# SUPPLYING DATA FOR AN RDF BASED CONTENT MANAGEMENT SYSTEM

By

KARTHIK CHINNAYAN MUTHUKUMARASWAMY

Bachelor of Engineering in Computer Science

Bharathiar University

Coimbatore,India

2002

Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2005

SUPPLYING DATA FOR AN RDF BASED CONTENT

MANAGEMENT SYSTEM

Thesis Approved:

Dr. MARCIN PAPRZYCKI

Thesis Advisor

Dr. JOHNSON THOMAS

Dr. DEBAO CHEN

Dr. GORDON EMSLIE

Dean of the Graduate College

# ACKNOWLEDGEMENTS

I thank my advisor Dr. Marcin Paprzycki for giving me an opportunity to work under him and in his ‘e-Travel’ project. I thank your for your encouragement, guidance and timely support throughout the entire duration of this project and thesis.

I would like to thank my committee members, Dr. Johnson Thomas, Dr. Martin Crossland and Dr.Debao Chen for their suggestions and opinions. I thank all the members of e-Travel project, especially, Mr. Minor Gordon, Mr. Jimmy Wright and Mr. Szymon Pisarek. Your ideas and technical insights during the course of this project were very valuable to me.

I would like to extend my gratitude to Dr. John Chandler, Dr. George Hedrick and all other computer science department faculty and staff for their support and assistance in times of need. I would like to thank all my friends here at Oklahoma State University for their support and encouragement when I needed them the most.

I am ever thankful to my parents Mr. Muthukumaraswamy and Mrs. Krishnaveni, my sister Ms. Sowmiya for their continuous support. Thank you for all your love and encouragement that will take me through every obstacle.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# LIST OF CODES LISTINGS

# CHAPTER 1

## INTRODUCTION

There are many travel sites offering great services to their customers. Some of these services include providing best deals on hotels, rental cars and flights. Travelocity and Expedia are two such sites that are popular among the travelers but, when a traveler expects more, like information on tourist attractions at or near his destination, these sites do not fair well. For example, Expedia's "activities" service provides great deals on theme parks and tours offered, but, only for few "well-known" destinations. Meaning, a search for 'Oklahoma City' returned with zero results. This is no good for a traveler visiting Oklahoma City.

The fact is that there are tons of travel related data available on the internet, but in bits and pieces. It is therefore possible to use these data to provide more travel choices and personalized information to the user. In this juncture, two popular questions namely, "How big is the Internet?" and "How fast is it growing?" seek immediate attention and are somewhat answered by Metamend [22], a web site optimization company. According to them, there are more than 171,000,000 domain hosts currently in use and about 10 million new, static pages are added everyday. Now with this large number of data sources, finding 'travel' data is like finding a needle in a haystack. Therefore a system that attempts to provide personalized user experience should take note of this situation.

Personalized Internet Services have now become an important topic of research and study. The concept of personalization is possible only with a system that can integrate data present in large number of repositories and deliver it to the user.

The design of an e-travel system that aims to provide Personalized Traveler Information Services can be found in the conference paper [36]. As it can be seen from the paper, the e-Travel system is 'Agent based' and has 2 main subsystems: Content Management System (CMS) and Content Delivery System (CDS), along with other agent components. CMS is responsible for organizing incoming and outgoing content from and to the system whereas CDS handles delivery of information to the user. Detailed description of CMS can be found in chapter 2. CDS and other related concepts are further discussed in section 1.1 titled 'Requirements for an E-Travel system'. Another paper [21] provides an updated view of the Agent Based travel system with a new subsystem included- the Content Collection System (CCS), whose functionality was a part of the CMS in [36].

Objective of this thesis is to propose an architecture for the CMS as discussed in [36]. This architecture will include functionalities to effectively collect and integrate data from heterogeneous data sources. It will also incorporate mechanisms for software agents to access the data stored, in order to provide personalized results to the user.

## 1.1 REQUIREMENTS FOR AN E-TRAVEL SYSTEM

As it was mentioned in [36], the e-Travel system in addition to content management system requires other supporting sub systems for its operation. This section adapted from another thesis work [3], provides an introduction to these supporting components.

Subsystems as described in the above mentioned thesis work:

1. Content Delivery System

   As mentioned before, this system handles personalized delivery of content to the users of the system. Users of this system access information via Internet enabled devices such as PC enabled browsers, Mobile phones, Palmtops etc…This subsystem is mostly agent based as it consists of agents like travel expert agent, advertising expert agent and internet search agent.

2. Storage

   There are many ways by which this storage subsystem can be implemented. Current Internet travel services have based their underlying storage system on traditional databases which limit their search capability. In this thesis, an RDF dependent storage system is used. This approach is further discussed in Section 4.5.6 titled 'Database'. Storage subsystem is included as a part of CMS in this thesis work.

3. GIS

   GIS subsystem is used to transform location into latitude and longitude pair (reverse geo-coding). For example, given an address of a restaurant, GIS subsystem will result in latitude and longitude values for it. There are 2

approaches to for incorporating a GIS subsystem. First, the GIS can be implemented in house, in other words, it can be developed as a part of the whole system. Second, the GIS subsystem can be obtained from a third property.

4. User Interface

   User Interface is a medium through which a user interacts with the system. This user interface should be developed with device independence in mind. I.e. it should provide the same functionality to all kinds of users. Such an interface can be designed and developed using XUL and XUP [59].

## 1.2 ORGANIZATION

The thesis is organized as follows: Chapter 1 introduces current status of travel web sites and where they stand when providing personalized content is concerned. It then presents the requirements of an e-Travel system. Chapter 2 provides an introduction for the Content Management System. Subsequent sections of this chapter provide detailed analysis of data sources available, how these data sources are heterogeneous in nature and why integration is necessary. Chapter 3 provides an analysis on existing approaches for data integration. In chapter 4, the proposed architecture is discussed. Notes from existing architectures and challenges to face during CMS architecture development are presented in this chapter. Components that make the CMS are also presented in detail. Chapter 5 provides a discussion on the implementation issues and provides directions for future work.

# CHAPTER 2

## HISTORY AND BACKGROUND

## 2.1 CMS – DESIGN ISSUES

Architecture for a Content Management System should address certain issues as mentioned in the Data Integration Primer by the U.S Department of Transportation [8]. They are,

a) Where the data comes from and who collects the data

b) Method and frequency of collection

c) Reference system or systems used

d) Structure, format, and size of data

e) How the data are processed, transmitted and stored

f) How the data is used

g) Applications that draw data from the databases

h) Types of reports produced.

Following sections of this chapter give more details on what these issues mean and Chapter 4 - 'Proposed Architecture' addresses most of them. For example, you can read detailed explanations on topics like 'Methods and frequency of collection' with respect to the "Agent based Travel System" in that chapter.

## 2.2 DATA SOURCES

Information on the web is made available by various services. Very few data from these services/ sources are well structured with semantic annotations (more information on why semantic/ meaning based data becomes important is presented in section 2.3), and most of them do not have a proper structure at all, for example HTML pages. Main groups of information services on the internet include gateways, search engines, portals and on-line databases.

a) Gateways (Reference Services)

   These reference services usually group URLs or WWW addresses of various information services on the web. Gateways are built and maintained manually, a complicated task by itself, taking into the account the dynamic nature of the web. School-libraries.org is an example for a gateway that provides resources and services for school libraries [47].

b) Search Engines

   Search engines are distinct from gateways as they automatically browse WWW resources and index the document they find. Information collected by these search engines include URLs, titles, indices and teasers about a resource. Google, Yahoo and MSN Search are examples of search engines that are more commonly used [12, 61,27].

c) Portals

   According to Webopedia [53], a portal is a website or service that offers a broad array of resources and services, such as news, on-line chat, email accounts and

advertising. Yahoo and MSN are two of the most popular portals [60, 26]. Vortals (vertical industry portals), considered a sub category of portal, provide information and resources for a particular industry. They are a new way of catering to customer's focused-environment preferences. For example [5] is a Business to Business vortal for the city of Chicago, Illinois.

d) Online Databases

Nowadays various databases and directories are available on the internet and they often have a web interface and a query language associated. For example, Oklahoma State University library provides a list of indexes and databases on various subjects to search from [30].

## 2.3 HETEROGENEOUS DATA TYPES

HTML (Hypertext Markup Language), a lingua franca for publishing hypertext on the World Wide Web is a non-proprietary format based on SGML (Standard Generalized Markup Language). It has gone through several stages of evolution and today's HTML has wide range of features reflecting the needs of a very diverse and international community. Although it served and still continues to serve its purpose, i.e. presenting data, new demands like information sharing and long term reuse of data was not achievable with its limited and predefined tag set.

XML (eXtensible Markup Language) [56], also derived from SGML was the solution to the above two problems. With its extensible (user defined tags) and platform independent nature, XML proves to be an effective way for structuring data. It is a family

of technologies that includes eXtensible Style sheet Language Transformation - XSLT [58], Document Object Model - DOM [9], and Cascading Style Sheets -CSS [6] etc... While HTML's purpose is to present the content, XML's purpose is to describe the content. Therefore, XML is not here to replace HTML, but to complement it.

With an exponential growth of information it became obvious that the Web can reach its full potential only if it becomes a place where data can be shared and processed by automated tools as well as by people. This in fact is the vision statement of the semantic web interest group [49]. Therefore efforts to put 'machine understandable data' on the web are quickly becoming a high priority. One such effort is to use a language that is capable of describing machine understandable data. This language has to satisfy 3 requirements as mentioned in ontoknowledge.org [48]. They are,

a) Universal expressive power,

b) Support for syntactic interoperability, and

c) Support for semantic interoperability.

