulting in less user-friendly interface. Therefore, a combined approach is used to solve this problem providing a hierarchy of values, including both generic and specific concepts. Multiple categories may be formed for the values corresponding to each global and a value hypernymy hierarchy is created for each category [HMY+04a].

Figure 4.13 represents that interface placementindia.com has only generic concepts for attribute job category i.e. Computers/IT whereas interface clickjobs.com has specific concepts i.e. IT-Hardware, IT-Networking and IT-Software.

A domain-specific ontology identifies the relationship between the values of two interfaces. If in the user interface the generic concepts “Computer/IT” is represented then a user interested only in a specific field i.e. IT-Hardware will get irrelevant results. Therefore, query against the user interface may need to be mapped to multiple values in some individual interfaces. If only specific concepts are represented in the user interface i.e. “IT-Hardware, IT-Networking, IT-Software” then a user who is interested in all the categories, will have to make three queries to get the desired result.

The solution is a combined approach providing a hierarchy of values, including both generic and specific concepts as in figure 4.13c. If a user is interested in Computer/IT related jobs then the meta-search engine can generate one query for figure 4.13a.
interface and three queries for figure 4.13b interface. But if a user is interested in IT-Software the meta-search can generate one query for figure 4.13a interface and one for figure 4.13b interface. This solution can solve the problem and helps the user in job search.

Computer and business related occupations (job categories) from widespread used Standard Occupation Scheme (SOC) and skills from International Co-operation Europe, Ltd have integrated into one format. This format is maintained in job ontology. Both generic and specific concepts of occupations are displayed at the meta-search query interface from the domain ontology and user can select any one of them.

4.3.4.4 Techniques Used for Meta-search Query Interface Generation

Our key requirement is to provide automatic techniques for schema/data matching and integration, utilizing the cumulative techniques from the database and ontology communities. We use multiple matchers, some of which use multiple string distance algorithms. Stemming and removal of stop words are also used. We adopt a single-ontology approach and utilize the domain ontology to find matchings between attributes of different search engine interfaces; a synonym matcher is also used during this process. The mappings generated are stored in XML format in integrated XML Schema for use by the query interface generator (see Appendix A for integrated XML Schema) and query dispatcher components. After schema/data matching and integration, a form for meta-search engine is generated (see Appendix B for job meta-search integrated interface).

Since our meta-search engine generation and creation processes use multiple matching criteria and multiple techniques/matchers, we term this a ‘hybrid’ approach. We use a combination of element-level techniques, structure-level techniques and ontology-based techniques (see [SE05] for a general review of the main techniques used in schema matching). The techniques that we use and an algorithm that discusses our schema and data integration are described below:

4.3.4.4.1 Element-Level Techniques

The element-level techniques we use include a string-based matcher, language-based matcher, data values cardinality matcher, ID matcher, default value matcher and alignment reuse matcher. We use element-level techniques to find similarity between schema elements, and between data values.

i. String based matcher

Our string-based matcher uses a stemming algorithm and different string distance functions to find a similarity between strings. In particular, the porter stemming algorithm removes the prefix and suffix of a string, handles singular and plural of concepts, and then finds the similarity between strings. The following are examples resolved with the porter stemming algorithm [Por06]:

Keywords Æ keyword

Provinces Æ province

Industry Æ industries
We utilize three different string distance algorithms, Levenshtein distance, Cosine similarity and Euclidean distance, and an aggregate of their similarity scores is used. The following are examples of strings matched using these string distance functions:

* Business operations → business
  * Intern → internship
  * Contractor → contract
  * Engineering software → software engineering
  * Financial services, banking → banking

### ii. Language based matcher

Our language-based matcher is based on natural language processing techniques, including tokenization and elimination. Tokenization involves the removal of punctuation, blank spaces, and adjustment of cases. Elimination involves the removal of stop words (a list of stop words for the given domain needs to be provided to the system). Stop words includes articles, prepositions and conjunctions etc. The following are examples of strings transformed using tokenization and elimination:

* “Enter a Keyword” → keyword
* “Career type(s)” → career types
* “Select a State:” → state
* “Full-time” → full time

### iii. Data values cardinality matcher

Our data value cardinality matcher uses the cardinality of attributes to find a match. For example, suppose an attribute “Job Type” that contains 7 data values may match an attribute “Type of Hour” that contains 8 data values or an attribute “Job Category” that contains 44 data values. In this situation, the number of data values can be compared, from which it can be inferred that attribute “Job Type” is more similar to “Type of Hour”.

### iv. ID matcher

If element name matching fails, then our ID matcher may help to find a match. Some examples of IDs from job search engines for the element “keyword” are
Suppose a search engine contains an element with name “Type of Skills” and ID="kwd". Suppose that the element name fails to match with any element in ontology. In this situation, the ID matcher will be utilized and it will compare “kwd” to elements of the ontology e.g. keyword, type of hour, job category etc. With the help of the string distance functions above, the ID matcher will find a similarity between “kwd” and “keyword”.

v. Default value matcher

Sometimes, search engine interfaces provide default values with attributes, so that if the user does not select any value, the default value is used. If a default value is available, our default value matcher can be helpful in increasing the matching results. For example, suppose there is ambiguity between the “Job Type” attribute of one schema and the “Type of Hour” or “Job Category” concepts of the domain ontology. The default value matcher can find that the default value “intern” of the “Job Type” attribute is matched to data value “internship” of the “Type of Hour” concept.

vi. Alignment re-use matcher

As already noted, our schema/data matching process is based on a domain ontology. This ontology is incrementally extended with synonyms and hypernyms of attributes from previously matched schemas. As soon as new matching is found, we store it in the domain ontology. When matching fragments of schemas, we employ an alignment re-use matcher to reuse these previously stored match results: if there already exists a matching for an attribute, then there is no need to attempt to match the attribute again.

4.3.4.4.2 Structure-Level Techniques

Structure-level matchers [RB01] consider a combination of elements that appear near to each other within a schema in order to identify a match. Two elements are considered similar if the elements in their local vicinity are similar. In particular, bottom-up structure-level matchers compare all elements in their sub-trees before two elements are compared i.e. data values are considered first. Top-down matchers compare first parent elements and, if they show some similarity, their children are then compared. This is a cheaper approach, and we utilize this, although it may miss some matches that a bottom-up matcher would detect.

For example, suppose there is a choice in matching an attribute “Job Type” of a schema with either attribute “Type of Hour” or attribute “Job Category” of the ontology. Our top-down matcher will match the children of “Job Type” with the children of “Type of Hour” (e.g. full time, part time, contract etc.) and with children of “Job Category” (e.g. computer science, business, engineering etc.). It will select whichever of these two attributes has the set of children having the closest combined match to the children of “Job Type”.

4.3.4.4.3 Ontology-Based Techniques

Finally, with respect to ontology-based techniques, we use a single ontology approach, and the domain ontology acts as a global ontology. After completion of the schema
integration process, the meta-search query interface generated contains concepts from this domain ontology. We recall that an XML Schema is generated for every search engine to be included in the meta-search. A synonym matcher is used to find similarities between such a source schema $S_1$ and the global ontology $O_G$, using synonyms associated with concepts in $O_G$. For example, in the job domain, synonyms for “job category” might be “industry”, “occupation”, “career type”, “function”. We note that a domain-specific ontology is likely to perform much better than traditional dictionaries or thesauri in finding semantic similarity between source terms.

During integration of two schemas, ontology acts as reference model to look up the relatedness of two schemas. The similarity between two schema elements can be determined by the distance between the two matching elements within the taxonomy [ADM+05].

For example, we have taxonomy $O_G$ and two schemas i.e. $S_1$ and $S_2$ as shown in figure 4.14. For example, the elements System Tester and System Programmer of $S_1$ and $S_2$ respectively are hyponyms of Software Engineering in the $O_G$. As System Tester and System Programmer share the same hyponym so there is some similarity between the two and a similarity matcher can assign a similarity value. Similarity value is dependent on the distance between the two terms within the taxonomy. As there is no syntactic relation between two terms System Tester and System Programmer but semantic nearness by taxonomy (tree in ontology) based matching can help to measure the similarity between schema elements.

![Figure 4.14: Semantic Nearness by Taxonomy based Matching](image)

### 4.3.4.4 Map Cardinality

Our algorithm can generate 1:1 mappings at the schema level, and 1:1, 1:n, n:1 and m:n mappings at the data level.

Few examples of 1:1 handled map cardinalities are

- **Keyword-title** $\rightarrow$ **Keyword**
- **Business opportunity** $\rightarrow$ **Business**

At data level integration, it can handle 1:n n:1 or m:n map cardinalities too. Few examples of handled 1:n, n:1 and m:n mapping cardinalities are

- **Information technology** $\rightarrow$ **Computer Science, IT**
4.3.4.4.5 Schema and Data Integration Algorithm

Following is an algorithm for schema and data integration that uses domain ontology and above mentioned multiple matchers and generates an integrated XML Schema for the meta-search engine.

Algorithm 4.1: Algorithm for Schema and Data Integration in Integrated XML Schema for job Meta-search engine by using domain ontology and multiple matchers

**Input:** XML Schemas of different job search engines

**Output:** Integrated XML Schema for meta-search engine after schema and data integration.

1. While( i<=total_no_of_schemas) where total_no_of_schemas = total number of XML Schemas of different job search engines and i= counter starts with one
2. 
3. Store “URL” of job search engine in integrated XML Schema
4. Store query access “Method” of job search engine in integrated XML Schema
5. Get all (m) job related attributes from the ith XML Schema where m= total number of job-attributes

   //SCHEMA INTEGRATION
6.   for (j=1; j<=m ; j++)
7.   {
8.     If there exist equivalent attribute for job attribute (j) directly by domain ontology’s classes, sub-classes or by synonym matcher
9.     Store attribute ID and equivalent attribute in the integrated XML Schema
10.    Else-if there exist equivalent attribute for job attribute (j) by applying string based matcher or language based matcher
11.    Store attribute ID and equivalent attribute in the integrated XML Schema
12.    Else-if there exist equivalent attribute for job attribute (j) by data-value cardinality matcher or top-down matcher
13.    Store attribute ID and equivalent attribute in the integrated XML Schema
14.    Else-if there exist equivalent attribute for job attribute (j) by ID or default-value matcher
15.    Store attribute ID and equivalent attribute in the integrated XML Schema
16.   } //end for j

   // END OF SCHEMA INTEGRATION

   //DATA INTEGRATION
17.   for (k=1; k<=m; k++)
18.   {
19.     If there exist children for kth job attribute
20.     {
21.       Get all children attributes (n) for the kth job attribute from XML Schema where n is total number of children for a particular job attribute
for (l=1; l<= n; l++)
{
    If there exist equivalent attribute for child (l) directly by domain ontology classes, sub-classes or synonym matcher
    Store child-name and equivalent for child in the integrated XML Schema
    Else-if there exist equivalent attribute for child (l) by string based matcher or language based matcher
    Store child-name and equivalent for child in the integrated XML Schema
} //end for l
} //end if
} //end for k

//END OF DATA INTEGRATION

The XML Schemas of the source search engines are given as input to our schema and data integration algorithm, and an integrated XML Schema for the meta-search engine is generated as an output. All the discovered mappings are stored within this schema. Our algorithm works as follows:

First, the set of attributes (i.e. schema elements) from every source XML Schema is extracted, and the schema matching and integration process starts. For every attribute, the algorithm attempts to find an equivalent attribute in OG by applying multiple matchers in the following order: a) directly within OG, possibly using also the synonym matcher, b) using the string-based matcher or language-based matcher, c) using the data-value cardinality matcher or top-down matcher, d) using the ID or default value matcher. If an equivalent attribute is detected at any step, the matching process stops and the discovered mapping is stored in the integrated XML Schema.

Our rationale for applying the various matching techniques in the sequence a)-d) above is as follows. The domain ontology is examined first, together with the use of a synonym matcher, as the ontology will be a source of high-quality, broad-coverage information about the domain. If a match fails to be found for an attribute, we then use the techniques in b) because they are cheaper than the techniques in c) (in terms of execution time), as observed from our experiments with several search engine interfaces. Finally, regarding d), we apply the ID and default value matchers last because they are of low precision: in many cases the ID is not meaningful (Web developers may use arbitrary IDs for HTML control elements) or a default value is not specified.

When the matching process for all the attributes is completed, the data matching and integration process starts. The children of each XML Schema attribute are matched against OG. Children attributes are only matched against attributes in OG if there has already been found to exist some similarity between the parent attribute in the XML Schema and the parent attribute in OG. The same matchers as in a) and b) above are applied in sequence and the mappings discovered are stored in the integrated XML Schema.
Our algorithm can generate 1:1 mappings at the schema level, and 1:1, 1:n, n:1 and m:n mappings at the data level. The integrated XML Schema generated, incorporating the discovered mappings, is then used for generating the integrated meta-search interface and for subsequent processing of queries.

4.3.5 Query Dispatcher

Query dispatcher is designed to meet the requirements of a job seeker. A job seeker can pose a query from the integrated meta-search interface. This query is rewritten by the meta-search engine to target all source search engines incorporated in the generation of the meta-search engine. Query dispatcher can dispatch the query of the job seeker from the meta-search query interface to corresponding individual job search portal’s query.