While syntactic interoperability talks about extracting data from a source, semantic interoperability defines mapping between unknown and known terms in the source.

XML satisfies the first 2 of the above 3 requirements mentioned. I.e. it is a formalism for defining a grammar, and any data can be encoded in XML if a grammar can be defined for it. Hence it satisfies the first requirement. XML by purpose aims at the structure of the document and does not impose any common interpretation of the data in the document. This is one reason why it satisfies the second requirement and not the third.

On the other hand, RDF considered an essential part of the semantic web and based on XML has already satisfied the first requirement [37, 49]. More details on RDF in section 4.3 titled 'RDF for building domain ontology' of Chapter 4 illustrate how RDF satisfies second and third requirements mentioned above.

At this point, data types under consideration range from HTML to RDF or from syntax to semantics. Therefore we can classify data sources in 2 ways: based on their structure/format (structurally heterogeneous) or based on their associated meaning (semantically heterogeneous).

a) Structural heterogeneity

   It is associated with differences in the underlying structure of source documents. For example, two different data documents can define the same concept but can differ in their basic structure. Examples are HTML web pages and RDF documents describing a hotel web page. While HTML pages use tags to describe an entity/ resource, an RDF document associates metadata to describe the same. More information on how RDF is employed can be found in section 4.3.

b) Semantic heterogeneity.

   While structural heterogeneity is with differences in structure, semantic heterogeneity is with the meaning. In this case, two data documents can have the same structure but still mean two different concepts. This is common with XML data documents having tags with same names but describing different concepts. This exact problem with XML documents can be resolved using XML name spaces, a collection of names identified by a URI reference [52]. It is because of

this data heterogeneity, there is a great need for efficient and practical integrating mechanisms.

## 2.4 INTEGRATION

As mentioned before, integration of data sources is an integral part of any Content Management System because of the diversity of data sources. Integration is also necessary to provide personalized services as it makes sure that all the necessary data to make inferences and decisions are readily available. According to Nasfa.net [8], general data integration benefits include,

a) Availability/ Accessibility
b) Timeliness (Databases are time stamped as to when they were updated etc…)
c) Integrated Decision making, and
d) Completeness (Historical and recently collected data can be found in the database. In addition, missing records or data fields can be easily flagged by the data management process).

## 2.5 METADATA

A key function of CMS is the creation/maintenance of metadata or 'data about data'. In general, metadata can be divided into technical metadata and business metadata.

a) Technical metadata describes the physical nature of data: how it is stored, its structure, location, access procedures etc… Technical metadata is meant for database administrators, designers and developers.
b) Business metadata on the other hand describe what is stored in the database, how it may be utilized and what it means in the real world.

More information on the topic can be found in [4]. The question 'How metadata plays an important role during integration?' will be answered in Chapters 3 and 4 when existing and proposed integration architectures are discussed.

# CHAPTER 3

## EXISTING ARCHITECTURES FOR CMS

There have been multiple architectures and methodologies presented for CMS and for integrating data. Possible integration techniques can be classified in two ways, first on the level of updated information required, second, on the mapping of source schemas to global schema. Some of the popular architectures under each approach are discussed below.

## 3.1INTEGRATION TECHNIQUES BASED ON THE LEVEL OF UPDATED DATA REQUIRED

There are two paradigms for information integration in this class: data warehousing (eager) approach and on-demand retrieval (lazy) approach [54]. In the data warehousing approach all necessary data is collected in a central repository ahead of time. On the other hand, in the on-demand retrieval approach, data is collected from integrated data sources during query evaluation, i.e. during run time. The latter approach is also called a mediated approach, since the module that decomposes queries and combines results is often referred to as the 'mediator'. MOMIS approach is an example for mediator based architecture [10].

### 3.1.1EAGER VERSUS LAZY APPROACH

Each of these approaches has its own advantages and disadvantages. For example, the lazy approach is well suited for situations where vast amount of data is available from large number of information sources, as it returns fresh results. But it may incur inefficiency and delay in query processing when information sources are slow, expensive and periodically unavailable. On the other hand, eager approach is well suited for immediate querying and analysis by clients. A major drawback is that it does not guarantee up-to date results at all times. The same concept is discussed in slightly different terms: Aggregation and Selection in a thesis work by another student [3]. Table below is from the above thesis material.

| Method | Advantages | Disadvantages |
|---|---|---|
| 1. Aggregation | Immediate availability of local content.<br>Require no preprocessing.<br><br>Implementation is simple. | Cache coherency problem.<br><br>Amount of data generated is large.<br>Resource Intensive. |
| 2. Selection | Amount of data generated is small.<br>Resource effective.<br>No cache coherency problem. | Require pre – processing.<br><br>Implementation is arduous.<br>Availability cannot be guaranteed. |

**Table 1: Aggregation and Selection Approaches**

## 3.2 INTEGRATION TECHNIQUES BASED ON MAPPING OF SCHEMAS

Schema, according to Webopedia [53] is the structure of a database system and is often a graphical depiction of a database structure. In this aspect, many data sources have

schemas associated with their data (e.g., XML schema, RDF schema). Therefore integration approaches are sometimes classified based on how data source schemas and target database schemas are related. These approaches range from 'Bottom-Up Approach for Integration of Heterogeneous data' [43], a Global-As-View (GAV) kind of approach to 'User Specific semantic integration' [31], a Local-As-View (LAV) kind of approach.

### 3.2.1 GAV BASED APPROACHES

A GAV approach is when concepts in the global schema are defined as views over the sources. In this case, the global schema has to be revised every time a source change, but query processing is straight forward and involves the concept of "query unfolding". Few examples of GAV based architecture are presented below.

In [43], different sources are integrated in a bottom-up approach to build domain ontology. It involves 2 steps: DTD-schema conversion and schema integration. In the first step, each DTD is converted into a conceptual schema in the Canonical Conceptual Model (CCM). In the second step, all local CCM conceptual schemas that result from the first step are integrated to form an ontology.

In another GAV based approach [20], all data source schemas are first transformed into union-compatible schemas that are syntactically identical. An arbitrary one of the union schemas is designated as the global schema or selected for further transformation into a new schema that will become the global schema.

A more advanced GAV based architecture can be found in the MOMIS approach [10]. In this case, data sources are "wrapped" into a special representation. Later a

common thesaurus is generated from the previous representation which is later used in forming a Global Virtual View (GVV) of the domain.

### 3.2.2 LAV BASED APPROACHES

A LAV approach is one where sources are defined in terms of global schema. Quality in this approach depends on how well we have characterized the sources. RDF based approach for integrating data sources [42] can be considered as an LAV type of data source integration. In this case, a Conceptual Model (CM) is maintained centrally at the schema level. The architecture is divided into 5 separate layers each of them performing five different but interrelated functions. Again this is a mediated approach as it contains a mediator to decompose queries and return results.

In [31] approach, integration of heterogeneous data sources is based on user's information needs, emphasizing his individual way of perceiving a domain of interest. It can be thought of as an extension of the LAV approach as it allows an *ex-ante* view definition that allows only data items that are semantically related according to the user's individual perception of the particular application domain to be integrated.

# CHAPTER 4

## PROPOSED ARCHITECTURE FOR CMS

## 4.1 NOTES FROM EXISTING ARCHITECTURES

Architectures that were discussed in the previous chapter dealt with many important concepts that we can take notes from. Ranging from simple bottom-up approach for XML sources to User specific semantic integration, almost all of them had few common things like data sources, converters, common base model and storage space. These are in fact the essentials for any Content management system.

Most of the approaches mentioned in the previous chapter were of the 'lazy' kind in one way or another. They have all the advantages of 'on-demand retrieval' like fresh results, effective resource utilization etc... However, I believe that we can combine the best of both worlds, lazy and eager approaches, to design an architecture that is suitable for this travel project. An ideal architecture of a content management system will be one in which some data is pre fetched, processed, integrated (using ontology – a global schema, discussion of that follows later in this chapter) and stored in the database, while other data, when not present in the database, is fetched and processed upon user queries. As we are dealing with a specific domain (travel) in this thesis, it is possible and practical to pre fetch travel related information from different sources.

## 4.2 CHALLENGES TO FACE

There are few very important issues that we need to consider before proposing architecture for the CMS. These issues can be considered as challenges and are listed in the following page.

1) Specifying procedures for acquiring and exploiting Internet sources

   The Content management system would gather information only from trusted sources, also called 'verified content providers' that are known to the system in advance. More details on who 'verified content providers' are and few examples for them are provided later in this chapter while discussing the components of the system. These sources have to be studied for their structure and topology for establishing proper navigation techniques and parsing mechanisms.

2) Providing an efficient storage and retrieval mechanism for data

   Another important issue is to adhere to an efficient way to store and retrieve information. We need to consider the fact that the data collected will be used by agents (like personal agents) to provide optimized results for customers. Hence there is a necessity of having a data model that supports inference and at the same time extensible to accommodate more concepts. Following ontology (definition follows shortly) based approach justifies our case. RDF and RDFS described in the following section can be used to build such an ontology.

3) Describing the domain with a single ontology

   According to Wikipedia [55],

> "Ontology is the attempt to formulate an exhaustive and rigorous conceptual schema within a given domain, typically a hierarchical data structure containing all the relevant entities and their relationships and rules within that domain".

According to [15], an ontology can be used,

a) For agent – agent communication in e-commerce
b) In semantic based search
c) To provide richer service descriptions that can be more flexibly interpreted by intelligent agents.