Some search engines use “Get” method to formulate a query while other may use “Post” method. XML-schema generator component extracts the type of method from HTML interface in advance and stores it in XML-schema. Type of “Method” is used to send a query to a particular search engine. As explained in chapter 2, in “Get” method data is encoded into the URL by the browser while in “Post” method the form data appears within a message body. Schema for every job search engine is stored in integrated XML Schema for meta-search engine as in listing 4.1.


```
<MetaSearchEngine>
    <JSE1>
        <URL> http://www.jobmonkeyjobs.com/GetJobs.rs </URL>
        <Method> Post </Method>
        <AvailableFor>
            <Country> Austria </Country>
            <Country> Pakistan </Country>
            <Country> United States </Country>
            <Country> United Kingdom </Country>
        </AvailableFor>
        <Competency> KEYWORD </Competency>
        <JobCategory id="CATEGORY">
            .......
        </JobCategory>
        <Location> STATE </Location>
    </JSE1>
    <JSE2>
        <Method> Get </Method>
        <AvailableFor>
            <Country> United Kingdom </Country>
            <Country> Pakistan </Country>
        </AvailableFor>
        <Competency> jobsearch </Competency>
        <JobCategory id="cid">
            <Business>
                <Accounting> accounting </Accounting>
                <Banking> banking </Banking>
                <Marketing> marketing, sales/marketing </Marketing>
                <Finance> finance </Finance>
            </Business>
        </JobCategory>
    </JSE2>
</MetaSearchEngine>
```
<Auditing/>
</Business>

<Computer_science> information technology, it/mis
    <Telecommunication> telecommunications </Telecommunication>
    <Internet> engineering internet, internet/new media </Internet>
    <Software_engineering> engineering software </Software_engineering>
    <Web_design> design</Web_design>
</Computer_science>

<Engineering>
    ..... 
</Engineering>

</JobCategory>

<JSE2>
    <URL> https://www.mymatrixjobs.com/candidate/Login.action</URL>
    <Method> Post</Method>
    <Keyword> keywordtitle </Keyword>
    ..... 
</JSE2>

<JSE3>
    <URL> https://www.mymatrixjobs.com/candidate/Login.action</URL>
    <Method> Post</Method>
    <Keyword> keywordtitle </Keyword>
    ..... 
</JSE3>

</MetaSearchEngine>

Domain ontology along with multiple matchers is used to find the semantic correspondence between the attributes of job portals and meta-search engine. Identified correspondence is stored in an integrated XML-schema for meta-search in XML format as shown in listing 4.1. Finally, integrated XML-schema for meta-search is used by the query dispatcher component to generate a full query for different job search engines from a meta-search engine.

Following is an algorithm for query generation and dispatching from meta-search interface:

Algorithm 4.2: Algorithm for Query Generation and Dispatching

Input: Competency, Job Category, Country etc from the meta-search query interface
Output: Equivalent queries for k job search engines where k = Total number of search engines involved in the meta-search construction.

1. Get Country = Country-value from the meta-search query interface
   Where Country-value = Job seeker’s choice mentioned in meta-search engine by a job seeker or detected by IP address of a job seeker
2. Get Competency = Competency-value from the meta-search interface by a job seeker
3. Get Job-category = Job-category-value from the meta-search interface by a job seeker
4. i=1 where i= search engine number, k= number of job search engines involved in meta-search, JSE_i = ith job search engine
5. While i <= k // Repeat for all the job search engines in the integrated XML-schema
for meta-search engine

If Country-value = One of the country of JSEi from the integrated XML-schema for meta-search

Get URL of JSEi from the integrated XML-schema for meta-search engine
Get Method of JSEi from the integrated XML-schema for meta-search engine // Get or Post
Find equivalent Country attribute for JSEi from the integrated XML-schema for meta-search engine
Find equivalent Country-value for JSEi from the integrated XML-schema for meta-search engine
Find equivalent Competency attribute for JSEi from the integrated XML-schema for meta-search engine
Obtain Competency-value from the meta-search interface
Find equivalent Job-category attribute for JSEi from the integrated XML-schema for meta-search engine
Find corresponding Job-category-value for JSEi from the integrated XML-schema for meta-search engine

.....................//same for all other attributes and their values
Generate query by concatenating URL, Country attribute, Country-value, Competency attribute, Competency-value, Job-category attribute, Job-category-value, and other attributes with values.

Check if method of JSEi is “Get” or “Post”
If Method = = Get,
Transmit query to JSEi by get method
Save result pages
i++
Goto 5
Else If Method = = Post
Transmit query to JSEi by post method
Save result pages
i++
Goto 5

Exit

After query generation and dispatching, HTML result pages with list of jobs are collected from the job portals.

4.3.6 Information Extractor

Information extractor component is used to extract individual job details from the collected job result pages with the help of record collector and result collector component as below:

4.3.6.1 Record Collector

Record collector of a meta-search engine identifies the job record section from the job result pages and extracts list or table of jobs with URLs and titles. A job record consists of at least job title and URL. Result pages returned by the job search portals after query dispatching are dynamically generated result pages. Result pages are
analyzed and pages with zero results are identified and omitted. It is observed that zero job result pages contain keywords like “0 jobs”, “no jobs”, “zero jobs” or phrase like “your search xyz returned 0 jobs”, “No jobs matched your search”, “Sorry, no jobs were found matching this request” or “No job listing currently in the database matched your menu choices” etc.

Result pages may consist of multiple forms with advertisements, other details and a job record section. Result page that contains a list of jobs need to be further analyzed for identification of job record section ignoring irreverent information i.e. advertisements etc.

There are different methods to generate wrappers to identify the job record section from search engine result pages. Few of them are as below:

4.3.6.1.1 Job Record Section Identification on the Basis of Keywords

In specific purpose search engine i.e. job search engines, we can define a list of domain related keywords i.e. job title, job category, company, posted date, location etc to locate the target record section. Job records can be identified from the located target record section.

<table>
<thead>
<tr>
<th>Title</th>
<th>Company</th>
<th>Posted</th>
</tr>
</thead>
<tbody>
<tr>
<td>PART TIME (West Vancouver, British Columbia)</td>
<td>PANEXPCOMPANIES</td>
<td>08/19/07</td>
</tr>
<tr>
<td>DB2 Database Administrator 1031A (Toronto, Ontario)</td>
<td>SOOLEY &amp; ASSOCIATES INC</td>
<td>08/18/07</td>
</tr>
<tr>
<td>Sr. J2EE Developer/Lead Weblogic Portal Developer (Mississauga, Ontario)</td>
<td>SOOLEY &amp; ASSOCIATES INC</td>
<td>08/18/07</td>
</tr>
</tbody>
</table>

Few job search engines show results in table format and use above mentioned keywords as titles of the columns as shown in figure 4.15. For such type of job search engines this method would be perfect.

But for the job search engines showing results in the list format do not use these keywords and show titles and URLs directly without using keywords as shown in figure 4.16. For such type of result pages, job record section identification on the basis of keywords will fail.

Few job search engines show results in table format and use above mentioned keywords as titles of the columns as shown in figure 4.15. For such type of job search engines this method would be perfect.

But for the job search engines showing results in the list format do not use these keywords and show titles and URLs directly without using keywords as shown in figure 4.16. For such type of result pages, job record section identification on the basis of keywords will fail.
4.3.6.1.2 Job Record Section Identification on the Basis of Dynamic Section

[ZMY06] defines that a typical result page consists of three sections i.e. static, semi-dynamic and dynamic. Static section remains same on the job result page and contains query independent content on the result page. Semi-dynamic section is affected by different queries but is not totally dependent on the content of specific query i.e. “Your search java returned 5 results” or “Your search information technology returned 20 results”. Dynamic section totally depends on the query terms and consists of a set of job records. It is observed that job records in the dynamic section have one display format.

In job record identification method on the basis of dynamic section, two result pages from the same job search engine with different query terms are matched. Dynamic section between start boundary and end boundary markers is identified. Start boundary marker can be detected at the end of static or semi-dynamic section on the result page. It is also required to ignore the query terms and digits to ignore the semi dynamic section from the result page. In the same way, end boundary markers are detected by reverse comparison and ignoring static or semi dynamic section. The section between start boundary marker and end boundary marker may contain job records. Start and end boundary markers for each job search engine can be stored for future job result extraction. But this method cannot work for all the sites and has many problems.

Few sites contain advertisements on the result page and these advertisements may change according to the searched query term that leads to the wrong identification of start and end boundary marker. Sometimes job sites also display job categories on the result page depending on the searched query term and that can lead to the wrong identification of dynamic section. Another drawback of this method is the generation of result pages with at least two different query terms and storing of start and end boundary marker in advance. It is possible that result page contain multiple dynamic sections. [ZMY06] investigated that 19 out of 100 search engines from the data set on [ZMW+05], in different domains produce result pages with multiple dynamic sections.

4.3.6.1.3 Job Record Section Identification on the Basis of Pattern Format

[ZMW+05] introduces a technique and developed a tool Visual information aNd Tag structure (ViNTs), to automatically extract the records from dynamically generated result pages. Job record section identification used by job meta-search engine on the basis of pattern format is a same type of technique as in [ZMW+05] with some modifications.

In this technique, regularity in visual content is used to extract the job record section from the result page. It is observed that in a result page, job records are similar to each other i.e. a hyperlink with a title, a brief description, location, date posted and a visual line. Job records are normally placed in the center of the result page and occupy a large portion of the result page.

Following five steps have been followed to identify the job record section and extraction of desired job URLs from the job result pages:
**Step 1: Pattern Construction**

First of all, pattern is constructed to provide a physical layout of a search records by considering the visual content features from HTML result page.

**Content Line:** Group of characters that visually form a horizontal line in the same section on the HTML result page is called a content line. Following types of content lines are identified for identification of job records:

**Link:** A content line that contains hyperlink “<a href>” tag of HTML is considered as a link line and is assigned a code L.

**Text:** A content line that only contains text is considered as a text line and is assigned a code “T”. Consecutive text lines are considered as a single text.

**Link-Head:** A content line that contains HTML tag <li> is considered as a link-head and is assigned a code “NN”.

**Record Separator:** A content line that contains HTML tags like <hr>, <tr>, <ul> is considered as record separator and is replaced with a visual line.

Considering above mentioned visual content features and their codes, pattern “TLTTT” is constructed for the job record in figure 4.17.

**Step 2: Identification of Candidate Patterns**

Next step is to identify the candidate patterns on the basis of exactly or almost same type of patterns. Candidate pattern is a pattern that can be a target job record pattern. Blank sections are removed and line numbers are also stored with the patterns to mark the start and end of the job record.

Following heuristics are used to identify the candidate patterns from the full HTML page patterns:

**Heuristic 1:** If pattern length is greater than 2 then consider that pattern as a candidate pattern for further identification otherwise ignore it. At HTML page job record consists of at least one link, title and text/bullet.

**Heuristic 2:** If pattern contains at least one link and text “LT” or “TL” then it is considered as a candidate pattern.

**Heuristic 3:** If current and next patterns are exactly same then it can be a candidate pattern.

**Heuristic 4:** If current and next patterns are not exactly same, then apply Levenshtein distance (LD) algorithm to check if they are almost same. If Levenshtein distance (LD)
is less than or equal to 3 (threshold value) then they are considered as almost same patterns and are candidate patterns.

In identification of candidate patterns, Levenshtein distance algorithm is used to find the similarity between two patterns. If two job records in figure 4.16 are carefully observed then we can see that two job records have slightly different pattern format. First job result record contains “more->” hyperlink at the end of job description (see second last line of first result record in figure 4.16) and second job result record do not contain “more->” hyperlink. Pattern for first job result record is LLTTTTLT and for second is LLTTTTT. It is required that both are identified as job records. Such types of minor differences in patterns are handled by Levenshtein distance algorithm. In this case, Levenshtein distance between two patterns is 1, representing that there is only a difference of one character in the strings. According to Heuristic 4 of step 2, these two patterns are considered as same.

**Step 3: Identification of Target Pattern**

Further processing on candidate patterns is required to identify the target job record patterns. From the candidate patterns weightage is assigned to the candidate patterns on the basis of frequency. Finally, candidate pattern with the highest weightage is selected as a target pattern.

**Step 4: Identification of Target-start-boundary Marker and Target-end-boundary Marker**

Line number of the first target pattern is considered as a candidate-start-boundary marker and the line number of the last target pattern is considered as a candidate-end-boundary marker. Candidate-start-boundary marker and candidate-end-boundary marker are further refined to have a target-start boundary marker and target-end-boundary marker. Nearest <table> or <ul> tag above the candidate-start-boundary marker is considered as the target-start-boundary marker and the closing </table> or </ul> tag after the candidate-end-boundary marker is considered as target-end-boundary marker. In between the target-start-boundary marker and the target-end-boundary marker is a target record section.

**Step 5: URL and Title Extraction**

Next step is to extract the URLs and titles from the target section. It is required to identify the correct URL from the job record as it may contain multiple URLs. For extracting URLs from target record section, target column is identified on the basis of “title or job title” keyword and URLs are extracted from the identified column. It can also be the case, that target column does not contain URL (thejobs.com). To handle such a situation, first URL in the job result record is considered as the desired URL. In case keyword “title or job title” is not available then extract first URL from the job record as the desired URL.

All the URLs and job titles from multiple job search engines are stored in a separate file. From 23 job search engines of different countries, job result records and URLs from 19 job search engines have been successfully extracted by using job record selection identification on the basis of pattern format.
4.3.6.2 Result Collector

Result collector of meta-search crawls/visits all the stored URLs, downloads individual result pages and identifies the job description. A job description may consist of type of work, salary, start date, end date, details, location and company etc on the individual job result page. Main aim of this step is to semantically understand the job descriptions of various search engines. Job title already extracted by a record collector component is used to identify the start of job description.

Different job search engines display job description in different formats at the job result page. Job descriptions use different number of attributes and names of attributes to represent the job details. For example, job result pages of techjobscafe.com use “Employment Term” and 6figurejobs.com use “Job Type” to represent “Type of Hours” attribute. Moreover, job result pages from techjobscafe.com use “Salary” and 6figurejobs.com use “Compensation” to represent a “Salary” attribute.

Different search engines use different concepts and data structures for results in their result pages. Our Result Collector component utilizes the domain ontology, multiple matchers, a stemming algorithm and multiple string distance functions in order to conform the different concepts and data structures of result descriptions and result attributes arising from different search engines (again by generating appropriate mappings). In this way, our hybrid approach is used in the extraction of search results too.

First step for result collection is label extraction from job result pages and next step is the result identification by ontology and matchers.

4.3.6.2.1 Label Extraction from Job Result Pages

During label extraction, labels are extracted from individual HTML job result pages as explained in attribute extraction phase (see section 4.3.2.1).

4.3.6.2.2 Result Identification by Domain Ontology and Matchers

After label extraction, labels are matched with the job ontology with the help of different matchers to convert them in a common format as below:

First, extracted labels i.e. concepts are directly matched with the concepts of “Job_Attribute” class from the job ontology. If the concept is directly found in the ontology then extracted label is replaced by the concept of “Job_Attribute”. For example,

Concept from job result page ➔ Concept from job ontology
“Salary” (techjobscafe.com) ➔ “Salary” of Job_Attribute

If the concept is not directly available then any one of the matcher i.e. string based matcher or language based matcher is used (explained in section 4.3.4.4.1). Extracted label is further stemmed or cleaned and is matched with the cleaned or stemmed concepts of “Job_Attribute” from the job ontology. For example,
Stemmed or Cleaned Concept from job result page → Stemmed or Cleaned Concept from job ontology
“Requirements” stemmed to “require” → “Requirement” stemmed to “require”
“Starting Date” stemmed to “start date” → “Start Date” stemmed to “start date”

If a match is not detected by above process then concept from the job result page is matched with the synonym of concept from the job ontology by using synonym matcher and is replaced by the actual concept. For example,

Concept from job result page → Synonym of Concept from job ontology
“Compensation” of 6figurejobs.com → Synonym of Salary is “Compensation”

If match is still not detected, then stemmed or cleaned concept from the job result page is matched with the synonym of concept and is replaced by the actual concept. For example,

Stemmed or Cleaned Concept from job result page → Synonym of Concept from job ontology
“Job Type(s)” stemmed and cleaned to “job type” → Synonym of “Type_Of_Hour” is “job type”

Below is an algorithm that describes the complete process of result identification and collection by domain ontology and different matchers:

Algorithm 4.3: Algorithm for Result Identification and Collection by Domain Ontology and Different Matchers

Input: Job result pages with job description from different job search engines

Output: Result page in a common format by using domain ontology and different matchers

1. While (i<=k) where k = total number of result pages from job search engines, i=counter for job result pages

2. {
3. Get ith result page with job description
4. Extract n labels from the result page where n= total number of labels, j=counter for labels
5. for (j=1;j<=n; j++)
6. {
7. If (Labelj = = concept from job attributes of job ontology)
8. {
9. Replace Labelj with the concept from job ontology
10. }
11. Else use string based matcher or language based matcher for stemming and cleaning of label and job ontology concepts
12. if (Stemmed or Cleaned Labelj = = Stemmed or Cleaned concept from job ontology)
13. {
14. Replace Labelj with the actual concept from job ontology
15. }
16. Else use synonym matcher to get synonyms from job ontology
17. if (Labelj = = Synonym of concept from job ontology)
18. {
19. Replace Labelj with the actual concept of synonym from job ontology
20. }
21. }
22. }
23. }
24. }
25. }
26. }
Else use string based matcher or language based matcher for stemming and cleaning of label

if (Stemmed or Cleaned concept of Label$_j$ = = Synonym of concept from job ontology)
{
    Replace Label$_j$ with the actual concept of synonym from job ontology
}

} // end for j
i++; goto 1
} // end while
Exit

Finally, result collector component converts result attributes into a single common format for presentation to the job seeker, by using job ontology and different matchers.

### 4.3.7 Result Merger

Result merger component of meta-search engine is responsible to merge all the results from multiple search engines into a single list. Job results collected by meta-search engine are further sent to the duplicate result eliminator component.

### 4.3.8 Duplicate Result Eliminator

It is possible that different job search engines return the same jobs so it is required to remove the duplicate jobs. Duplicate result eliminator of meta-search engine is responsible to detect and then remove duplicate jobs by the identification of same URLs or part of URLs. Following heuristics can be used for the identification of duplicates [RRS03]:

- A URL of the form http://techjobscafe.com often refers to http://www.techjobscafe.com
- If two result pages have same titles and the same relative paths in two different URLs then it is assumed that both URLs point to the same document.

After duplicates are eliminated, job results are stored in a database. MYSQL database contains a table “job-record” to store all the jobs with description i.e. URL, job title, type of hour, salary and country etc.

### 4.3.9 Result Ranker

Result ranker component can rank the jobs according to salaries and geographical distance depending on the preferences of a job seeker.
Salary identified and extracted from the job search engines need to be converted into a single format and currency. Result ranker is responsible to convert salary into a single format and then ranking of jobs. It is to be noted that salary field has been identified and extracted with the help of result identifier but its value might be represented in different formats depending on the job portal as below:

- Salary value may be in different currencies depending upon the job search engines or location of the job.
- Salary value is given but currency is not mentioned
- Salary value may be given on yearly, weekly, monthly or hourly basis depending upon the job search engine.
- Salary value may be given in a range format (minimum and maximum) or not.
- Salary value is expressed with k to represent 1000, million to represent 1,000,000.

First of all, a regular expression is used to extract digits from the salary string. From the job pages currencies are identified by matching currency-names, currency-symbols and currency-abbreviations with the “meta-country” table of MYSQL. If the currency is not identified by matching process, it is obtained by detecting the country through IP address of job pages with the help of “GeoLite Country” as explained in chapter 5.

After converting salary into a single currency format, it is also required to determine that salary given is on yearly, weekly, monthly or hourly basis from the salary string. Few examples of extracted salary string are 30,000 – 40,000 €, 20k to 25k, Upto 40k USD and Rs 25000 per month etc. Identification of “–, to, upto, per month” from the salary string can help in the detection of salary range. But if the salary range cannot be determined from the salary string, it can be identified from the range of “salary_range” MYSQL table and then salary is converted into a single format. If salary is expressed with k or million etc, then it is converted to the numeric format accordingly. After converting all the salaries in a single format, jobs are ranked according to the salaries.

If geographical area is the main ranking preference of the job seeker, the distance between the areas where jobs are being offered and job seeker’s preferred area needs to be calculated. The less the distance the higher the ranking position of the job.

4.3.10 Collection of Preferences

There can be two types of preferences. First the preferences of the meta-search provider and second the preferences of a job seeker. Preferences of meta-search provider can be the preferences of an organization that needs meta-search engine. Meta-search provider can prefer to have a meta-search engine for a particular geographical area or a job category. For example, a meta-search provider may want to have meta-search engine that offers jobs only in Austria. Another meta-search provider may want to offer IT related jobs only.

Jobs returned from individual search engines have salaries in different currencies. In proposed meta-search engine, meta-search provider can also give preference for currency, so that job seeker can view the job salaries in a single currency. Meta-search
engine detects the currency of job salaries returned from multiple search engines and converts the salary according to the preferences of meta-search provider.

Meta-search provider can also set the salary range that salary must be shown on daily, weekly, monthly or yearly basis. Job returned from different search engines may have different salary ranges. One search engine displays job salary on monthly basis and other on yearly basis. For the uniformity of job salaries, these are converted into one format i.e. daily, weekly, monthly or yearly according to the preference of meta-search provider.

Meta-search engine can rank the job results according to the preferences of the job seeker. Job seeker may want to rank the results by salary or geographical area. If preference of a job seeker is to rank by salary then jobs are ranked by the average of minimum and maximum salaries. But if minimum and maximum salary is not available then ranking is done on the basis of available salary. Job seeker may want to display salary in a particular currency so system provides a facility to convert the salary according to the preferred currency of a job seeker. Job seeker can also change the salary range.

4.3.11 Cache Management

At the end of search process, job results are stored in the “cache” table of MYSQL for a short duration. These stored jobs can be also displayed immediately to the job seeker who is looking for the same type of jobs again.

When a job seeker sends a request by integrated interface of meta-search engine, it is first observed that the required searched jobs are already available in the cache memory or not. If the required jobs are already available in the cache memory then jobs are displayed from the cache memory. But if the required jobs are not available in the cache memory then the whole process of job search starts from scratch. Accessing jobs from the cache memory is faster as compared to the search from scratch.

Meta-search engine can also keep the record of frequently searched competencies in the “frequency” table of MYSQL and can store the job results for these competencies in advance to save the time of a job seeker.

4.4 Ontology Development for Human Resource Management

We have developed an ontology that supports two different projects: a project on meta-search in job portals [DN06] and a competence management system at universities [DP07]. A university competence management system may use a meta-search engine to search for appropriate jobs for students. In both cases, the systems need a common language to enable the matching of profiles and preferences of the searching companies and students. Ontology for human resource management can define concepts i.e. job, competency and certain attributes for such concepts as well as relationship between these concepts. Thus, we may define that a certain job may require a specific type of competencies and to achieve this specific level of competence a particular knowledge and experience may be needed.

We have identified various requirements that ontology for human resource management must fulfil. It must contain the following:
• job related attributes and their synonyms for construction of meta-search interface
• matching job requirements and competency profiles of the candidate and vice versa
• synonyms of occupations and competencies
• relating occupations to required competencies
• concepts for personal attributes
• scale and measurement concepts for competencies and learning objects
• dependencies between competencies

The ontology is separated in sub-ontologies for occupations, competencies and learning objects to solve the problems in e-recruitment and competence management. For ontology modelling, we have used a Web Ontology Language (OWL) that supports richer expressiveness than RDF Schema. To query the competency and occupation sub-ontology, we have used a Simple Protocol and RDF Query Language (SPARQL) as explained in chapter 5. Our ontology is also used for the construction of user interface of meta-search engine (see Appendix A for our domain ontology in OWL). Our aim is to achieve all the benefits of semantic reasoning for competence management and e-recruitment by developing competency logic. Figure 4.18 explains how our ontology can be used in e-recruitment and university. Our ontology relates sub-ontologies such as competency sub-ontology and occupation sub-ontology. It supports matching of offers and demands for both the recruiters and job seekers. It can be used to find the semantic relationships, e.g. synonym and hypernym between job categories of various semantically different job portals [DN06].

### 4.4.1 Occupations Sub-Ontology

During integration of job portals, we have found that different job portals are representing the job categories in different ways. Some job portals contain different names to represent the job category. Some other job portals contain the generic concepts or specific concepts. For the unified interface of meta-search engine, we have collected job attributes from different job search engines and their corresponding attributes from HR-XML schemes into occupation sub-ontology.

We have integrated the computer and business related occupations into one meaningful and appropriate format from widespread used standards i.e. Standard Occupation Classification (SOC) and skills from International Co-operation Europe Ltd. This developed format is maintained in our domain ontology. Our domain ontology can be used to translate the concepts related to job descriptions and job categories from the job portals, according to the integrated format. The translated
concepts are used on the unified interface of job MSE. We have identified following attributes and relationships for the “Occupation” class of job ontology:

- “synonyms”, OWL data or object type property to keep the similar terms for occupations and job attributes.
- “related_skill”, OWL object type property maintains the relation between occupation and competency ontology.
- “belongs_to”, OWL object type property maintains the relations between competency and occupation schemes i.e. SOC or scheme by International Co-operation Europe Ltd’s. It is also used to maintain the relation between job attribute and HR-XML scheme from where the attribute has been selected [DNP07].

4.4.2 Competencies Sub-Ontology

We have considered two types of competencies: functional competencies and behavioural competencies as shown in figure 4.19. Functional competencies are expressed in terms of knowledge, skills and abilities. They are also called hard, job related or technical competencies. Behavioural competencies (also called soft competencies) define the personal behaviours or attitudes. An example of behavioural competency is leadership i.e. the ability to guide, motivate and influence personnel to meet the organization goals. We have focused on the computer and business related functional competencies. We have also identified different major, minor and sub-groups for functional and behavioural competencies in our competency sub-ontology. We have also maintained the abilities, skills and knowledge connected with a job that someone requires to perform a job effectively.

In OWL, “attributes” of classes (concepts) are represented by OWL data type properties and “relationships” between classes are represented by OWL object type properties. We need to represent how a certain competency is related to other concepts as well as we need attributes that are used to store certain values of an individual person. We have identified following attributes and relationships to define a competency in our ontology:

- “has_knowledge_level”, OWL object type property to record the knowledge-level of the competency.
- “has_experience_level”, OWL object type property to record the experience-level of the competency. It has further three sub-properties i.e. `has_computer_experience_level`, `has_business_experience_level` and `has_behavioural_experience_level`.
- “has_grade” OWL object type property to record the satisfactory level for knowledge and experience levels. It has further two sub-properties `has_experience_grade` and `has_knowledge_grade`.
- “synonyms”, OWL data or object type property to maintain synonyms of competencies.
- “requires”, OWL object type property to describe that one competency B is necessary to reach a grade of another competency A.
- “influences”, OWL object type property to describe that one competency B influences the grade of another competency A. B is not necessary to reach a grade of competency A.
“has_evidences”, OWL object type property to describe the evidences for a competency of an individual person.

Competencies at experience level have sub properties but not at the knowledge level because both of them are evaluated in different ways. Experience level determination varies from subject to subject. Knowledge, experience and grade levels are given in more detail in section 4.4.3. We have also identified that there is a part-type relationship between competencies. For example, C++ is a part-of Object Oriented Programming [DNP07].

4.4.3 Levels and Grades for Competencies

Competencies are associated with competency levels to describe different degrees of an abstract competency type. For our ontology, we have defined levels to measure computer science, business, and behavioural competencies as shown in Table 4.4.

Table 4.4: Levels to Measure Competencies

<table>
<thead>
<tr>
<th>Competencies</th>
<th>Knowledge Level</th>
<th>Experience Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer Science</td>
<td>Elementary, Intermediate, Advanced</td>
<td>Basic, Advanced</td>
</tr>
<tr>
<td>Business</td>
<td>Beginner, Competent, Professional</td>
<td></td>
</tr>
<tr>
<td>Behavioral</td>
<td>Basic, Professional, Corporate</td>
<td></td>
</tr>
</tbody>
</table>

For some computer science skills i.e. programming, we have added one more Intermediate experience level. It is to be noted that level determination is not based upon time in the job and few competencies have influence on others. For example, communication competency has influence on presentation competency and vice versa. Written communication skills have effect on the text of slides and oral communication skills have effect on the talk during presentation. Competencies can be unidirectional or bidirectional dependent.

We have distinguished grades for knowledge and experience level. Grades value can be between 0 and 1. Grades for levels can also be interpreted as satisfactory level. For example, we can say that if a person has an intermediate knowledge level programming skills then grade assigned is 0.5 and if a person has advanced experience level programming skills then grade assigned is 1. The competency of a grade issuing authority (expert/professor) issuing the grade, must be at least one level higher than the
competency of the person to be evaluated. For example, the presentation technique can be evaluated by a computer science professor/teacher at the lowest level. Higher level experts in communication training must be the evaluators of presentation skills. Thus a professor at a computer science department may evaluate programming skills on a higher level than a professor in a business management department.

Moreover, if one person has the highest grade issued by a secondary school authority and another has the highest grade issued by a university authority, then the representation must state that a person with university grade is at a higher competency level than the person having a secondary school grade [DNP07].
CHAPTER 5
IMPLEMENTATION

The following chapter discusses the technologies i.e. Java servlets, Jena Framework, MySQL Java connector, MySQL, Tomcat, Protégé, Mozilla XForm plugin, GeoLite country and Java Document Object Model API used for the implementation of meta-search engine in human resource domain.