There are basically two approaches for developing ontologies, a 'top- down' approach (i.e., developing an ontology large enough to encapsulate many notions) and 'bottom – up' approach (developing domain specific ontologies and linking them to form a whole). In this case, an ontology that pertains to the travel domain has to be developed, by following the 'bottom-up' approach. Developing an ontology is an important step towards the integration of heterogeneous data sources.

4) Providing a mapping mechanism (i.e., a logical connection) between information sources and domain ontology

This is in fact one of the most challenging tasks while building the content management system. It involves linking/mapping data collected from different sources to the domain ontology that pertains to the final database.

## 4.3RDF FOR BUILDING THE DOMAIN ONTOLOGY

As mentioned in chapter 1, RDF is a framework for describing data about web sources. A web resource is anything that can be identified by a URI (Uniform Resource Identifier). In this section we will see some basic features of RDF and how it is an effective choice for building the domain ontology. Also, we will see some features that make it a suitable choice for use in agent travel system. An RDF statement is usually made up of a subject, a predicate and an object. Consider the example given in [38]. The English statement,

*Statement 1:* **http://www.example.org/index.html** has a **creator** whose value is **John Smith**

is described in RDF as, a subject http://www.example.org/index.html

a predicate http:// purl.org/dc/elements/1.1/creator

an object http:// www.example.org/staffid/85740

We can see from the statements above that URIs are used to describe a subject as well as predicates and objects. This usage of URIs to identify resources helps us in the process of semantic web. For example, consider the English representation of an RDF statement,

**http://**www.example.org/staffid/85740 has a **father** whose value is **Bruce Smith**

Now from the above 2 statements we can infer that 'father' of the 'creator' of the resource http://www.example.org/index.html is Bruce Smith. Further we sometimes need not define our own terms, rather include terms that have already been defined elsewhere.

For example, in the first statement, ‘creator’ was a term that was borrowed from a well known ontology description called ‘Dublin Core’ [11].

Each RDF statement/ expression having a subject, predicate and object is called a triple and a set of such triples is called the RDF graph. A triple corresponding to the above statement is “**ex: index.html dc: creator exstaff: 85740**” where ex, dc and exstaff are namespace prefixes. The figure below depicts an RDF graph model with its subjects and objects as nodes. The arc in the graph is the property that links the corresponding subject and predicate.

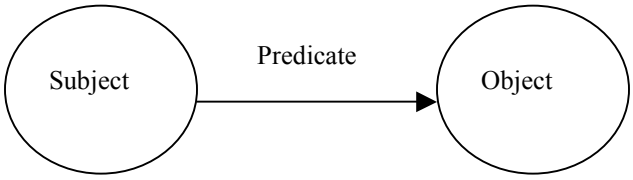



**Figure 1: Node and Directed Arc Diagram**

While triples are short hand notation for a statement, RDF/XML is the normative syntax for writing RDF. For example, RDF/XML for *statement 1* is given below.

```
<?xml version="1.0"?>
<rdf:RDF        xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
                xmlns:exstaff=http://www.example.org/staffid/
                xmlns:dc="http://purl.org/dc/elements/1.1/"
                xmlns:ex="http://www.example.org/">

  <rdf:Description rdf:about="http://www.example.org/index.html">
    <dc:creator exstaff:85740>
  </rdf:Description>

 </rdf:RDF>
```

**Listing 1: Sample RDF/XML**

RDF provides a way to express simple statements about resources, but there is also a need to have vocabularies (list of terms) to be used in those statements. Some

extensions to RDF like ‘rdfs: class’, ‘rdfs: range’ etc… provides the ability to describe classes and properties so that we can develop our own vocabulary. They are often used to describe terms and the relationships between them. These extensions are grouped under a category called the RDF Schema [39]. Thus we can prepare an ontology corresponding to domain by involving RDF along with RDFS.

## 4.4 ARCHITECTURE DIAGRAM

The pictorial depiction of the proposed architecture is provided in Figure 2 below. As mentioned in the initial paragraphs of this chapter, the architecture consists of few basic components that are essential for every content management system. In addition, it has the RDF based global ontology (to define the travel domain) and the Jena application component [17]. There are wrappers for each type of source. The source metadata file will be used to keep track of various sources registered to the system. Database accessed using the Jena component is used to store resource instances based on the global ontology. Detailed descriptions of each component and their usage are presented in the remainder of this chapter.

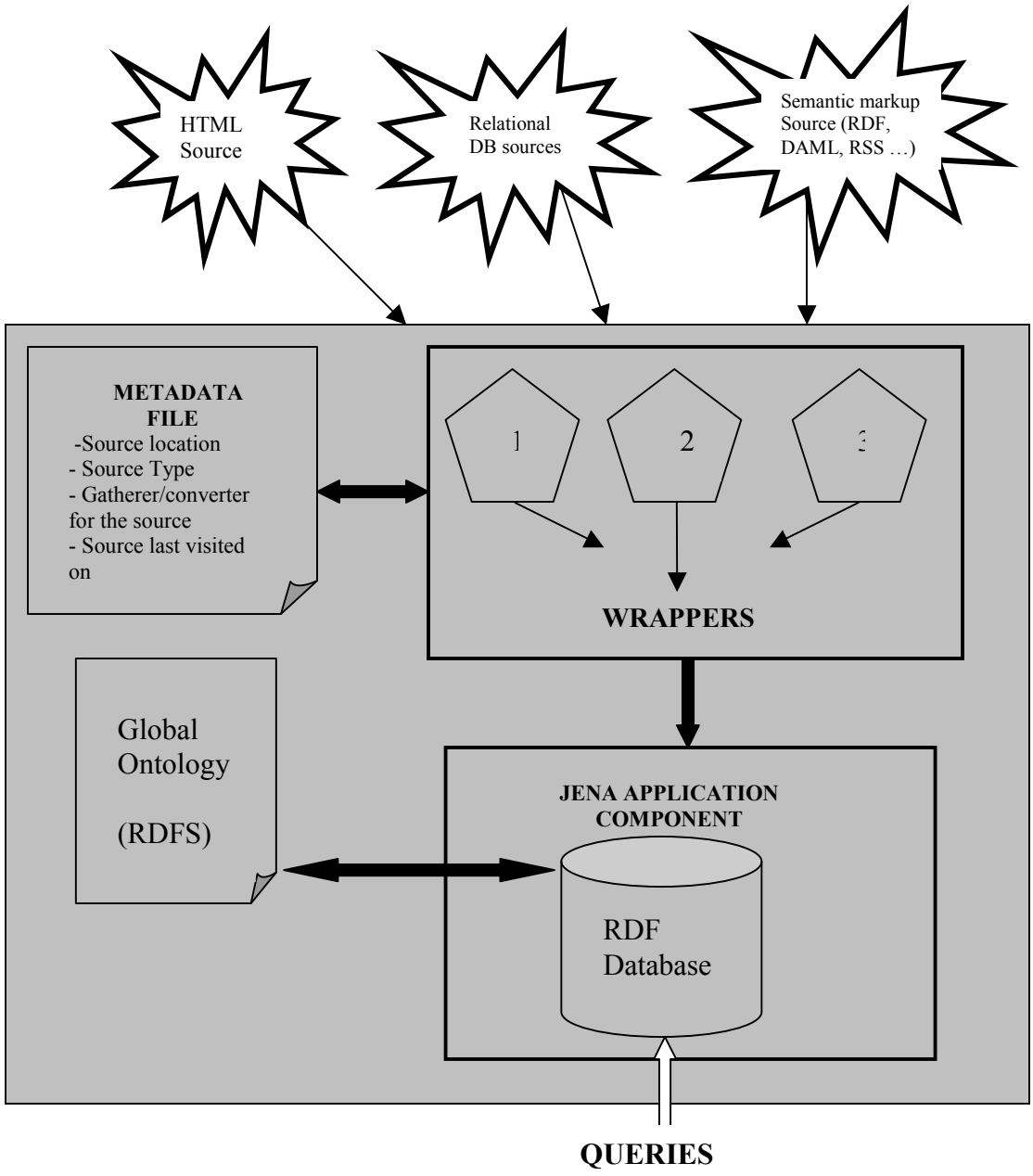



**Figure 2: CMS Architecture Diagram**

## 4.5 COMPONENTS IN THE PROPOSED ARCHITECTURE

### 4.5.1DATA SOURCES

Different possible data sources for a content management system were mentioned in chapter 2. We also saw how these data sources are heterogeneous in nature and why integration is necessary. In this thesis, data sources are expected to possess the following qualities.

1) Considered useful and trustworthy

2) Sufficiently structured.

Trustworthiness can be associated with what are called “Verified Content Providers” (VCP). This concept is followed throughout the Agent Travel Project and implies those content providers who maintain a degree of conformance to expected standards of accuracy, format and availability of described travel options. Here we assume that VCPs maintain well defined structures and does not change them often. For example, we can label Travelocity.com to be a VCP as the structure of their HTML pages describing hotels in each and every city/state are same and does not change often. Another example of VCP is chefmoz.org that has an RDF data dump of hotels all over the world. This RDF data dump is based on a schema proposed and maintained by chefmoz.org. Other VCPs include online databases with an associated schema and a web interface for accessing them.