5.1 Meta-Search Architecture from Implementation View

Job meta-search interface is a java servlet available to the meta-search provider for the creation of meta-search engine or to the job seeker for job search. Figures 5.1, represents meta-search architecture from implementation view with different technologies used in meta-search engine development.

Our meta-search engine uses the tomcat for the generation of meta-search query interface. During query interface generation and information extraction, Jena framework is used to communicate with domain ontology developed with Protégé-OWL editor. MySQL Connector/J 3.1 is used for storing/retrieving extracted jobs from various Web pages in/from MYSQL database. GeoLite country helps to determine the country of job search engine or meta-search engine’s visitor. Meta-search query interface generated by our system is a XForm that is displayed with the help of Mozilla XForm plug-in.

Below is a description of each technology that is used in the development of meta-search engine:

![Figure 5.1: Meta-Search Architecture from Implementation View](image-url)
5.2 Java Servlets

Server programs are responsible of receiving request, processing and returning a response to the Web server. There exist different server-side technologies i.e. CGI, ISAPI and NSAPI, ASP and Java servlets.

We have used java servlets for our job meta-search application. Java servlets are server-side programs that run on a Web server and build Web pages on the fly. “Java servlets is same as non-visual applet that runs on the Web server. It has life cycle similar to that of an applet and runs inside a Java Virtual Machine (JVM) [DHM+00]” Java servlet services HTTP requests and returns results as HTTP responses. In other words java servlets provide a way for a server side code to communicate with Web-based clients. The basic life cycle of a servlet is as follows (also shown in the figure 5.2).

- A server loads and initializes the servlet. Server runs the servlet's `init()` method.
- The servlet handles zero or more client’s requests.
- The server removes the servlet. Servlets run until the server destroys them. When a server destroys a servlet, the server runs the servlet's `destroy()` method. Some servers do this step only when they shut down.

Web pages built by java servlets can be based on data submitted by the user. The data on these Web pages may change frequently or may use information from corporate databases. We can move java servlets from one platform to another that supports Java. Java Servlets written for I-Planet Enterpise Server can run virtually on Apache, Microsoft IIS, or WebStar without any changes. Java servlets are efficient and support multiple users to simultaneously access the same servlet without bringing down the server. Java servlets can access all the Java APIs and are re-useable.

![Figure 5.2: Basic Servlet Life Cycle](image)

Java Servlet Development Kit (JSDK) consists of servlet API and a servlet engine. “Servlet API provides a general model of a class which performs a service. Servlet engine, saves the servlet programmer from having to worry about the details of connecting to the network, caching requests, and producing correctly formatted responses” [DHM+00]
5.3 Jena Framework

We have used Jena framework for querying and using domain ontology in our meta-search application. “Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine. Jena is open source and its framework includes a) A RDF API b) Reading and writing RDF in RDF/XML, N3 and N-Triples c) An OWL API d) In-memory and persistent storage e) SPARQL query engine [Jena07]”.

A Jena framework supports us in the integration of different schemas and query interface generation for meta-search engine. It also helps us in the identification of job results from the result pages and conversion of different result formats into a single format with the help of SPARQL query engine and domain ontology.

5.4 MySQL Java Connector

“MySQL Connector/J is a native Java driver that converts JDBC (Java Database Connectivity) calls into the network protocol used by the MySQL database. It lets developers working with the Java programming language easily, build programs and applets that interact with MySQL and connect all corporate data, even in a heterogeneous environment. MySQL Connector/J is a Type IV JDBC driver and has a complete JDBC feature set that supports the capabilities of MySQL [MysqlCJ07].

We have used MySQL Connector/J 3.1 to store/retrieve extracted jobs from various Web pages in/from MYSQL database. According to [MysqlCJ07] the latest production version of the driver is 50-100 percent faster in most situations than the previous version.

5.5 MySQL

An application that needs some form storage and retrieval of data may use relational databases i.e. MySQL. MYSQL is an open source Relational Database Management System (RDBMS) that enables a user to create, maintain and manage electronic databases. MySQL is cheap, easy to install, easy to use, lightweight, fast, robust and scalable database that supports good features. MySQL works on many different platforms and is extremely portable between platforms. MySQL uses a Structured Query Language (SQL) to store, access, delete, update and process data in a database. SQL functions are implemented using a highly optimized class library and should be as fast as possible. MySQL database is available as command-line tool as well as GUI tool for administration purposes. MySQL handles large databases and handle multiple user connections in a better way. It is better suited to Web applications as well [JGT99] [JC04].

We have used MySQL for job meta-search engine to store extracted jobs from various search engines. Following are the list of tables created in MySQL to support job meta-search engine:

- “job_results” to store extracted jobs i.e. job_title, type_of_hour, salary, job description etc.
• “meta_country” to store country name, currency, currency symbol, currency rate and language of country.
• “salary_range” to store hourly, weekly, monthly and yearly salary ranges.
• “cache_table” to store old job results with keyword, creation date and name of table where job results are stored.
• “frequency_table” to store keyword and their usage frequency to store commonly used job keywords.

5.6 Tomcat

“Tomcat is a Java servlet container and Web server from the Jakarta project of the Apache Software Foundation (http://jakarta.apache.org)”.

Tomcat provides an implementation of both servlet and the JavaServer Pages (JSP) specifications. It also provides an environment for Java code to run in cooperation with a Web server. The Tomcat servlet engine can be used in combination with an Apache Web server or other Web servers. Because Tomcat includes its own HTTP server internally, it is also considered a standalone Web server.

Tomcat is written in Java and runs on any operating system that has a JVM. Tomcat runs as a Windows service or Linux or Unix Daemon. By default it connects with a port 8080. Tomcat service has at least one connector and container in which an engine such as Catalina provides a service. Tomcat 5.x implements the Servlet 2.4 and JSP 2.0 specifications. It also reduced garbage collection and improved performance and scalability [JI03] [Tomcat07].

Our job meta-search application runs on Tomcat on Windows XP.

5.7 Protégé

Protégé is an ontology editor for creating and editing ontologies and knowledge bases. “Protégé implements a rich set of knowledge-modelling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. The Protégé platform supports two main ways of modelling ontologies via the Protégé-Frames and Protégé-OWL editors”. Protégé ontologies can be exported into a variety of formats including RDF(S), OWL, and XML Schema” [Protege07].

It is an open source, extensible, platform-independent environment that can run on MS Windows (NT/2000/XP), the common versions of Unix including Linux, Solaris and on the Mac OS X. Protégé is extensible based on Java and provides a plug-and-play environment that makes it a flexible base for rapid prototyping and application development. It is scalable and provides an intuitive and easy-to-use graphical user interface [Protege07]

To develop domain ontology for job meta-search engine, we have used Protégé-OWL editor that is an extension of Protégé and supports the Web Ontology Language.

5.8 GeoLite Country

GeoIP is the proprietary technology that uses MaxMind's IP geolocation data and services to determine geographical and other information about Internet visitors in
real-time. GeoIP can determine the country, region, city, postal code, area code, longitude/latitude, connection speed, ISP, company name etc of the Web site visitor with the help of series of complex algorithms. In these days such type of technologies are used for delivering customized content, targeted ads, Web log statistics, digital rights management, ad-serving, fraud screening, Web analytics and firewall/spam protection [Max07]. GeoLite Country is similar to the GeoIP Country database, but is slightly less accurate. We have used GeoLite Country instead of GeoIP because it is freely available.

Different search engines represent job description i.e. currencies or salary ranges for the job salary field in different formats. For example one job search engine may show job salary in Euro currency and other may in Dollars. One search engine may show salary on yearly basis and other in monthly. It is required to convert salary currency or salary ranges from various search engines into a uniform format. To convert currency to a uniform format, we need to know the currency of the salary from each job search engine. But some job search engines do not mention currency, so we have to detect currency by country. We used GeoLite Country to detect the country of individual search engines.

It is claimed at http://www.maxmind.com/app/geolitecountry, that accuracy of GeoLite Country is over 98% and it is updated monthly at the beginning of each month.

5.9 Mozilla Xform Plugin

Integrated meta-search query interface generated by meta-search engine is XForm that separates data and logic from presentation. Clear architecture of XForms makes application more robust, scalable, faster and secure. There exists plug-ins for Windows Internet Explorer and Firefox that add XForms support to the browsers. We have processed the meta-search integrated XForm in our Mozilla Firefox browser with the help of Mozilla XForm plugin (https://addons.mozilla.org/en-US/firefox/addon/824).

“Mozilla XForms project is to implement the W3C XForms 1.1 Candidate recommendation in Mozilla as an extension. The extension aims to work with all Mozilla projects building on the Mozilla platform, like Firefox, SeaMonkey, XULRunner, etc [MozXForm08]. There also exist Server-side solutions like Chiba that are good for deployment [Chiba07].

5.10 Document Object Model (DOM) API

“The Document Object Model (DOM) is an application programming interface (API) for valid HTML and well-formed XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated...... In the DOM, documents have a logical structure which is very much like a tree; to be more precise, which is like a "forest" or "grove", which can contain more than one tree. Each document contains zero or one doctype nodes, one root element node, and zero or more comments or processing instructions; the root element serves as the root of the element tree for the document [DOM00]”.

Java Document Object Model Level 2 Core API is a java based implementation of DOM API that is used for XML processing. Java DOM API allows programs to
dynamically access and updates the content and the structure of documents [JavaDOM].

We have used Java DOM API for implementation of various components of job meta-search engine.
CHAPTER 6
EVALUATION

For the evaluation of our meta-search engine, we have selected job search engines of different countries given in table A.1 (See Appendix A).

6.1 Case Study

Examples 6.1.1 and 6.1.2 are related to two search engines in the jobs domain (http://www.jobs.net and http://www.mymatrixjobs.com) and this domain is used to show how our schema/data matching and integration process operates for meta-search query interface generation and query processing. The URLs of possible job search engines are given as input to our system, and the GUI of the job meta-search engine is automatically generated. All schema and data mappings are detected with the use of the techniques described in chapter 4. An integrated schema is generated on the basis of these mappings and stored in xml format for further use in GUI (XForm) generation and query dispatching phase.

We have also used our domain ontology in the job domain described in chapter 4. Our domain ontology contains different attributes and data values for attributes in the job domain. For example, the data values for the attribute “Type_of_Hour” are {Contract, Full_Time, Internship, Part_Time, Permanent, Student, Temporary, Voluntary}. The attribute “Occupation” has multiple sub-classes, and the “Computer_Science” sub-class has data values {Software_Engineer, Administrator, Multimedia_Designer, System_Specialist etc.}.

6.1.1 Query Interface Generation for the Job Meta-search Engine

Each source job search engine has a different interface and job search criteria. For simplicity, we describe here just a fragment of our case study, and consider just two simple schemas from the full set of job search engines used in the case study. We also consider only a subset of the attributes and data values from these schemas.

S1 is the schema for search engine http://www.jobs.net, and contains attributes “Enter Keywords(s)”, “Enter a City”, “Select a State”, “Select a Category”, “Employment Type”. The “Select a Category attribute” has data values {Business Development, General Business, Information Technology, Science, Telecommunications, Design}. The “Employment Type” attribute has data values {Full-Time, Part-Time, Contractor, Intern}.

S2 is the schema for search engine https://www.mymatrixjobs.com and contains attributes “Keywords”, “City or Zip”, “States”, and “Job Type” which has data values {Contract or Permanent, Contract, Permanent}.

Our job domain ontology OG contains a class “Job attributes” with sub-classes “Competency”, “City”, “State”, “Job Category”, and “Type of Hour”. “Job Category” has multiple synonyms, and data values {Computer science, Business, Engineering, Telecommunication, Web Design etc.}, along with synonyms for each one of these. “Type of Hour” has synonyms “Employment Type” and “Job Type”, and data values {Contract, Full-time, Internship, Part-time, Permanent, Student, Temporary, Voluntary}, along with their individual synonyms. In OG there is a class-subclass relationship between each job attribute and its set of data values.
When the schema/data matching process starts, the first S₁ is matched with O𝐆. By applying a combination of matchers as described in the previous section, the following schema-level mappings are discovered:

\[
\begin{align*}
S₁.\text{Enter Keyword(s)} & \rightarrow O𝐆.\text{Competency} \\
S₁.\text{Enter a City} & \rightarrow O𝐆.\text{City} \\
S₁.\text{Enter a State} & \rightarrow O𝐆.\text{State} \\
S₁.\text{Select a Category} & \rightarrow O𝐆.\text{Job Category} \\
S₁.\text{Employment Type} & \rightarrow O𝐆.\text{Type of Hour}
\end{align*}
\]

As we use a top-down structural matching approach, when schema-level concepts are successfully matched, then data-level matching starts. At the data level, the following mappings are discovered:

\[
\begin{align*}
S₁.\text{Business Development} & \rightarrow O𝐆.\text{Business} \\
S₁.\text{General Business} & \rightarrow O𝐆.\text{Business} \\
S₁.\text{Information Technology} & \rightarrow O𝐆.\text{Computer Science} \\
S₁.\text{Science} & \rightarrow O𝐆.\text{Computer Science} \\
S₁.\text{Telecommunications} & \rightarrow O𝐆.\text{Telecommunication} \\
S₁.\text{Design} & \rightarrow O𝐆.\text{Web Design} \\
S₁.\text{Full-Time} & \rightarrow O𝐆.\text{Full-time} \\
S₁.\text{Part-Time} & \rightarrow O𝐆.\text{Part-Time} \\
S₁.\text{Contractor} & \rightarrow O𝐆.\text{Contract} \\
S₁.\text{Intern} & \rightarrow O𝐆.\text{Internship}
\end{align*}
\]

Next, S₂ is matched with O𝐆 and the following matchings are discovered at the schema level:

\[
\begin{align*}
S₂.\text{Keywords} & \rightarrow O𝐆.\text{Competency} \\
S₂.\text{City or Zip} & \rightarrow O𝐆.\text{City} \\
S₂.\text{States} & \rightarrow O𝐆.\text{State} \\
S₂.\text{Job Type} & \rightarrow O𝐆.\text{Type of Hour}
\end{align*}
\]

and at the data level:

\[
\begin{align*}
S₂.\text{Contract or Permanent} & \rightarrow O𝐆.\text{Contract} \\
S₂.\text{Contract or Permanent} & \rightarrow O Gdańsk.\text{Permanent} \\
S₂.\text{Contract} & \rightarrow O Gdańsk.\text{Contract}
\end{align*}
\]
From these mappings, schema attributes and data values, an integrated job search schema, $S_{MSE}$ for the meta-search query interface is then generated. This consists of attributes Competency, City, State, Job Category and Type of Hour. Attribute Job Category has data values \{Business, Computer science, Telecommunication, Web Design\} and Type of Hour has data values \{Full-time, Part-Time, Contract, Internship, Permanent\}. Finally, a GUI is generated from $S_{MSE}$ for the job meta-search engine.

### 6.1.2 Query Processing by the Job Meta-search Engine

A job seeker can pose a query from the integrated meta-search interface GUI, in terms of the integrated schema $S_{MSE}$. This query is rewritten by the meta-search engine, to target every search engine involved in the meta-search engine generation process. For example, suppose a job seeker poses a query $Q_{MSE}$ requesting all Contract jobs with keyword “java” in the computer science field:

$$Q_{MSE} : \text{Jobs (Competency=Java, Job Category=Computer science, Type of Hour=Contract)}$$

$Q_{MSE}$ query is transformed using the earlier discovered mappings both at the schema and the data level, to target each individual search engine. So we have queries $Q_{11}$ and $Q_{12}$ below targeted at http://www.jobs.net and queries $Q_{21}$ and $Q_{22}$ targeted at https://www.mymatrixjobs.com:

$$Q_{11} : \text{Jobs (Enter Keyword(s)=java, Select a Category=Information technology, Employment Type = Contractor)}$$

$$Q_{12} : \text{Jobs (Enter Keyword(s)=java, Select a Category=Science, Employment Type=Contractor)}$$

$$Q_{21} : \text{Jobs (Keywords=Java, Job Type = Contract)}$$

$$Q_{22} : \text{Jobs (Keywords=Java, Job Type = Contract or Permanent)}$$

$Q_{11}, Q_{12}, Q_{21}, Q_{22}$ are then submitted to the two search engines, the results are extracted by the Information Extractor component of our meta-search engine architecture, and are then merged, ranked (according to the preferences of the information seeker) and returned to the job seeker.

### 6.2 Evaluation of Processes

Different components in the meta-search creation and usage processes are evaluated as below:

#### 6.2.1 Evaluation of Job Meta-search Engine Creation Process

In this section, we show results collected during evaluation of different components of job meta-search engine that are involved in the job meta-search creation process. Figures 6.1-a and 6.1-b represent the interfaces from two job search engines. Preference collector component of meta-search engine has been created to get the preferences of meta-search provider i.e. country and job category for which meta-
search engine is required. As soon as meta-search provider enters the preferences, the meta-search engine creation process starts. The search engine selector component selects the job portals from the already stored list of portals that meet the meta-search provider’s preferences and shows them to the meta-search provider. At this stage, meta-search provider still has the facility to choose the search engines from the list of job search engines for job meta-search engine creation.

The evaluation of main components involved in job meta-search engine creation process is given below:

### 6.2.1.1 Evaluation of Interface Extractor

The interface extractor component of meta-search engine extracts the target form from job search interface page and ignores the other forms, advertisements and extra details of the HTML page. Figure 6.2 shows the extracted interface from job search engine.

![Figure 6.2: Extracted Interface of www.jobs.net](image-url)
http://www.jobs.net with the help of interface extractor component. We can see in Figure 6.2 that every pair of job attribute and input element is grouped together by LEX (layout-expression-based extraction) technique then is separated by HTML line. In the same way, interfaces of all the selected job search engines are extracted and stored.

For evaluation of interface extractor component, we evaluate LEX technique applied for grouping of labels and elements. We also evaluate the relationship type of elements. Table 6.1, shows the results of experiments for the evaluation of LEX technique. Column 2 of table 6.1 shows the manually identified labels from 21 job search interfaces. Column 3 shows the extracted and correctly grouped labels with elements by using LEX technique of interface extractor component. The experiments show that LEX technique is 86.2% accurate in grouping of labels and elements.

Table 6.1: Evaluation of LEX Technique

<table>
<thead>
<tr>
<th>No. of Job Search Engines</th>
<th>Total Labels (Manual)</th>
<th>Extracted Labels with LEX</th>
<th>% of LEX Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>116</td>
<td>100</td>
<td>86.2%</td>
</tr>
</tbody>
</table>

For the extracted attributes, it is important to evaluate the relationship type. Relationship type can be group, range or part. Table 6.2 shows, the results of experiments for the evaluation of relationship type elements.

Group type elements represent elements that can have multiple elements i.e. check boxes, radio buttons, or text boxes. Column 2 in table 6.2 represents, manually identified group type elements from search engine’s interfaces. Column 3 represents the number of group type elements extracted and identified by interface extractor component.

Range type elements represent the elements whose labels contain some keywords or patterns e.g. “between”, “from”, “to”. Column 5 in table 6.2 represents, manually identified range type elements from search engine’s interfaces and column 6 represents the number of range type elements identified and extracted by interface extractor component.

Part type elements are other than group and range type elements and they can have a single element i.e. single check-box or a select list etc. Column 8 in table 6.2 represents, manually identified part type elements from search engine’s interfaces and column 9 represents the number of part type elements identified by interface extractor component.

Table 6.2: Evaluation of Relationship Type of Elements

<table>
<thead>
<tr>
<th>No. of Search Engines</th>
<th>Total Group Type Elements (Manual)</th>
<th>Extracted Group Type Elements</th>
<th>% of Extracted Group Type Elements</th>
<th>Total Range Type Elements (Manual)</th>
<th>Extracted Range Type Elements</th>
<th>% of Extracted Range Type Elements</th>
<th>Total Part Type Elements (Manual)</th>
<th>Extracted Part Type Elements</th>
<th>% of Extracted Part Type Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>18</td>
<td>13</td>
<td>72.2%</td>
<td>1</td>
<td>1</td>
<td>100.0%</td>
<td>97</td>
<td>86</td>
<td>88.7%</td>
</tr>
</tbody>
</table>
Experiments show that interface extractor is 72.2% accurate in the identification of group type elements, 100% accurate in the identification of range type elements and 88.7% accurate in the identification of part type elements.

From 23 job search engines, interface extractor fails to extract interface for two search engines because one interface does not contain any label but consists of only single text box with search button. Another interface contains labels that are represented by images and interface extractor component cannot read text from images. Interface extractor component also fails to extract a type of interface for a search portal that contains frames such as http://www.jobspk.com.

6.2.1.2 Evaluation of XML-Schema Generator

XML-Schema generator component of meta-search engine generates an XML-Schema for the extracted interface to make it machine readable and understandable. Listing 6.1 shows an XML Schema generated by XML Schema generator component of meta-search for www.jobs.net. In the same way, XML Schemas for all the selected job search engines are created and stored for further use.


```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- SCHEMA IS FOR WEBSITE::http://www.jobs.net/ -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="RootJob">
    <xs:complexType>
      <xs:sequence>
        <xs:element id="qskwd" name="enter_keywords" type="xs:string"/>
        <xs:element id="qscty" name="enter_a_city" type="xs:string"/>
        <xs:element id="qssts" name="select_a_state" default="-all united states-">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration id="all,us" value="-all united states-"/>
              <xs:enumeration id="al,us" value="alabama"/>
              <xs:enumeration id="ak,us" value="alaska"/>
              <xs:enumeration id="wy,us" value="wyoming"/>
              <xs:enumeration id="wv,us" value="west virginia"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element id="qsjbt" name="select_a_category" default="-all job categories-">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration id="all" value="-all job categories-"/>
              <xs:enumeration id="jn001" value="accounting"/>
              <xs:enumeration id="jn002" value="admin &amp; clerical"/>
              <xs:enumeration id="jn054" value="automotive"/>
              <xs:enumeration id="jn038" value="banking"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
For the evaluation of XML-Schema generator component, we have focused on simple and complex type elements of XML Schema. We have also checked that values for the simple and complex type elements are correctly extracted or not. Table 6.3 shows the results of experiments for the evaluation of XML-Schema generator.

Simple type XML elements include all those elements that can have a single value i.e. single input-box, select element without multiple attribute and radio elements. Column 2 in table 6.3 represents manually identified simple elements at the search engine’s interface and column 3 represents the number of simple XML Schema elements generated by XML Schema generator.

Complex type XML elements include all those elements that can have multiple values i.e. select-box with multiple attributes that support multiple selections, multiple checkboxes and any type of other multiple elements. Column 5 in table 6.3 represents
manually identified complex type elements at the search engine’s interface and column 6 represents the number of complex elements identified by XML Schema generator.

The experiments show that XML Schema generator component is 89.4% accurate in generation of the simple elements and 96.0% accurate in the generation of complex elements.

For the evaluation of values for XML Schema elements, it is checked that whether XML Schema generator extracts correct values for simple and complex type elements or not. Column 8 in table 6.3 shows the number of elements of XML Schema whose values are not correctly identified. The experiments show that percentage of error is 8.6% in the extraction of values for schema elements and the major reason of error is the presence of JavaScript or VBScript in HTML interfaces.

Table 6.3: Evaluation of XML-Schema Generator

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>66</td>
<td>59</td>
<td>89.4%</td>
<td>25</td>
<td>24</td>
<td>96.0%</td>
<td>10</td>
<td>8.6%</td>
</tr>
</tbody>
</table>

From 21 search engine’s interfaces, XML Schema generator fails to generate XML-Schema for three interfaces because of JavaScript in HTML search interfaces.

6.2.1.3 Evaluation of Query Interface Generator

Query interface generator component of meta-search engine uses generated XML Schemas and transforms each schema into HR-XML-conform schema by explicit translation rules. We see in table 6.4 (column 1 and column 2) that search engines (jobs.net and mymatrixjobs) are using different representation concepts. Domain-ontology and multiple matchers are used for translation between concepts to HR-XML Schema (column 3). Difference in data structures and different granularities of knowledge is also handled by domain ontology and multiple matchers.

Table 6.4: Concept Translation to HR-XML by Domain Ontology and Multiple Matchers

<table>
<thead>
<tr>
<th>Jobs.net</th>
<th>Mymatrixjobs.com</th>
<th>HR-XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enter Keyword(s)</td>
<td>Keywords</td>
<td>Competency</td>
</tr>
<tr>
<td>Enter a city</td>
<td>City or Zip</td>
<td>City</td>
</tr>
<tr>
<td>Select a State</td>
<td>States</td>
<td>State</td>
</tr>
<tr>
<td>Select a Category</td>
<td>-</td>
<td>Job Category</td>
</tr>
<tr>
<td>Jobs Posted Within</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Employment Type</td>
<td>Job Type</td>
<td>Type of Hour</td>
</tr>
</tbody>
</table>

Translations between concepts shown in table 6.4 are stored in integrated XML Schema as shown in listing 6.2 by job meta-search creation process.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<MetaSearchEngine>
  <JSE1>
    <URL>http://www.jobs.net/</URL>
    <Method>GET</Method>
    <Competency>qskwd</Competency>
    <State>qssts</State>
    <City>qscty</City>
    <Job_Category id="qsjbt">
      <business>
        business development, general business
        <accounting>accounting</accounting>
        <banking>banking</banking>
        <marketing>marketing</marketing>
        <finance>finance</finance>
        <auditing/>
      </business>
      <computer_science>information technology, science</computer_science>
      <telecommunication>telecommunications</telecommunication>
      <internet/>
      <software_engineering>
        computer_science
      </software_engineering>
      <web_design>design</web_design>
      <engineering>
      </engineering>
    </Job_Category>
    <Type_Of_Hour id="qsetd">
      <Full-Time>full-time</Full-Time>
      <Internship>intern</Internship>
      <Contract>contractor</Contract>
      <Part-Time>part-time</Part-Time>
      <Permanent/>
    </Type_Of_Hour>
  </JSE1>
  <JSE2>
    <URL>https://www.mymatrixjobs.com/candidate/Login.action</URL>
    <Method>POST</Method>
    <Competency>keywordtitle</Competency>
    <State>state</State>
    <City>location</City>
    <Type_Of_Hour id="jobtype">
      <Full-Time/>
      <Internship/>
      <Contract>contract,contract or permanent</Contract>
      <Part-Time/>
      <Permanent>permanent,contract or permanent</Permanent>
    </Type_Of_Hour>
  </JSE2>
</MetaSearchEngine>
```
Integrated XML Schema from listing 6.2 is used for the creation of job meta-search query interface (XForm). Figure 6.3 shows an integrated interface for meta-search engine from www.jobs.net and www.mymatrixjobs.com. Before dispatching query to individual job search engines, integrated XML Schema is used by query dispatcher component for transformation of query from job meta-search interface.

For simplicity, data values for only “job category” and “type of hour” are integrated. Data values for other job attributes can be handled in the same way.

### 6.2.2 Evaluation of Job Meta-search Engine Usage Process

In this section, we give results collected during the evaluation of different components of job meta-search engine that are involved in the job meta-search usage process.

#### 6.2.2.1 Evaluation of Query Dispatcher

We again considered the jobs.net and mymatrixjobs.com from the list selected of job search engines, for the evaluation of query dispatcher.

Figure 6.4, shows the transformation of query from job meta-search interface into four queries before dispatching it to two job search engines. For the sake of simplicity, we considered only three job attributes i.e. competency, job category and type of hour and ignored the other attributes from the query. For example, following query is posed by the job seeker from the job meta-search engine:

```
Competency=java
Job Category=Computer Science
Type of Hour=Contract
```
From the integrated XML Schema (Listing 6.2) it can be seen that “Computer Science” category from jobs.net is “Information Technology” and “Science”. We can see in figure 6.4 that following two transformations of query from job meta-search interface to individual job search engine queries take place for www.jobs.net.

\[
\begin{align*}
\text{Competency}=\text{java} & \rightarrow Q\text{SKWD}=\text{java} \\
\text{Job Category}=\text{Computer Science} & \rightarrow Q\text{SBJT}=\text{Information Technology} \\
\text{Type of Hour}=\text{Contract} & \rightarrow Q\text{SETD}=\text{Contractor}
\end{align*}
\]

and

\[
\begin{align*}
\text{Competency}=\text{java} & \rightarrow Q\text{SKWD}=\text{java} \\
\text{Job Category}=\text{Computer Science} & \rightarrow Q\text{SBJT}=\text{Science} \\
\text{Type of Hour}=\text{Contract} & \rightarrow Q\text{SETD}=\text{Contractor}
\end{align*}
\]

Integrated XML Schema for jobs.net and jymatrixjobs.com (Listing 6.2) is used for the query transformation. In the same way, again two transformations of query from job meta-search interface to individual job search engine queries take place for www.mymatrixjobs.com. Finally, query dispatcher constructs the complete queries for each selected job search engine and dispatches them to the individual job search engines. When more job attributes are involved, it is possible that during transformation of query from meta-search engine to multiple search engines, some job search results are lost.
6.2.2.2 Evaluation of Information Extractor

After query is sent to multiple search engines, the information extractor component of meta-search engine is activated and multiple result pages are downloaded. Figure 6.5 shows a result page collected by information extractor component when query is sent to www.jobs.net. For the evaluation of information extractor component, we evaluate record collector component and result collector component separately.