According to [32], an information source can be described in terms of business and technical characteristics. Information sources in terms of Business characteristics include,

1) Active Information sources – These sources of data send any new information available without any action from the recipient side. It is more like the "Push" technology. An example of an active information source [2] is any site that provides RSS (RDF Site Summary Feeds) feeds [44]. RSS, based on RDF/XML provides short descriptions of web content along with links to detailed versions of the content as an XML file.
2) Passive Information sources – Passive sources are those that must be visited and new documents must be downloaded by the recipient. Few examples are HTML pages and RDF data dumps.

Technical characteristics with respect to information sources deal with the way documents are stored. In this regard, information sources are classified as,

1) Static Sources, where documents are stored as separate files (e.g. HTML, XML) in the file system of the web server
2) Dynamic sources, where documents are stored in underlying databases and presented to users based on queries.

Every data source handles entity types or resources. For example, a travel website will provide information on hotels and restaurants. By this way it provides links to many instances of hotels and restaurants in a given area. These resources instances (e.g. Hampton Inn of Stillwater) belonging to the hotel entity type are the ones we are interested in. By having a collection of these instances along with instances of parks nearby, we can provide a customer with personalized results.

## 4.5.2 GLOBAL ONTOLOGY

As mentioned in the above section titled 'Challenges to Face' of this chapter, ontology for this thesis [23] was developed in RDF. It covers the travel domain and includes concepts like restaurants, airlines, hotels etc… The figure [Figure 3] in the following page provides a pictorial view of concepts covered in the above mentioned ontology. This corresponds to the real- world part of the ontology. One can see that the concept 'place' has latitudes and longitudes property associated with it. This helps in deciding the exact location of a place (e.g. a restaurant) using its geographic co-ordinates.

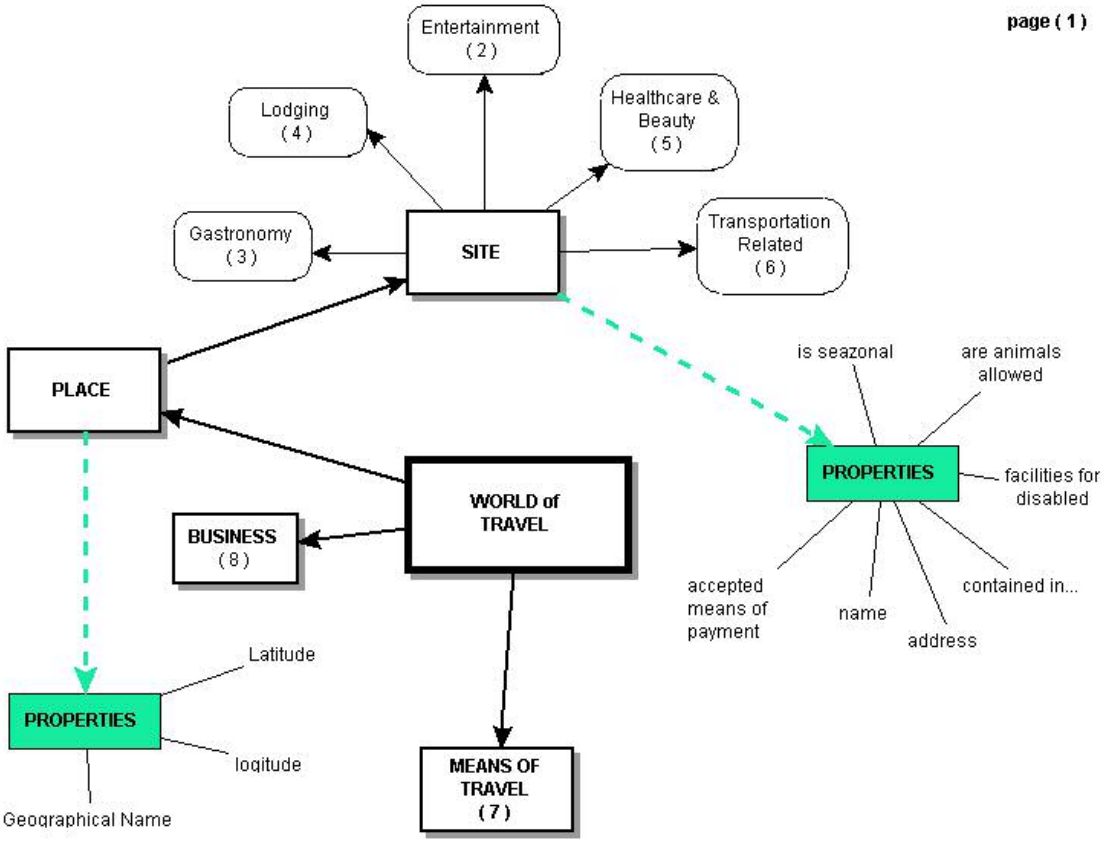



**Figure 3: Real World part of Travel Ontology [ Minor]**

On the other hand, there is a business part of the ontology that deals with 2 basic entities namely a customer and a ServiceProvider. There are other entities like ticket and connection to further enhance the business part of the ontology.

### 4.5.3 WRAPPER

This component includes Data Gatherer/ Converter sub components. Wrappers by definition are applications that transform data from one form to another. The functionality of the Wrapper component in this architecture is to gather data from various sources. Data in this case corresponds to the list of desired resource instances from the source, for example to collect details on all the hotels in a given area.

These data are later mapped to the global schema described in RDF(S).

a) Data Gatherer/ Converter component:

The process of gathering & converting data depends on the type of data source. For example, we can parse HTML pages for data. This parsing can be done with the help of few ready made parsers that are available on the Internet or a custom parser can be written. As said above, HTML pages are semi-structured or sometimes not structured at all. Therefore any parser (Readily available/ Custom made) has to take this drawback into consideration. Few other sources like relational databases can be queried for data. SQL Server2003 for example has an option to return responses to queries in XML format.

On the other hand RDF data sources can be directly mapped to the underlying data model without any modifications. All the mappings to final RDF model are performed in the Jena Component discussed below.

### 4.5.4 METADATA FILE

Another important component in this architecture is the wrapper metadata file. It is closely associated with the wrapper component and keeps information that is pertinent to data gatherer/ converter module. For example, it stores the list of data sources available, type of source (for example, web page, relational database, RDF document, RSS feeds etc…) and data gatherer/ converter associated with it and dates when a particular source was visited. This information is significant for building a gatherer/ converter system that is 'adaptive' with respect to time. For example, the structure of an HTML page from a VCP changes less frequently. A data gatherer/ converter can eventually understand how frequently a source changes with the help of this wrapper metadata file. This file will be an RDF/XML file allowing easy manipulation of data with the presence of custom defined tags. As the number of sources dealt change when new sources are added and outdated sources are removed, constant updating of wrapper metadata file becomes essential and this further justifies the use of XML.

### 4.5.5 JENA APPLICATION COMPONENT

Two main functions provided by this component include a) Mapping data derived from different data sources to the underlying RDFS model and eventually storing them in the database b) Querying the RDF database for responses to user queries (uses Jena framework).

Jena, a Java framework for building Semantic web applications provide a programmatic environment for RDF and contains an RDF API, In-memory and persistent storage and support for RDQL- a query language for RDF [40]. In addition, Jena provides inference support that help us (using axioms and rules associated with the reasoner and optional ontology information) to derive additional information from the RDF data

### 4.5.6 DATABASE

Database is essential as in any content management system. For this thesis a database that is supported by Jena for persistent storage is chosen. This is due to the fact that Jena/db modules provide an implementation of the Jena model interface with ability to store and retrieve RDF statements using a database [18]. Currently Jena supports MySQL, PostgreSQL and Oracle for storing persistent RDF data [28, 33, 29].

### 4.5.7 MAIN ADMINISTRATOR COMPONENT

This component controls the startup and functioning of the content management system. This component helps the database administrator to work with Jena models stored in the database. For example, we will be able to load and print models using this component. In addition to that, it will display a graphical user interface (GUI) for the CMS source scheduler screen. The source scheduler screen will be used to schedule selected sources at a given time.

# CHAPTER 5

## IMPLEMENTATION ISSUES

## 5.1 FINDING TRAVEL RELATED SOURCES

Within the Agent Based Travel system context, there is a dedicated Search Agent responsible for finding travel related information from the Internet. Semantic understanding capability is an essential attribute of this agent in order to differentiate travel content from others. More details on how this is carried out can be found in another Master's thesis work [3, 36].

In a typical scenario, the search agent will communicate with the CMS with the new source discovered. These sources are then analyzed by the CMS for their topology and structure before storing them in the list of VCPs. This functionality requires certain upgrades from both the components (CMS and Search agent) and is yet to be implemented. It is further discussed in the 'future work' section of chapter 6.

Each source is defined in the sources metadata file, an XML file that was mentioned earlier in Section 4.5.4. This file contains custom tags to describe each source. A copy of the file with 4 sources defined in it is included as an Appendix (Appendix A). As seen from the sources metadata file, each HTML source is identified by its hompage URL. For example, a data source for hotels called hotelguides is identified using hotel-guides.us, its homepage. More tags like type, content, lastVisitedOn and lastModifiedOn

provide more details about the source. The last 2 tags mentioned above along with the freqOfVisits tag help in deciding the next visit date for each source and the nature of the source.

### 5.1.1 SELECTING RESOURCE INSTANCES FROM A SOURCE

When we decide on the data source, the next step is to decide what resource types present in the source are we interested in. For testing purposes, it was decided to collect all the hotels in a given city. To help with this, an XML file with a list of cities in a state was prepared. A copy of this file cityList.xml with 2 states Alabama and Oklahoma is presented as Appendix I. cityList.xml allows easy addition and deletion of information with the presence of easy to understand tags. A user will eventually input the source name, country and state of interest. The CMS will refer the cityList.xml file to get the list of cities for the country and state mentioned. It then proceeds to collect all hotel instances for each city, one by one, from the mentioned data source. This gathering process if further explained in sections below.