6.2.2.2.1 Evaluation of Record Collector

Upper part of figure 6.5 shows a result page which contains extra hyper-links and text and lower part of figure 6.5 shows a job record section. The upper section with extra hyperlinks and text is ignored and job record section is extracted with record collector component of meta-search engine as shown in figure 6.6-a. Next the required URLs and job titles from the record section are identified, so that detailed job description can be extracted. Figure 6.6-b shows a list of few URLs and job titles extracted from the target job record section. We used the regularities in HTML tag structures on the result pages for record collection phase and experiments show that this technique achieves high accuracy.

For the evaluation of record collector component, we have focused on the identification of total number of record sections from the full HTML result page and the URLs of jobs. Later on job titles are extracted from the identified record section. Table 6.5 shows the results of experiments for record collector evaluation and it can be seen that record collector component is 90.5% accurate in record section identification, 95.3% accurate in job’s URLs identification and 63.2% accurate in job’s titles identification.

<table>
<thead>
<tr>
<th>No. of Job Search Engines</th>
<th>No. of Correctly Identified Record Sections</th>
<th>% of Correctly Identified Record Section</th>
<th>No. of Jobs at Result Page (Manual)</th>
<th>Total No. of Identified Job’s URLs</th>
<th>% of Identified Jobs URLs</th>
<th>Total No. of Identified Job’s Titles (Manual)</th>
<th>% of Identified Job’s Titles</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>19</td>
<td>90.5%</td>
<td>277</td>
<td>264</td>
<td>95.3%</td>
<td>175</td>
<td>63.2%</td>
</tr>
</tbody>
</table>

It is noted that one of the reasons of incorrect identification of titles, URLs and record section is few number of records on the result page.

6.2.2.2.2 Evaluation of Result Collector

Result collector component visits the identified URLs, to collect the job details. Figure 6.7 (upper part) shows an HTML page with full job description that consists of job attributes i.e. Contact, Phone, Fax, Ref ID, Posted, Location, Base Pay, Employee Type, Industry, Manages Others, Job Type, Company, Responsibilities and Requirements etc. Domain ontology along with multiple matchers is used by the result collector component for the identification and collection of job details from a job page.

Figure 6.7 shows (lower part), that job attributes are identified, converted into a single format and then stored in a table as below:
The identification of other attributes is not shown in the figure 6.7 for the sake of simplicity.

For the evaluation of result collector component, we have focused on the attributes identified by domain ontology and transformed to HR-XML equivalent attributes. Table 6.6 shows the results of experiments for result collector evaluation and it can be seen that result collector is 77% accurate in semantic understanding of job attributes and then transforming them to HR-XML equivalent attributes.

Table 6.6: Evaluation of Result Collector

<table>
<thead>
<tr>
<th>No. of Job Search Engines</th>
<th>Total No. of Job Attributes at the Job’s Result Page (Manual)</th>
<th>Number of Attributes Identified by Domain Ontology</th>
<th>% of Attributes Identified by Domain Ontology</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>135</td>
<td>104</td>
<td>77.0%</td>
</tr>
</tbody>
</table>

It is observed that sometimes result collector component fails to identify the job related information with the help of domain ontology because of lack of proper labels at the result page. In this case multiple matchers are helpful in the semantic understanding of job description.

6.2.2.3 Evaluation of Result Ranker

Results from multiple search engines are merged together by the result merger component and stored in a MySQL database. It is possible that a same job is accessible by multiple search engines and meta-search stores duplicate job records. The duplicate result eliminator component can eliminate the duplicate jobs by the identification of same URLs or part of URLs as described in chapter 4 (section 4.3.8). After duplicates elimination the result ranker component is used.

Jobs collected from multiple search engines are ranked and displayed to the user. If user is interested to rank the jobs according to salary, then all the jobs having salary are considered. Sometimes job salary is not mentioned in the job description, so such types of jobs are ignored while ranking by salary. Result pages may contain salary values in text, semi-numeric or numeric form. The result pages with text salary values i.e. nil, unspecified, open or negotiable are ignored. Ranking of job according to salary needs identification of numeric i.e. 85000 and semi-numeric i.e. up to 75k base, 130-180k, or $ 50,000 to 60,000 /year etc salary values from the job result pages. After the identification of salary values, salaries are converted into a single currency format and range. We have focused on the identification of numeric and semi-numeric salary values from the job result pages. If currency cannot be detected from the job record then it is detected with the help of IP address and country. Some search engines show salary on daily basis and other on weekly, monthly or yearly basis. If it is not mentioned that salary is on daily, weekly, monthly or yearly basis then it is determined
with the help of “salary_range” table of MySQL maintained by our meta-search engine. Some regular expressions are also used to convert salary to a common format. Figure 6.8 shows a list of ranked jobs where job salary is converted into Euro currency and per year basis. Figure 6.8 shows ranking done on the minimum salary because for few jobs maximum salary is not given. The system also provides an option to the job seeker to rank the jobs according to the average of minimum and maximum salary.

Table 6.7 shows the experiments results for the evaluation of result ranker component. It can be seen in table 6.7 that result ranker is 100% accurate in the identification of numeric and semi-numeric salary values and their conversion to a single currency and format.

Table 6.7: Evaluation of Result Ranker

<table>
<thead>
<tr>
<th>No. of Job Result Pages with Salary</th>
<th>Total No. of Text Salary Values (Manual)</th>
<th>Total No. of Numeric and Semi-Numeric Salary Values (Manual)</th>
<th>Total No. of Identified Numeric and Semi-Numeric Salary Values by Result Ranker</th>
<th>% of Identified Numeric and Semi-Numeric Salary Values by Result Ranker</th>
</tr>
</thead>
<tbody>
<tr>
<td>175</td>
<td>80</td>
<td>95</td>
<td>95</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 6.7: Evaluation of Result Ranker
Fig. 6.6: a - Extracted Record Section from jobs.net

b- Extracted URLs and Job Titles from the Record Section of jobs.net
Figure 6.7: Identification of Job Attributes from jobs.net Result Page with the Help of Domain Ontology
6.3 Evaluation of Hybrid Approach in Schema Matching

We have evaluated our techniques in the job domain, using the following job search engines:
- http://www.careerbuilder.com
- http://www.learn4good.com/jobs/
- https://www.mymatrixjobs.com/candidate/Login.action
- http://www.jobs.net/
- http://jobsearch.monster.com/

Figure 6.9 shows the contributions of element-level, structure-level and ontology-based techniques in the matching process for each job search engine. We see that for the careerbuilder search engine, for example, our hybrid approach identifies a total of 6 job-related attributes, with element-level techniques identifying 3 attributes, structure-level techniques 1 attribute and ontology-based techniques 2 attributes. The results for the other search engines are shown in a similar way, and we can see the benefits of adopting our hybrid approach.

Combining the above results, we calculate an overall contribution to the identification of job-related attributes within all the search engine interfaces of 60.60% for element-level techniques, 15.15% for structure-level techniques, and 18.18% for ontology-based
techniques. When we combine all the techniques, our hybrid approach for meta-search engine generation achieves overall correctness of $60.60\% + 15.15\% + 18.18\% = 93.93\%$, where we define correctness as:

$$\text{number of attributes correctly identified over the set of search engine interfaces}$$

$$\text{total number of attributes in the set of search engine interfaces}$$

The precision achieved in this experiment was 100\% (all the attributes identified were correct) and the recall was 93.93\%. For this particular experiment, if the ordering of the groups a)-d) described in Section 4.3.4.4.5 of chapter 4 is altered, then the same overall set of matchings would be discovered, but this may not be the case in general i.e. different orderings of application of a)-d) may yield different sets of matchings.

### 6.4 Performance

The experiment for schema matching and integration for eight search engines in section 6.3, took 1 minute and 44 seconds for the job meta-search query interface generation process, on a machine with 1.60 GHz processor, 512 RAM and running Microsoft Windows XP. For two job search engines (http://www.jobs.net. and http://www.mymatrixjobs.com from case study), the same schema matching and integration process took 28 seconds.

The experiment for query dispatching, information extraction, result merging, duplicates elimination and result ranking for searching ‘java’ related jobs, from three job search engines took 17 seconds with 1.60 GHz processor, 512 RAM and running Microsoft Windows XP. The same query took 34 minutes when sent to 23 job search engines. It is to be noted that Internet downloading speed also affects the speed of meta-search engine.

### 6.5 Existing Problems and Required Improvements

Following are some problems with the existing system and improvements that can be done to improve the efficiency of our meta-search system:
• During interface extraction, sometimes system cannot identify the correct label from the search interface. More algorithms should be developed and applied to extract correct label for an element and attribute. Sometimes, interface labels are given as a text within an image. The interface extractor component cannot read text from images. So it cannot detect label from images. In such cases, optical character reader (OCR) can read text from images and could solve this problem.

• There must be a solution to extract interface from search portals that contain frames i.e. http://www.jobspk.com because it is not possible with the existing system to extract a form from these type of search portals.

• Existing interface extractor component cannot obtain values for a single-select list or multiple-select list, which dynamically obtain values according to the selection of user using JavaScript or VBScript. For example, in search engine jobshejobs.com, values of multiple select list for attribute label “or_province” cannot be obtained in advance because until and unless user selects a “job location”, list of province is not available. There must be a solution to handle these types of problems in future.

• During query transformation, it is possible that we can lose some search results. There must be a solution so that data loss is reduced to minimum during query translation.

• Record collector component of meta-search engine, extracts the record section on the basis of pattern format between the target-start-boundary marker and the target-end-boundary marker (as discussed in chapter 4). Pattern at the result page represents a job result. Record collector component can perfectly identify the record section if there is more number of patterns available on the result page. But if the result page contains less number of patterns available, record collector component cannot detect the right section for jobs extraction.

• We have used the regularities in HTML tag structures on the result pages for record section identification but use of visual content features i.e. font size, type and color can be further utilized to improve the record section identification.

• Existing meta-search engine can only visit and download the first result page for the extraction of job record and result section. It is possible that a search engine shows job results at multiple pages as shown in figure 6.10- a and figure 6.10- b. It is required that information extractor component can visit multiple result pages returned by the job search engine for extraction of jobs.

Figure 6.10-a: Part of Result Page from 6figurejobs.com

Figure 6.10-b: Part of Result Page from jobs.net
• Record collector component extracts URLs and titles of jobs and then result collector component visits the extracted URLs for job details. But if URL is constructed on the basis of JavaScript as in http://jobs.imdiversity.com/, then it cannot be visited by result extractor.

• The result collector component extracts the job attributes i.e. labels from the result page and semantically identifies the job attributes with the help of job ontology. Again sometimes it cannot identify the correct label from the result page therefore more matchers for label identification are required.

• Result identification with the help of ontology and few matchers return a good result but it is a slow process. For the first 8 search engines in Appendix A, it can identify the job results from 255 result pages in 29 minutes. As job seeker cannot wait for a long time for job results, so we maintained a cache table in MYSQL to keep at least 3 days old results. So that if job seeker is interested in the previously searched results, then results can be directly shown from cache table in few seconds. But if job seeker is looking for a kind of job that has not been searched before by the meta-search engine then he/she has to wait for a long time. To solve the above problem, we can introduce a crawler in future that crawls different job search engines weekly for all frequently searched job keywords, identifies the jobs and stores in meta-search database for the job seeker.

• Meta-search database may also contain the expired job results therefore it is required that all the expired jobs, must be automatically removed from the database. Different search engines show expiry dates in different formats. For example, for one search engine 1/12/2007 is 1st Dec, 2007 and for other it can be 12th January, 2007. So all the dates must be converted in a single format and then expired jobs must be automatically removed from the database.

• “GeoLite Country” is used in our meta-search engine to detect the country of the job seeker or job search engine with accuracy of over 98%. According to http://www.maxmind.com/app/geolitecountry, “GeoIP Country” can also help us in the detection of country of the job seeker or search engine with more than 99% accuracy.

• Existing meta-search engine can rank the job results only according to the salary. But user can be interested to rank the job results according to the nearest location or relevancy.
CHAPTER 7
CONCLUSION AND FUTURE WORK

This chapter consists of two parts. In the first part we conclude our work and second part gives some future research directions.

7.1 Conclusion

Search is an important and popular activity on the Internet but primary search tools i.e. search engines, subject directories and social network search engines are unable to provide the desired results to the user. Meta-search engine is an excellent choice for a specific topic search.

We have developed a meta-search engine in the human resource domain that can solve the problems of meta-search providers and job seekers. Our system can meet the various demands that modern meta-search providers and job seekers expect from a current online job market. We have identified two processes involved in meta-search engine a) Meta-search engine creation process and b) Meta-search engine usage process.

Meta-search creation process is a tricky and time consuming task especially at schema extraction, schema matching, and schema and data integration level. The contributions of this dissertation related to meta-search creation process are as follows:

- We have presented an approach for integrating data from different job portals in a meta-search in order to support job seeking people to master the large number of available job portals. Our main focus in this dissertation is on the automated extraction of the structure of provided information. This structure is used in meta-search to integrate the different sources. The main difference between our work and existing research work is that we used HR-XML for schema integration. Each scheme is translated to a HR-XML-conform schema. There is no published work available for integration of machine readable schemas and HR-XML Schema for meta-search engine in human resource management.

- We have developed a prototype system that uses a hybrid approach for schema and data matching and integration in meta-search engines. Our main focus has been on the schema/data matching and integration aspects of meta-search engine generation and usage. We have introduced a hybrid approach that leverages the cumulative work of the large research community in order to resolve schema/data matching and mapping problems between heterogeneous search interfaces. Our experiments in the job domain show that the combined use of element-level, structure-level and ontology-based techniques increases the correctness of matching during the automatic integration of the source search engine interfaces.

- Our techniques and results provide a contribution in the area of generating more comprehensive and more concise meta-search query interfaces, more accurate meta-search query processing, and more comprehensive and concise presentation of search results to users. Using domain ontology is advantageous if the data values of the ontology attributes are also modelled within the ontology. Data level matching can be undertaken with greater precision and, in our context of search engine interface integration, we have the added advantage of typically a limited set of data values for each attribute, as compared with the data values typically arising in a
database integration setting, for example. In our setting, we generally have fewer values to compare, and the matching takes less time and computational effort.

• This helps meta-search providers in automatic creation of meta-search engine according to their choice in a short time.

Job meta-search engine usage process is also tricky at jobs extraction and semantic understanding of job description. The contributions of this dissertation related to meta-search usage process are as follows:

• In meta-search usage process, again our hybrid approach uses cumulative work of the large research community together with the Semantic Web technologies in automatic extraction and semantic understanding of job description. This supports job seeking people to master the large number of available job portals.

• Integrated system is helpful for job seekers to navigate multiple search engines without visiting them separately and increases the Web coverage for them. Developed meta-search engine in human resource domain has ability to search the jobs from invisible Web too and increases the precision, recall and quality of job results.

• Job seekers do not have to spend their time to comb through large number of job results and long documents to find the relevant information.

• Developed meta-search engine facilitates job seekers with fast navigation of better quality results in the modern employment market and provides an overview of all job portals.

• Job ranking after converting all the salaries to a single currency and range format saves the job seeker from manual comparisons of jobs and helps in accessing better jobs.

• Caching of jobs in the system speeds up the response time of the requests from the job seekers.

This dissertation is an attempt to improve the automatic schema and data integration process of job search engines. It also helps in the semantic understanding of job descriptions from job search engines by using HR-XML, semantic Web technologies and different type of matchers. Other contributions of this dissertation are as follows:

• We have developed an ontology model with occupations, competencies and learning objects for e-recruitment and competence management systems [DNP07]. We identified some attributes for occupation and describe functional and behavioural competencies more thoroughly. Traditional schema and interface integration in the past was done without any attempt to understand the meaning of terms from individual schemas. Terms can have synonyms, different associations, interrelationships and can be used in many senses. Use of our modeled domain ontology and HR-XML for the generation of integrated schema and interface, help in understanding the meaning of terms and improve the quality of search interfaces.
• Due to use of domain ontology and HR-XML, our meta-search interface contains standardized attributes and their values so that job seekers can easily construct the job search criteria for multiple search engines.

• Integration of classification schemes for recruitment i.e. Standard Occupational Classification (SOC) System and skills from International Co-operation Europe, Ltd in domain ontology, provide a controlled vocabulary for job categories at meta-search interface.

• We have presented design patterns that can architect complex meta-search engine construction processes and provide us with flexible design philosophy. We have introduced design patterns for meta-search engine construction process and its important components. The reusable design patterns for meta-search components can be reused several times by not only in meta-search domain but also in some different application domain after some modifications. Design patterns for meta-search engines are flexible enough and have facility to add or remove features for different components with minimum effort in the future. These design patterns can speed up the development process for new developers as they don’t need to re-discover the design problems and can save their precious time. These design patterns for meta-search also enhance communication between the team members by providing them with shared vocabulary and a way to alter or extend some parts of the system independently of all other parts.

• Application of semantic Web technologies in the meta-search domain prove to be helpful and provide automatic and efficient schema integration, query translation, integrated query interface generation and result identification.

• Meta-search in human resource domain will be helpful in reducing the unemployment rate of a country.

7.2 Future Work

There is room for improvements in the existing job meta-search engine. This section proposes improvements in the existing system and some future research directions.

• We plan to investigate introducing further matchers and techniques.

• Schema matching and integration process can be improved by applying all the matchers on all the attributes from search engine’s interfaces. From all the discovered set of matches for a particular attribute, the system must choose the match with the highest confidence. Applying all the matching techniques on all the attributes will definitely take more time but better matching results can be produced. In this way, the system will also be able to act as a learning system that it can suggest the best matching technique after an application of some matchers.

• During schema matching and integration, we have applied matchers in groups in a particular order (see section 4.3.4.4.5 of chapter 4). But, if the order of the groups of matchers is altered, then it is possible that different orders yield different sets of matches. In future, we plan to check the effects of application of matchers in different order on i) cumulative set of matches discovered ii) proportion of matches
found by different type of techniques and iii) time taken to undertake the matching process.

- We will also try to introduce some benchmarks to be helpful in the comparison and evaluation of schema/data matching and integration systems.

- We will report on the query processing performance of meta-search engines.

- We will develop a GUI at schema integration level that gives an option to a user to select the semantic relationship from identified mappings.

- Meta-search must support a job agent that can help the job seeker in searching a job with a supply of available jobs dynamically at a Web service interface.

- Developed meta-search engine can be used in a business to generate revenue. There can be three parties involved for the generation of revenue. First, recruiters who help other companies, agencies and job search providers in search of potential candidates. Second, job seekers who are in search of better jobs. Third, banner advertisement firms who help other firms by providing advertisement services. Job meta-search engine can help the recruiters by registering them to have potential candidates. It can also help the job seekers to subscribe for different job offers. It may be beneficial and profitable for banner advertisement firms by providing Web space to these firms for advertisement.

- There can be many ways of generating revenue from a meta-search engine a) the recruiters and the companies pay for getting registered/subscribed to have potential candidates b) job seekers pay to get subscribed or for the registration to have suitable job offers c) the recruiters pay only if they find a potential candidate d) job seekers pay only if they find a suitable job e) banner advertisement firm pay only if they get banner advertisement services by the meta-search Website. Firms pay each time an advertisement is displayed on the Website. Firms also pay flat fee when visitors click on advertisement. There can be another option that firms pay on the basis of sales performance i.e. if a visitor clicks an advertisement and then purchases a product. The business model for meta-search engine seems to be workable by providing networking tasks.

- Our modelled ontology can be used for comparing competency profiles of certain individuals with those of expected competencies in job offers. It can also be used for finding differences between given goal profiles and those of certain individuals.

- Modern human resource management focuses more on competencies than on job titles or job positions. At the moment only few job portals reflect this trend. However, in the near future this will change and the detailed description of required competencies will gain impact. This can be modelled with HR-XML, too. If job offerings are based on the specification of required competencies and job applicants submit queries with their detailed competencies (possibly part of CV) then the matching will be more complex and fuzzy-based.
Technically, following improvements at the implementation level are required in the existing system:

- At this time, our job search engine selector component does not crawl the Web automatically for identification of job search engines but URLs of candidate source search engines to be made known to our system. In future, we plan to introduce a crawler that can crawl the Web automatically, identifies and categorizes the search engine’s interfaces and jobs pages of any company on the fly from the Web. Identification and categorization of Web pages can be done by using domain ontology and multiple matchers. Domain ontology can help in the semantic identification of job concepts and then accordingly categorizes the search interfaces and Web pages. In this way, not only search interfaces but Web pages of companies with few jobs will also be selected. In brief, we want to discover and categorize the sources on the fly from the Web for the meta-search engine construction.

- The system must support a crawler that crawls different job search engines weekly or bi-weekly for frequently searched job keywords, identifies the jobs and stores in meta-search database in advance for the job seekers.

- Labels representing job attributes at the search interfaces and at the job description pages play an important role in schema integration, interface integration and information extraction so more label extraction techniques can be developed. The existing system cannot read a label whose text is an image. So it is possible to integrate an optical character reader (OCR) in the system that can read text from images.

- At this time our system can handle textual data only at data integration level but in future we will work to handle numeric and range type data too.

- During schema and data integration process, all mappings for multiple numbers of job search engines are stored in the integrated XML Schema. But if the job search interface of any job search engine is updated or changed then all the stored mappings for that particular search engine will become invalid. In this situation, we have to re-generate the meta-search engine so that new mappings are stored in the integrated XML Schema.

- It is required in future that meta-search engine can also identify different regularities according to the geographic location. For example, in some countries extra salary is given at Christmas event and in other countries extra bonus is given at religious events.

- Query translation from meta-search engine to individual search engines must be improved so that maximum number of job results can be obtained from multiple search engines.

- The system must be able to handle frames, JavaScript and VBScript in the HTML search interfaces and result pages.
• Existing meta-search engine can only visit and download the first result page for the extraction of job records and result section. It is possible that a search engine shows job results at multiple pages. The system must be improved so that information extractor component can visit multiple result pages returned by the job search engine for the extraction of jobs.

• Information extractor component needs some development so that it can identify the record section even if there are less number of patterns i.e. job records. It must also use more visual content features i.e. font size, type and color to improve the record section identification.

• Result ranker component can be improved to rank the extracted jobs on the basis of nearest location or relevancy. Moreover, all the expired jobs must be removed automatically from the database of meta-search engine.

• It is required in future that meta-search engine can measure the user’s trust in a particular search engine. For example, at the end of search process, meta-search engine must ask the job seeker to enter the trust level. The trust level can be obtained from a user as a feedback which can be measured through the level of satisfaction for a particular search engine. It could tell us about the choice of different meta-search engines that which search engine is most suitable for a particular query.
### APPENDIX – A
### TABLES AND LISTINGS

Table A.1: List of Job Portals

<table>
<thead>
<tr>
<th>S.No</th>
<th>Name</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>can jobs</td>
<td><a href="http://www.canjobs.com/index.cfm">http://www.canjobs.com/index.cfm</a></td>
</tr>
<tr>
<td>3</td>
<td>Job Monkey</td>
<td><a href="http://www.jobmonkeyjobs.com/GetJobs.rs">http://www.jobmonkeyjobs.com/GetJobs.rs</a></td>
</tr>
<tr>
<td>4</td>
<td>6 figure job</td>
<td><a href="http://www.6figurejobs.com/ExecSearchJobs.cfm?CFID=7553183&amp;CFTOKEN=92122739&amp;noCache=718237&amp;am=20">http://www.6figurejobs.com/ExecSearchJobs.cfm?CFID=7553183&amp;CFTOKEN=92122739&amp;noCache=718237&amp;am=20</a></td>
</tr>
<tr>
<td>5</td>
<td>Jobshejobs</td>
<td><a href="http://www.jobshejobs.com/">http://www.jobshejobs.com/</a></td>
</tr>
<tr>
<td>7</td>
<td>Net-temps</td>
<td><a href="http://www.net-temps.com/findjobs.html">http://www.net-temps.com/findjobs.html</a></td>
</tr>
<tr>
<td>8</td>
<td>Monster</td>
<td><a href="http://jobsearch.monster.com/Search.aspx">http://jobsearch.monster.com/Search.aspx</a>?</td>
</tr>
<tr>
<td>10</td>
<td>WorkoPolis</td>
<td><a href="http://www.workopolis.com/">http://www.workopolis.com/</a></td>
</tr>
<tr>
<td>11</td>
<td>Jobs.net or CareerBuilder</td>
<td><a href="http://www.jobs.net/">http://www.jobs.net/</a></td>
</tr>
<tr>
<td>12</td>
<td>IT Calssifieds</td>
<td><a href="http://www.itclassifieds.com/jobsearch/jobsearch.asp">http://www.itclassifieds.com/jobsearch/jobsearch.asp</a></td>
</tr>
<tr>
<td>13</td>
<td>After College</td>
<td><a href="http://www.aftercollege.com/">http://www.aftercollege.com/</a></td>
</tr>
<tr>
<td>14</td>
<td>My Matrix</td>
<td><a href="https://www.mymatrixjobs.com/candidate/Login.action">https://www.mymatrixjobs.com/candidate/Login.action</a></td>
</tr>
<tr>
<td>15</td>
<td>Jobs Boston</td>
<td><a href="http://jobs.boston.com/careers/jobsearch">http://jobs.boston.com/careers/jobsearch</a></td>
</tr>
<tr>
<td>16</td>
<td>Chronicle</td>
<td><a href="http://chronicle.com/search/jobs/">http://chronicle.com/search/jobs/</a></td>
</tr>
<tr>
<td>17</td>
<td>Nation Job</td>
<td><a href="http://www.nationjob.com/education/search?prior=x0,p165&amp;aloc=1135&amp;asal=1173&amp;searchflag=1">http://www.nationjob.com/education/search?prior=x0,p165&amp;aloc=1135&amp;asal=1173&amp;searchflag=1</a></td>
</tr>
<tr>
<td>18</td>
<td>FlipDog</td>
<td><a href="http://www.flipdog.com/jobs/vienna/">http://www.flipdog.com/jobs/vienna/</a></td>
</tr>
<tr>
<td>19</td>
<td>JobServe</td>
<td><a href="http://www.jobserve.com/">http://www.jobserve.com/</a></td>
</tr>
<tr>
<td>20</td>
<td>Total Jobs</td>
<td><a href="http://www.totaljobs.com/">http://www.totaljobs.com/</a></td>
</tr>
<tr>
<td>21</td>
<td>Im Diversity</td>
<td><a href="http://jobs.imdiversity.com/jobseekerx/SearchJobsForm.asp">http://jobs.imdiversity.com/jobseekerx/SearchJobsForm.asp</a></td>
</tr>
<tr>
<td>22</td>
<td>Job Vertise</td>
<td><a href="http://www.jobvertise.com/search">http://www.jobvertise.com/search</a></td>
</tr>
<tr>
<td>23</td>
<td>Learn For good</td>
<td><a href="http://www.learn4good.com/jobs/">http://www.learn4good.com/jobs/</a></td>
</tr>
</tbody>
</table>
Listing A.1: Domain Ontology in OWL

```xml
<owl:Ontology rdf:about="#Ontology"/>

<!-- Domain Classes -->
<owl:Class rdf:about="#Domain"/>

<!-- Subclasses -->
<owl:Class rdf:about="#ComputerScience"/>
<owl:Class rdf:about="#SoftwareEngineering"/>
<owl:Class rdf:about="#BusinessManagement"/>

<!-- Properties -->
<owl:ObjectProperty rdf:about="#hasDegree"/>
<owl:ObjectProperty rdf:about="#hasLanguage"/>
<owl:DatatypeProperty rdf:about="#hasSalary"/>
```

118
Convert project specifications and statements of problems and procedures to detailed logical flow charts for coding into computer language. Develop and write computer programs to store, locate, and retrieve specific documents, data, and information. May program web sites.