## 5.2 GATHERING & STORING DATA FROM SOURCES

As shown in (Figure 4) below, operations mentioned in Section 4.5.7 can be performed using the MainAdminScreen.

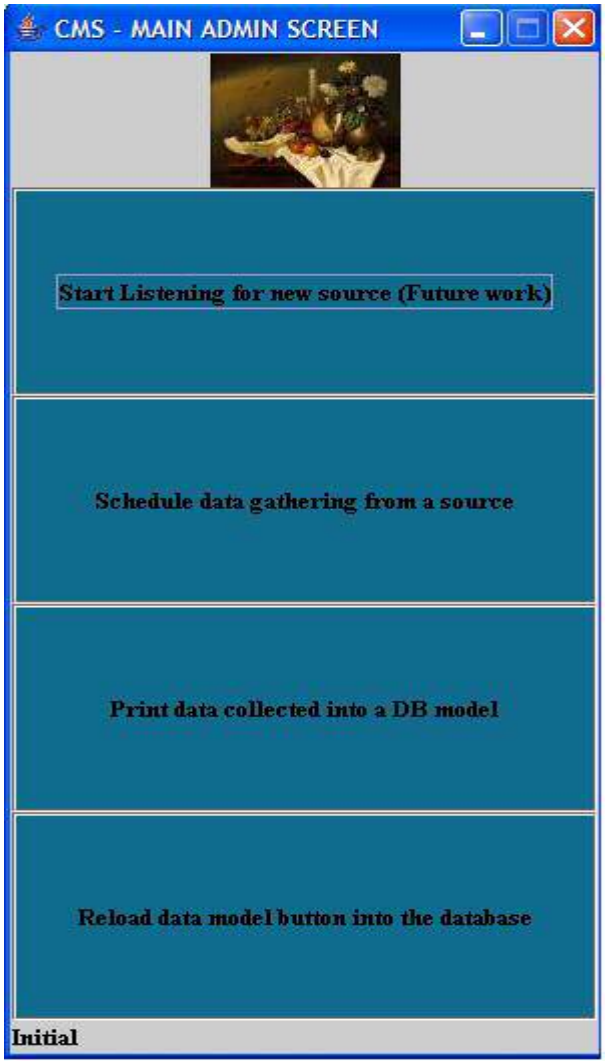



**Figure 4: Main Administrator Screen in CMS**

The process to schedule data gathering/ storing from various data sources is initiated from the source scheduler screen shown in Figure 5 below (Java code in Appendix B). The source from which the data has to be gathered is selected from the list box. Source metadata file discussed in the previous section is used to populate the list box above.

The source metadata file is to keep track of data sources. It is particularly useful to deal with passive information sources mentioned in Section 4.5.1 above. On the other

hand, RSS data feeds does not requires this information as it is dynamic/ active in nature. But in this thesis work, all types of data sources including RSS feeds are considered to be passive and are treated in the same way, i.e., by visiting them at specified intervals. By following this approach any change in data sources are easily coordinated. But, this approach comes with an overhead, i.e., having the CMS to deal with sources that could be handled differently. A solution to this problem is an RSS aggregator that can keep track of RSS feeds from various sources and convey all updates made to them [RSSAggregator, SampleAggregators] . It is further discussed in the future work section.

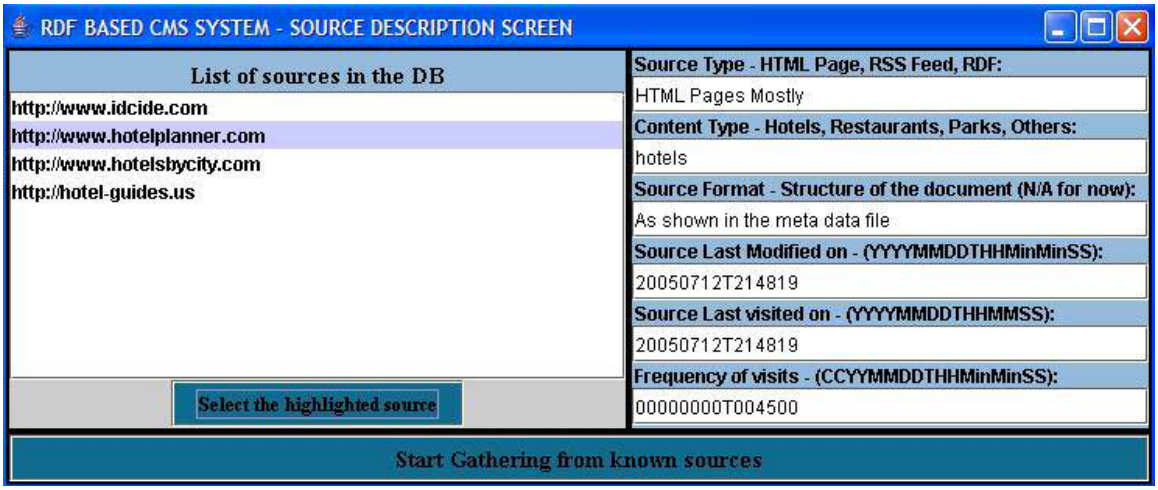



**Figure 5: Source Scheduler Screen**

After selecting the desired source, clicking the 'gather from source' button on the screen will correspond to a series of actions as shown in Figure 6. Simultaneous gathering from multiple sources is implemented using the concept of multi-threading in Java [50]. This functionality is obtained using the Quartz scheduler [34]. For example, the data source 'hotel-planner.com' is a separate 'timer task' (timer task represents a thread in the Quartz scheduler terminology). Each 'timer task' will have a 'job' associated with it. (Again, a

'job' corresponds to a series of activities in the Quartz Scheduler domain). There are values like task name, schedule time, task lifetime etc associated with each timer task that the Quartz scheduler uses while scheduling it.

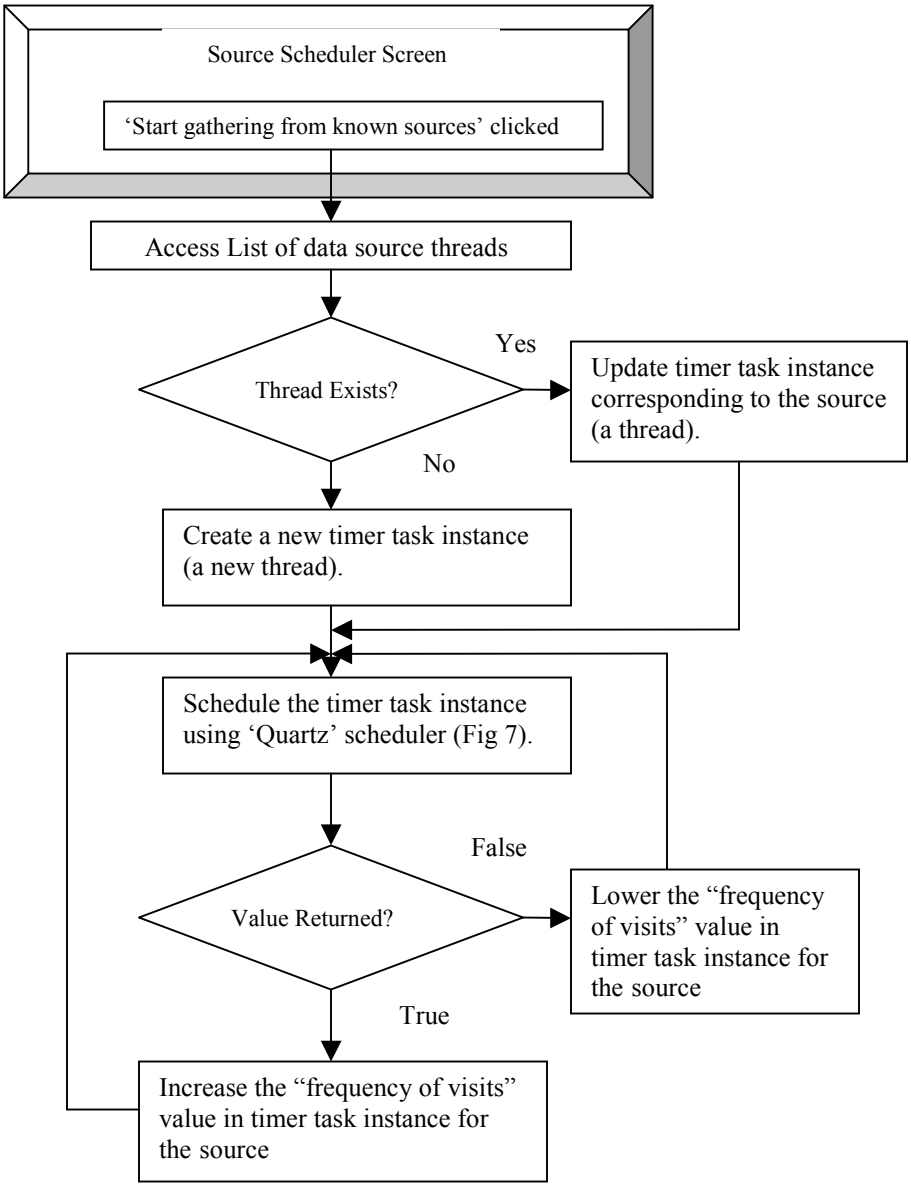



**Figure 6: Process of scheduling a timer task corresponding to a data source**

The following abstract piece of code illustrates how a job is scheduled using the Quartz scheduler (Listing 2).

```
…
// Get an instance of the Quartz scheduler
Scheduler sched = sf.getScheduler();
// Create an instance of the Job (with 2 parameters: 1. Job name, 2: Timer task name
JobDetail job = new JobDetail(“Sample Job 1”, GatherData.class);
// Create a trigger with parameters that include startTime and triggerName.
SimpleTrigger trigger = new SimpleTrigger("Sample Trigger 1" + “Trigger Class 1”, tempSrcType,
startTime, null, 0, 0);
// Scheduling the job using the scheduler (with jobdetails and trigger)
Date ft = sched.scheduleJob(job, trigger);
// starting the job execution
sched.start();
….
```

**Listing 2: Scheduling a job using the Quartz scheduler (Java Code)**

In this thesis, the timer task is labeled ‘PageMaipulator’ whose associated sequence of operations is shown in Figure 7. In figure 6, a task scheduled by the ‘Quartz’ scheduler ‘fires’ at the date and time specified during its initialization (that is with a use of a trigger as shown in Listing 2 above). After the task has finished executing, the date and time for its next execution is obtained from the modified source metadata file. Modifying the metadata file is explained in detail in Section 5.4. Java code that corresponds to PageManipulator timer task is shown in Appendix C.

“PageManipulator” returns a Boolean value of either true or false. As seen from Figures 6 and 7, TRUE is returned when there was an update (i.e. a new resource from the source was stored or values for existing resource were modified) and FALSE otherwise.

## 5.3 IMPLEMENTING WRAPPERS

As mentioned in Section 4.5.3, a wrapper is responsible for gathering and converting data into a desired form. As shown in the architecture diagram above (Figure 2), there are different wrappers for each type of source. For example, a wrapper for an HTML type source will implement mechanisms to load and parse data from web pages. On the other hand, a wrapper for RDF data source has to implement mechanisms for mapping data from the source model to the target model.

In case of HTML wrappers, a scheduled source is visited once it is triggered. The initial page for the source is then loaded using a loader.

Parsing a document largely depends on the structure of it. For example, HTML pages are often semi-structured or not structured at all as mentioned earlier [13]. There were many parsers available on the internet [13, 14]. These parsers were not effective when a defective HTML page (one with missing or jumbled tags) was dealt. On the other hand, Mozilla.org's [24], Gecko DOM Reference section presented a set of DOM APIs that were used in various Mozilla based browsers [25]. These APIs provided a mechanism to access specific elements from an HTML page/ XML document which is usually represented as a tree in the DOM context. As this DOM design/implementation is independent of any programming languages, its structural representation was available from a single, consistent API. To maximize the efficiency of this approach, the structure of the document (HTML tags and data elements) had to be identified beforehand. Consider the sample HTML document below (Listing 3).

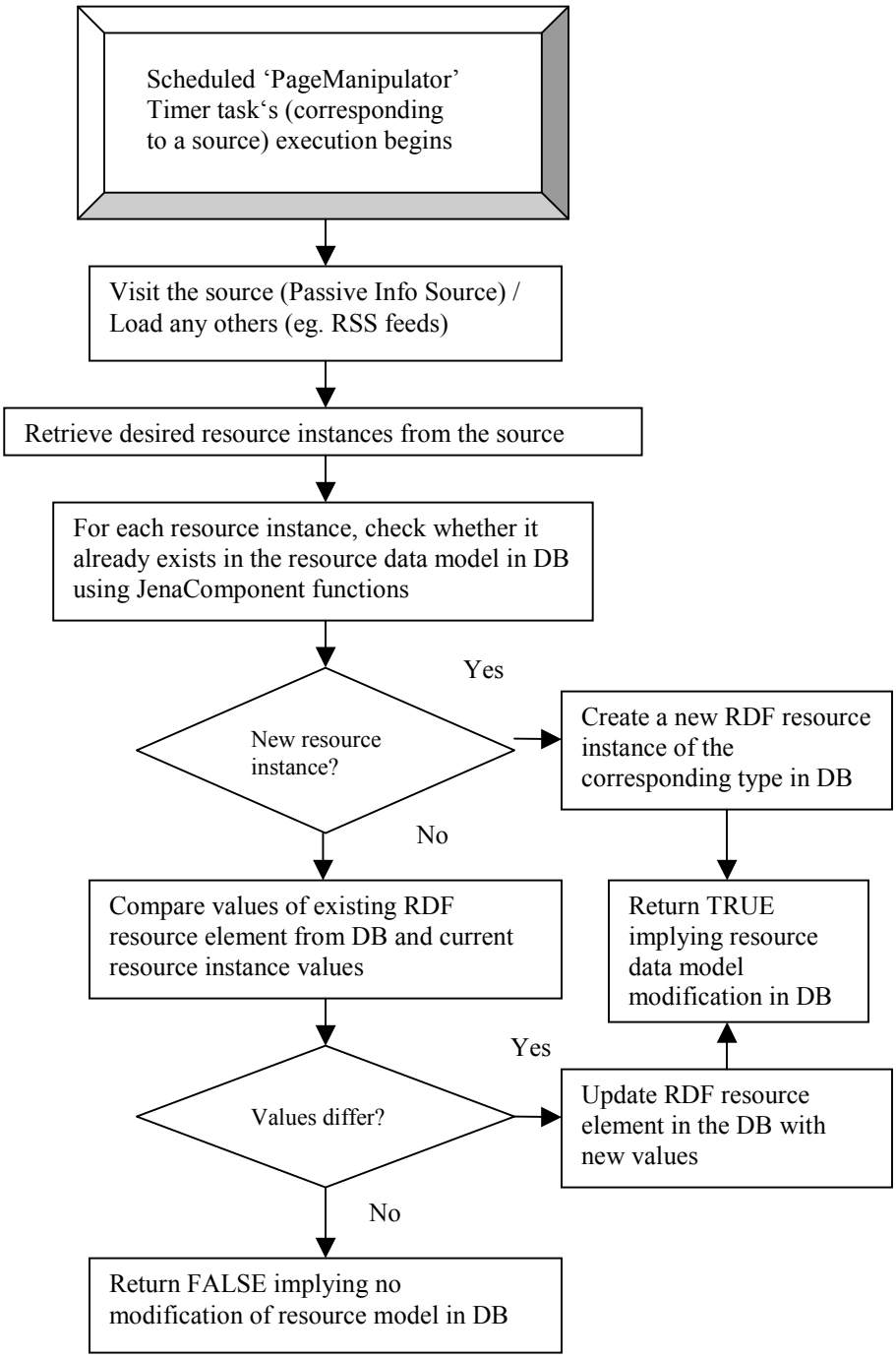



**Figure 7: Sequence of actions associated with the 'PageManipulator' Timer task**

```
<html>
<head> Sample Page <head>
<body>
<area shape="circle" coords="163,140,9" href="http://www.fairfieldinn.com " title="Fairfield Inn" >
<area shape="circle" coords="173,140,9" href="http://www.hamptoninn.com " title="Hampton Inn" >
<area shape="circle" coords="183,140,9" href="http://www.mariottinn.com " title="Mariott Inn" >
</body> </html>
```

**Listing 3: Source code of a simple HTML page**

The code below (Listing 4) shows how a JavaScript function uses Mozilla DOM APIs to access all the area elements and their attributes in the above HTML document.

```
function start() {
  alert("Inside the google local .js file");
  var area1 = document.getElementsByTagName("area");
  alert("Number of AREA elements" + area1.length);
  for(var j = 0; j< area1.length;j++)    {
    var areaAttrs = area1[j].attributes;
    for(var i=0;i< areaAttrs.length;i++) {
     alert("" + area1[j].getAttribute(areaAttrs[i].nodeName.toLowerCase()));
} }}
```

**Listing 4: JavaScript using DOM APIs to access 'area' elements in Listing 3**

The Mozilla DOM API approach had 2 main drawbacks. They were,

a) The HTML page has to be downloaded locally in order to use JavaScript to access data.

b) Writing a JavaScript for each HTML data source was a tedious task. In most cases the scripts were very long and hard to update.

The Cyberneko HTML parser written by Andy Clark addresses the above 2 drawbacks. Also, its inbuilt tag balancer can scan and "fix up" many common problems like missing tag elements, mismatched element tags etc…More information on this parser can be found on the Apache.org page [7]. Information from the parsed pages can then be

extracted using standard XML interfaces. In this case, JAXEN – an XPath Engine for Java was used to extract specific details from the parsed pages [16, 57].

Extracting data using CyberNeko and JAXEN is a 2 stage process: loading the document and using XPath expressions to extract data from loaded pages. The Java code snippet (from Appendix C) below illustrates this process Listing 5.

```
/* Create a DOCUMENT object from the URL of the source page */
// Stage 1: create HTML parser, parse the source and create a DOCUMENT
// object from the parsed data//
org.cyberneko.html.parsers.DOMParser par1 = new org.cyberneko.html.parsers.DOMParser();
String systemId = "URL of a sample page";
XMLInputSource source = new XMLInputSource(null, systemId, null);
par1.parse(source);
Document doc = par1.getDocument();

// Stage 2: Form a XPAth object and extract data from the DOCUMENT
// object formed above//
DOMXPath dxp = new DOMXPath("some Xpath expression");
List results = (List) dxp.evaluate(doc);
```

**Listing 5: Using CyberNeko HTML parser and JAXEN Xpath engine**

A sample HTML page containing the list of hotels for a given city, XPath expression for extracting city names from the HTML page and extracted data are shown in the appendix (Appendix D).