(SOC) 15-1030 - Computer Science

Analyze, design, test, and evaluate network systems, such as local area networks (LAN), wide area networks (WAN), Internet, intranet, and other data communications systems. Perform network modeling, analysis, and planning. Research and recommend network and data communications hardware and software. Include telecommunications specialists who deal with the interfacing of computer and communications equipment. May supervise computer programmers.

(SOC) 15-1040 - Computer and Information Technology

(AO) 15-1050 - Information and Computer Security Analyst

(AO) 15-1060 - Database Administration

(AO) 15-1070 - Software Engineering

(AO) 15-1080 - Information Security Analyst

(AO) 15-1090 - Computer Network administrator

(AO) 15-1100 - Computer and Information Systems Managers
Listing A.2: Individual XML Schema for top-consultant.com

```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<!--URL=http://www.top-consultant.com/default1.asp?SID=GO&C=1&amp;KW=healthcare_consulting&amp;MT=Broad/ -->
<!-- METHOD=post -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="RootJob">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="firms" default="all firms" id="sel_firm">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="all firms"/>
              <xs:enumeration value="accenture"/>
              <xs:enumeration value="access ag"/>
              <xs:enumeration value="barracuda search"/>
              <xs:enumeration value="oracle"/>
              ...
              <xs:enumeration value="total quality institute"/>
              <xs:enumeration value="uk consulting jobs"/>
              <xs:enumeration value="vesterling consulting"/>
              <xs:enumeration value="xuccess consulting"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="location" default="uk: london &amp; se" id="sel_location">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="all locations"/>
              <xs:enumeration value="uk: london &amp; se"/>
              <xs:enumeration value="uk: outside of london"/>
              <xs:enumeration value="germany"/>
              <xs:enumeration value="united states"/>
              <xs:enumeration value="us: east coast"/>
              <xs:enumeration value="us: west coast"/>
              <xs:enumeration value="us: mid west"/>
              <xs:enumeration value="us: other"/>
              <xs:enumeration value="canada"/>
              <xs:enumeration value="france"/>
              <xs:enumeration value="rest of europe"/>
              <xs:enumeration value="rest of world"/>
              <xs:enumeration value="asia pacific"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="area" default="all areas" id="sel_sector">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="all areas"/>
              <xs:enumeration value="positions with consulting firms"/>
              <xs:enumeration value="positions outside of consulting"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="all sectors"/>
    <xs:enumeration value="crm"/>
    <xs:enumeration value="business process improvement"/>
    <xs:enumeration value="e-business"/>
    <xs:enumeration value="finance/ accounting"/>
    <xs:enumeration value="hr consulting"/>
    <xs:enumeration value="it / software development"/>
    <xs:enumeration value="marketing &amp; sales"/>
    <xs:enumeration value="outsourcing"/>
    <xs:enumeration value="strategy"/>
    <xs:enumeration value="technology"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="sector" default="all sectors" id="sel_sector2">
</xs:element>

<xs:complexType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="all industries"/>
    <xs:enumeration value="automotive/ aerospace"/>
    <xs:enumeration value="distribution / logistics"/>
    <xs:enumeration value="education"/>
    <xs:enumeration value="engineering &amp; manufacturing"/>
    <xs:enumeration value="facilities mangement"/>
    <xs:enumeration value="financial services"/>
    <xs:enumeration value="general management"/>
    <xs:enumeration value="healthcare &amp; pharma"/>
    <xs:enumeration value="information technology"/>
    <xs:enumeration value="leisure / lifestyle"/>
    <xs:enumeration value="purchasing / supply chain"/>
    <xs:enumeration value="retail / consumer goods"/>
    <xs:enumeration value="science / research"/>
    <xs:enumeration value="technology"/>
    <xs:enumeration value="transportation"/>
  </xs:restriction>
</xs:complexType>

<xs:element name="industry" default="all industries" id="sel_industry">
</xs:element>

<xs:complexType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="all salaries"/>
    <xs:enumeration value="£20-40k"/>
    <xs:enumeration value="£40-60k"/>
    <xs:enumeration value="£60-80k"/>
    <xs:enumeration value="£80-100k"/>
    <xs:enumeration value="£100-150k"/>
    <xs:enumeration value="£150k +"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="salary" default="£80-100k" id="sel_salary">
</xs:element>
</xs:sequence>
</xs:complexType>
Integrated XML Schema from Multiple Search Engines

Below is an integrated schema in XML format, generated by job meta-search engine by the integration of following job search engines:

- http://www.canjobs.com
- http://www.brightspyre.com
- http://promotions.monster.com
- http://www.jobs.net
- https://www.mymatrixjobs.com

Listing A.3: Integrated XML Schema

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<MetaSearchEngine>
  <JSE1>
    <URL>http://www.canjobs.com/</URL>
    <Method>POST</Method>
    <Keyword>keywords</Keyword>
    <Province>site</Province>
    <Job_Category id="ctype">
      <business>
        <accounting>accounting</accounting>
        <banking>banking</banking>
        <marketing>marketing, sales/marketing</marketing>
        <finance>finance</finance>
        <auditing/>
      </business>
      <computer_science>information technology, science, engineering  computer, it/mis, it/mis programmer/analyst</computer_science>
      <telecommunication>telecommunications</telecommunication>
      <internet>engineering  internet, internet/new media</internet>
      <software_engineering>engineering software</software_engineering>
      <system_analyst>it/mis  programmer/analyst</system_analyst>
      <system_designer/>
      <system_programmer>it/mis  programmer/analyst</system_programmer>
      <system_tester/>
      <software_engineering/>
      <web_design>design</web_design>
      <computer_science/>
      <engineering>engineering, engineering chemical, engineering civil, engineering computer, ........</engineering>
    </Job_Category>
  </JSE1>

  <JSE2>
    <Method>GET</Method>
    <Keyword>jobsearch</Keyword>
    <Job_Category id="cid">
      <business>business operations</business>
      <accounting>accounting/auditing</accounting>
      <banking>financial services banking</banking>
      <marketing>sales and marketing</marketing>
      <finance>finance and administration</finance>
      <auditing>accounting/auditing</auditing>
    </Job_Category>
  </JSE2>
</MetaSearchEngine>
```
<internet>internet</internet>

<software_engineering>
<system_analyst/>
<system_designer/>
<system_programmer/>
<system_tester/>
</software_engineering>

<web_design>arts design entertainment and media</web_design>

<computer_science>
<engineering>architecture and engineering, engineering / technical services

..........</engineering>
</Job_Category>
</JSE2>

<JSE3>
<URL>http://promotions.monster.com/keywordemployer/?WT_srch=1</URL>
<Method>GET</Method>
<Keyword>q</Keyword>
<State>state</State>
<City>city</City>
</JSE3>

<JSE4>
 <!--See Listing 6.2, Chapter 6 for http://www.jobs.net/--> 

..........</JSE4>

<JSE5>
 <!--See Listing 6.2, Chapter 6 for https://www.mymatrixjobs.com/candidate/Login.action --> 

.........</JSE5>
</MetaSearchEngine>
APPENDIX – B
SCREENSHOTS

Below is an integrated interface (XForm), generated by job meta-search engine by the integration of following job search engines:

- http://www.canjobs.com
- http://www.brightspyre.com
- http://promotions.monster.com
- http://www.jobs.net
- https://www.mymatrixjobs.com

Figure B.1: Integrated Interface of Job Meta-search
APPENDIX – C
LIST OF PUBLICATIONS

- Tabbasum Naz, Jürgen Dorn and Alexandra Poulavassilis (June 2009), A Hybrid Approach to Schema and Data Integration for Meta-search Engines, In: Proceedings of 9th International Conference on Web Engineering, San Sebastian, Spain


- Jürgen Dorn and Tabbasum Naz, (Apr 2008), Structuring Meta-search Research by Design Patterns, In: Proceedings of International Computer Science and Technology Conference (ICSTC), San Diego, California, USA


REFERENCES


44-55, Asilomar, California


[Con99b] “Forms”, At: http://www.w3.org/TR/html401/interact/forms.html#h-17.1


[Con04b] “RDF Primer”, W3C Recommendation (2004), At: http://www.w3.org/TR/rdf-primer/

[Con04c] “RDF Vocabulary Description Language 1.0: RDF Schema W3C Recommendation” (2004), At: http://www.w3.org/TR/rdf-schema/

[Con07a] “SPARQL Query Language for RDF W3C Candidate Recommendation” (2007), At: http://www.w3.org/TR/rdf-sparql-query/#introduction


[JavaDOM] At:http://java.sun.com/j2se/1.4.2/docs/api/org/w3c/dom/package-summary.html


[Jena07] “Jena- A Semantic Web Framework for Java” At:
http://jena.sourceforge.net/index.html


Technical Report, DBAIEC, Institute Faculty of Informatics, TUWien, Austria


[Por06] Porter M (2006), At: http://tartarus.org/~martin/PorterStemmer/


Tolksdorf R, and Eckstein R, (April 2004), ”XML Clearinghouse Report 8 Recruitment”, Freie University, Germany

Apache Tomcat”: At: http://tomcat.apache.org/


“XForms”, At: http://www.w3schools.com/xforms/default.asp

“RDF”, At: http://www.w3schools.com/rdf/default.asp

“RDF Schema”, At: http://www.w3schools.com/rdf/rdf_schema.asp


### LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMA++</td>
<td>COmbining Match</td>
</tr>
<tr>
<td>HR-XML</td>
<td>Human Resource – Extensible Markup Language</td>
</tr>
<tr>
<td>HTML</td>
<td>Hyper Text Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and Communication Technology</td>
</tr>
<tr>
<td>IE</td>
<td>Information Extractor</td>
</tr>
<tr>
<td>IEXP</td>
<td>Interface Expression</td>
</tr>
<tr>
<td>LEX</td>
<td>Layout Expression based Extraction</td>
</tr>
<tr>
<td>MDR</td>
<td>Mining Data Records</td>
</tr>
<tr>
<td>MSE</td>
<td>Meta-search Engine</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>OWL</td>
<td>Web Ontology Language</td>
</tr>
<tr>
<td>RAN</td>
<td>Representative Attribute Name</td>
</tr>
<tr>
<td>RDF</td>
<td>Resource Description Framework</td>
</tr>
<tr>
<td>RDF-S</td>
<td>Resource Description Framework-Schema</td>
</tr>
<tr>
<td>SOC</td>
<td>Standard Occupation Scheme</td>
</tr>
<tr>
<td>SPARQL</td>
<td>Simple Protocol and RDF Query Language</td>
</tr>
<tr>
<td>WISE</td>
<td>Web Interfaces of Search Engines</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
INDEX

A
Abstract factory pattern 46
Alignment re-use matcher 65
All-in-one meta-search engine 5
Attribute 31
Attribute analysis 57
Attribute extractor 56
AutoMed 28

C
Cache management 80
Cache manager 4
Class 31
Clustering approach 49
COMA++ 27
CommonCV 35
Competency 36
Competency sub-ontology 82
Components of ontology 31
Confounding conflicts 8
Cosine similarity 28
Crawler 2
CUPID 27

D
Data integration 61
Data value cardinality matcher 64
Default value matcher 65
Design Pattern 26, 45, 49, 50, 51, 52
Desktop meta-search engine 5
DOM (Document Object Model) API 89
Domain Ontology 31
Duplicate eliminator 5, 78

E
Element level techniques 63
Euclidean distance 28

G
Geo-Lite country 88

H
Hidden Web meta-search engine 5
Holistic schema matching approach 49
HR-BA-XML 34
HR-XML 35
HTML 16
HTML Form 17
Human resource domain 53
Human resource management 35, 80
Hybrid approach 49, 63, 107, 111

I
ICSE (International Classification of Status in Employment) 38
ICT (Information and Communications Technology) 38
ID matcher 64
Individual 31
Information extractor 50, 71, 94, 102
Integrated vocabulary 31
Interface extractor 50, 55, 94, 102
Internet recruitment 12
ISCO (International Standard Classification of Occupations) 37
ISCO-COM 37

J
Java servlets 86
Jena framework 87
JobOlize 13
Job seeker 13

L
Language based matcher 64
Levenshtein distance 28
Lixto meta-search 23
Lixto transformation server 23
Lixto visual wrapper 23

M
Map cardinality 66
MetaExplorer 24
MetaIntegrator 24
MetaQuerier 24
Meta-search creation process 44, 53, 93
Meta-search engine 3
Meta-search query interface generator 59
Meta-search usage process 45, 54, 100
Mozilla XForm Plugin 89
MySQL 79, 87
MySQL java connector 87

N
Naming conflicts 8

O
Occupation sub-ontology 81
Ontology 30, 80
Ontology based techniques 65
Ontology mapping 26
Ontology matching 26
OWL 33

P
Protégé 88

Q
Query dispatcher 5, 69, 100
Query interface 4
Query interface generator 49, 61, 63, 91, 98
Query processing 93

R
RDF 32
RDFS 33
Record collector 70, 102
Recruiter 12
Relationship 31
Result collector 76, 102
Result merger 5, 51, 78
Result ranker 5, 52, 78, 103

S
Scaling conflicts 8
Schema integration 59
schema mapping 26
Schema matching 26
Search approach 1
Search engine selector 4, 55
Semantic heterogeneity 8
Semantic Web 30
SIDES (Staffing Industry Data Exchange Standard) 36
Snorri meta-search 24
SOC (Standard Occupation Classification scheme) 34, 37
Social network search engine 3
SPARQL 34
Strategy design pattern 50
String based matcher 63
Structure heterogeneity 8
Structure level techniques 65
Subject directory 2
Syntactic heterogeneity 8

T
Tomcat 88
Topic hierarchy 31

U
Upper level ontology 31

V
ViNT (Visual Information aNd Tag Structure) 25, 50

W
WISE-Iextractor 20
WISE-Integrator 20
Wrapper 25
W3C XML Schema 14

X
Xform 18
Xform plugin 89
XML 14
XML Schema 13
XML Schema generator 58, 96
XML technology 14