Parsing an RDF document or an RSS feed is relatively simple. As mentioned above, these files have a structure associated with them that allows us to use readymade parsers. On the other hand, Jena component mentioned in Section 4.5.5 is built for handling RDF based files. Thus by providing inbuilt functionalities like model persistence and RDQL query capabilities, Jena proves to be a simple choice for manipulating RDF related documents. This saves us the task of having separate parsers for RDF data sources.

## 5.4 STORING RESOURCE INSTANCES IN A DATABASE

Please refer to Figure 7 above where you can find steps involved in storing a resource instance into the database. Each resource instance created is given a unique identifier to distinguish it in the database. When details about a particular entity instance (say, Mariott Inn of Stillwater, OK, USA) are extracted from a data source by using a corresponding wrapper, a Jena resource instance is created under its name. Later all the additional details like amenities available in that particular hotel are associated with the Jena resource instance created. The abstract Java code (Listing 6) below demonstrates this specific process. Full code available in Appendix F shows all the functions related with the persistent database.

```
// Create a class instance (i.e., a resource instance of resource type 'cl')
//and using the name 'instanceID'
OntModel m = getCustomOntologyModel();
OntClass cl = m.getOntClass( NS + className );
Individual inst = m.createIndividual( instanceID,cl);
// Add properties to the class instance (i.e., add details like 'amenities'
 as properties to the resource instance //created above).
OntProperty p1 = m.getOntProperty(NSpace + propertyName);
inst.addProperty(p1,propertyValue);
```

**Listing 6: Creating a resource instance and adding properties to it**

While creating the class instance, it is checked whether one in the same name already exists in the database. Again Figure 7 above clearly illustrates this procedure. Changes if made are taken into consideration and the next visiting date and time are adjusted accordingly in the source metadata file. Java code snippet below (Listing 7) illustrates

this procedure. Complete code covering functions to work with the source metadata file is given in Appendices G and H.

```
/ /* Update the source metadata file **//
//Get instance of MetaDataFile Interface
MetaDataFileInterface mdFI = new MetaDataFileInterface();
CurrentTime ct = new CurrentTime();
String strDT = ct.getCurrentDate_Time(); // get time in specified format
strDT = strDT.replaceAll(":","");
strDT = strDT.replaceAll(" ","T");
strDT = strDT.replaceAll("-","");

//If there were any changes made to the resource instance from  the
// source under consideration, update specific vaues of that source identified
//by 'srcHomeURL' in the source metadata file
 if(sourceWasModified == true)
   {
    //System.out.println("here 1...firstNode");
     mdFI.updateFreqOfVisitsNodeValue(srcHomeURL);
     mdFI.updateTimeNodeValue(srcHomeURL,"lastVisitedOn",strDT);
     mdFI.updateTimeNodeValue(srcHomeURL,"lastModifiedOn",strDT);
   }
else
if(sourceWasModified == false)
   {
    //System.out.println("here 2...firstNode");
     mdFI.updateTimeNodeValue(srcHomeURL,"lastVisitedOn",strDT);
   }
```

**Listing 7: Update source metadata file**

The source is then rescheduled using Quartz Scheduler with updated parameters from the source metadata file.

## 5.5 QUERYING THE DATABASE

Jena provides few basic functions that can used to perform some top level queries like listing all classes or individuals present in the persistent model in database. More advanced querying is achieved using the RDQL support from Jena [41]. Jena's inference

support can be used by software agents to derive additional information [19] from the data stored as Jena models in the database.

# CHAPTER 6

## CONCLUSION AND FUTURE WORK

This thesis work is an attempt to propose an architecture for the Content Management System to be used within the Agent Based E-Travel system. This architectural framework is responsible for integrating heterogeneous data types available from various sources. Integrating data sources has two main activities associated with it: gathering and storing data. During this process, various existing architectures for integrating data were studied. This study was based on issues like heterogeneity of data sources, ease of use and its compatibility with other components in the Agent Based e-Travel domain. After analyzing these architectures, their advantages and limitation, a customized framework for integrating data was proposed. This architecture is based on an RDF based agent ontology that serves as a common model for integrating data sources. RDF was chosen for its extensibility and for its ability to provide more accurate characterizations of a real world concept with the help of RDFS. The architecture was tested with the help of hotel entity defined in the agent ontology.

## 6.1 FUTURE WORKS

There are lots of extensions possible with this architecture. They are mentioned below.

A definite extension to this work is to establish a communication methodology between the search agent and the content management system. As mentioned in the previous chapters, this extension requires some basic modifications in the CMS architecture i.e. CMS has to be implemented using agent technologies in order to establish communication links (using methods like ACL) with other agents in the E-Travel framework [1, 51].

At this time, all data sources are considered to be passive. This is an overhead when active sources like RSS / ATOM feeds are concerned. In this case, an API associated with a news aggregator can be utilized to look for any updates in the feeds that are registered with the aggregator [35].

Another main work is to implement proper mapping mechanisms that preserve the semantic integrity of data sources. For example, ‘free newspaper’ and ‘complimentary newspaper’ convey the same amenity. Currently, they are stored as different amenities. In this particular type of scenario, mapping mechanisms that covey their relationships are required.

# REFERENCES


[1] – Agent Communication Language Specifications. http://www.fipa.org/repository/aclspecs.html

[2] – Active source, an example. http://www.newsok.com/rss/entertainment.xml

[3] – Andy Nauli. Using Software Agents to Index Data for an E-Travel system. *Thesis work submitted to the Faculty of Graduate college of Oklahoma State University.* August 2003.

[4] – Bhavani Thuraisingham. XML Databases and the Semantic Web, 109 – 115: CRC Press, 2002.

[5] – ChicagoCityNet. A vortal for the city of Chicago, Illinois. http://www.chicagocitynet.com

[6] – CSS - Cascading Style Sheets. http://www.w3.org/Style/CSS/

[7] – Cyberneko: HTML Scanner and Tag Balancer. http://people.apache.org/~andyc/neko/doc/html/index.html

[8] - Data Integration Primer. U.S Department of Transportation. http://www.nasfa.net/OLD%20PAGE/Conference%20and%20Tradeshow/2002/Presentations/alfelor%20handout.pdf

[9] – DOM – Document Object Model. http://www.w3.org/DOM/

[10] - Domenico Beneventano, Sonia Bergamaschi. The MOMIS methodology for integrating Heterogeneous Data sources. http://dbgroup.unimo.it/Momis/

[11] - Dublin core metadata initiative. http://dublincore.org/

[12] – Google Search Engine http://www.google.com

[13] – HTML parser. http://htmlparser.sourceforge.net/

[14] – HTML Swing parser from Sun.
http://java.sun.com/products/jfc/tsc/articles/bookmarks/

[15] – Ian Horrocks. Logical Foundation for the Semantic Web. Reasoning with Expressive Description Logics: Theory and Practice.
http://www.cs.man.ac.uk/~horrocks/Slides/glasgow.pdf

[16] – JAXEN: Universal Java XPATH engine. http://jaxen.org/

[17] – Jena: A semantic web framework for Java. http://jena.sourceforge.net/

[18] – Jena 2 Database Interface. http://jena.sourceforge.net/DB/

[19] – Jena 2 Inference support. http://jena.sourceforge.net/inference/

[20]– Lucas Zamboulis, Alexandra Poulovassilis. XML data integration by Graph Restructuring. http://www.dcs.bbk.ac.uk/~lucas/pubs/Zam04.pdf

[21] – Marcin Paprzycki, Minor Gordon. Designing Agent Based Travel Support System.
http://www.cs.okstate.edu/~marcin/mp/cvr/research/ISPDC_2005.pdf

[22] – Metamend: Search Engine Optimization Experts.
http://www.metamend.com/internet-growth.html

[23] – Minor Gordon, Aleksander Kowalski, Marcin Paprzycki, Tomasz Pelech, Michał Szymczak, Tomasz Wasowicz. *Ontologies in a travel support system*.

[24] – Mozilla Open Source Organization. http://www.mozilla.org/

[25] – Mozilla Gecko DOM Reference. http://www.mozilla.org/docs/dom/domref/

[26] – MSN Portal. http://www.msn.com

[27] – MSN Search Engine. http://search.msn.com/

[28] – MySQL Open Source Database. http://www.mysql.com/

[29] – Oracle Database. http://www.oracle.com/database/index.html

[30] - Oklahoma State University Library – Indexes and Databases.
http://www.library.okstate.edu/database/index.htm

[31] - Patrick Ziegler, Klaus R. Dittrich. User-Specific semantic integration of Heterogeneous data: The SIRUP approach. http://www.ifi.unizh.ch/stff/pziegler/papers/ZieglerICSNW2004.pdf

[32] – Pawel Jan Kalczynski. Software Agents to filter business information from the Internet to the Data Warehouse.

[33] – PostgreSQL Database. http://www.postgresql.org/

[34] – Quartz Enterprise Job Scheduler. http://www.opensymphony.com/quartz/

[35] - Radio's XML-RPC Interface for the Aggregator. http://radio.userland.com/developer/APIs/aggregatorApi

[36] – Rafal Angryk, Violetta Grant, Minor Gordon, and Marcin Paprzycki. Travel support system – agent based framework. In Hamid R. Arabnia and Youngsong Mun, editors, *Proceedings of the International Conference on Internet Computing*, IC'2002, Las Vegas, Nevada, USA, June 24-27, 2002, volume 3, pages 719-725, CSREA Press.

[37] – RDF: Resource Description Framework. http://www.w3.org/RDF/

[38] – RDF Primer. http://www.w3.org/TR/rdf-primer/#rdfmodel

[39] – RDFS: RDF Schema. http://www.w3.org/TR/rdf-schema/

[40] – RDQL: A Query Language for RDF. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

[41] – RDQLJena: A programmer's introduction to RDQL (Jena Tutorial). http://jena.sourceforge.net/tutorial/RDQL/

[42] – Richard Vdovjak, Geert-Jan Houban. RDF Based architecture for Semantic Integration of Heterogeneous Information sources. http://wwwis.win.tue.nl/~houben/respub/wiiw01.pdf

[43] - Ronaldo dos Santos Mello, Carlos Alberto Heuser. A Bottom-Up Approach for integration of XML sources. *International workshop on Information Integration on the web – Rio de Janeiro, Brazil – 2001*. http://www.cos.ufrj.br/wiiw/papers/16-Ronaldo_Melo(26).pdf

[44] – RDF Site Summary - http://web.resource.org/rss/1.0/spec

[45] – RSSAggregator. http://en.wikipedia.org/wiki/News_aggregator

[46] – Sample News Aggregators. http://en.wikipedia.org/wiki/List_of_news_aggregators

[47] – School-Libraries.org http://www.school-libraries.org/

[48] – Stefan Decker, Frank van Harmelen, Jeen Broekstra, Michael Erdmann, Dieter Fensel, Ian Horrocks, Michael Klien, Sergey Melnik. The Semantic Web – On the respective roles of XML and RDF. http://www.ontoknowledge.org/oil/downl/IEEE00.pdf

[49] – Sematic Web http://www.w3.org/2001/sw/

[50] – Threads: Doing Two or More tasks at once
http://java.sun.com/docs/books/tutorial/essential/threads/

[51] – UMBC's Description on Agent Communication Languages
http://agents.umbc.edu/technology/acl.shtml

[52] - Universal Resource Identifiers
http://www.w3.org/Addressing/URL/URI_Overview.htm

[53] – Webopedia. Online Computer Dictionary for Computer and Internet Terms and Definitions http://www.webopedia.com/

[54] – Widom J. Integrating Heterogeneous Databases: Lazy or Eager? ACM Computing Surveys 28A(4), December 1996, invited position paper
http://dbpubs.stanford.edu:8090/pub/1996-37

[55] – The free encyclopedia. http://www.wikipedia.org/

[56] – XML: Extensible markup language. http://www.w3.org/XML/

[57] – XPATH: XML Path Language. http://www.w3.org/TR/xpath

[58] – XSLT: Extensible Stylesheet language family. http://www.w3.org/Style/XSL/

[59] – XUP: XML User Interface Protocol - http://www.w3.org/TR/2002/NOTE-xup-20020528/

[60] – Yahoo portal. http://www.yahoo.com/

[61] – Yahoo Search. http://search.yahoo.com/

# Appendix A

## Java code for SourceMetadataFile

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sourceDescriptions[<!ATTLIST source id ID #REQUIRED>]>

 <sourceDescriptions>
                                        <!--   IDCIDE.COM -->

<source id="http://www.idcide.com"> <!-- idcide.com -->
 <hotels format="allSmall" initialPartURL="/hotels/StateAbbreviation/CityName.htm" spaceFiller="-">
<!-- Directly to list of hotels in city -->
<XPathExpressions> <!-- X Path Expressions to be used for extracting details using JAXEN -->
hotelsListPage hotelLinkXP="//DIV[@id='base']//H4/following-sibling::A/@href"
hotelNameXP="//DIV[@id='base']//H4/following-sibling::A/text()"/>
<hotelDetailsPage hotelAddressXP="//H3[text() = 'Location']//following-sibling::P/text()"
hotelAmenitiesXP="//H3[text() = 'Features']//following-sibling::DIV//B//following-sibling::text()"/>
 </XPathExpressions>
 </hotels>
 <type> HTML Pages</type>
 <content> hotels </content>
 <format> As shown in the meta data file</format>
 <lastModifiedOn>20050712T214951</lastModifiedOn>
 <lastVisitedOn>20050712T214951</lastVisitedOn>
 <freqOfVisits>00000000T004500</freqOfVisits>
</source>

                                        <!--   HOTELPLANNER.COM -->

<source id="http://www.hotelplanner.com"> <!-- hotelPlanner.com -->
 <hotels format="N/A" initialPartURL="/Hotels-Directory/Country-
CountryAbbreviation/CountryName/CountryName-Hotels-Directory.html" spaceFiller="-"> <!-- Directly
to list of states in a country -->
<XPathExpressions> <!-- X Path Expressions to be used for extracting details using JAXEN -->
<statesListPage stateLinkXP="//H4[text() = 'Select a State for your Group']//following-
sibling::TABLE//A/@href" stateNameXP="//H4[text() = 'Select a State for your Group']//following-
sibling::TABLE//A/text()"/>
<citiesListPage cityLinkXP="//H4[text() = 'Select a City for your Group']//following-
sibling::TABLE//A/@href" cityNameXP="//H4[text() = 'Select a City for your Group']//following-
sibling::TABLE//A/text()"/>
```

<hotelsListPage hotelLinkXP="//H1/following-sibling::TABLE[5]//TR//A[1]/@href"
hotelNameXP="//H1/following-sibling::TABLE[5]//TR//A/B/text()"/>
<hotelDetailsPage hotelAddressXP="//B[text() = 'Address']//following-sibling::text()"
hotelAmenitiesXP="//LI/text()"/>
</XPathExpressions>
</hotels>
<type> HTML Pages Mostly </type>
<content> hotels </content>
<format> As shown in the meta data file</format>
<lastModifiedOn>20050712T214819</lastModifiedOn>
<lastVisitedOn>20050712T214819</lastVisitedOn>
<freqOfVisits>00000000T004500</freqOfVisits></source>

<!-- HOTELSBYCITY.COM -->

<source id="http://www.hotelsbycity.com"> <!-- hotelsByCity.com -->
<hotels format="N/A" initialPartURL="/all-us-states" spaceFiller="_"> <!-- Directly to list of states in a
country -->
<XPathExpressions> <!-- X Path Expressions to be used for extracting details using JAXEN -->
<statesListPage stateLinkXP="//TABLE[@class = 'linktable']//A/@href" stateNameXP="//TABLE[@class
= 'linktable']//A/text()"/>
<citiesListPage cityLinkXP="//TABLE[@class = 'linktable'][1]//A/@href"
cityNameXP="//TABLE[@class = 'linktable'][1]//A/text() | //TABLE[@class =
'linktable'][1]//A//B/text()"/>
<hotelsListPage hotelLinkXP="//TABLE[@class = 'hotelinfo'][1]//TH[1]//A/@href"
hotelNameXP="//TABLE[@class = 'hotelinfo'][1]//TH[1]//A/text()"/>
<hotelDetailsPage hotelAddressXP="//DIV[@class = 'detail-address']/text()"
hotelAmenitiesXP="//H1[text() = 'Amenities:']/parent::DIV/following-
sibling::TABLE[1]//TR//TD/text()"/>
</XPathExpressions>
</hotels>
<type> HTML Pages Mostly </type>
<content> hotels </content>
<format> As shown in the meta data file</format>
<lastModifiedOn>20050712T163341</lastModifiedOn>
<lastVisitedOn>20050712T164925</lastVisitedOn>
<freqOfVisits>00000000T004500</freqOfVisits>
</source>

<!-- HOTEL-GUIDES.US -->

<source id="http://hotel-guides.us"> <!-- hotel-guides.us -->
<hotels format="N/A" initialPartURL="/usa-hotels.html" spaceFiller="-"> <!-- Directly to list of states in a
country -->
<XPathExpressions> <!-- X Path Expressions to be used for extracting details using JAXEN -->
<statesListPage stateLinkXP="//TD[@valign = 'top']//FONT[@size = '-1']//A/@href"
stateNameXP="//TD[@valign = 'top']//FONT[@size = '-1']//A/text()"/>
<citiesListPage cityLinkXP="//DIV[@align = 'center'][1]//A/@href" cityNameXP="//DIV[@align =
'center'][1]//A/text()"/>
<hotelsListPage hotelLinkXP="//LI/A[1]/@href" hotelNameXP="//LI/A[1]/text()"/>
<hotelDetailsPage hotelAddressXP="//SPAN[@class = 'mediumpagetitle']/parent::TD//text()"
hotelAmenitiesXP="//SPAN[text()= 'Services/Facilities/Amenities:']/parent::TD//LI/text()"/>
</XPathExpressions>
</hotels>
<type> HTML Pages Mostly and Web pages </type>
<content> hotels </content>
<format> As shown in the meta data file</format>
<lastModifiedOn>20050712T193141</lastModifiedOn>
<lastVisitedOn>20050712T193125</lastVisitedOn>

<freqOfVisits>00000000T030000</freqOfVisits>
</source>
</sourceDescriptions>

# Appendix B

## Java code for SourceSchedulerScreen

```
/*
                    *  SOURCE SCHEDULER SCREEN *
 *  This java class performs the following functionalities:
 *  1. Creates the source scheduler screen by extending the JFrame class
 *  2. Provides facilities for the CMS administrator to schedule an existing source for data
gathering
 */

package CMS;

// importing general Java libraries
import java.io.*;
import java.util.*;

// importing Swing and AWT related libraries
import javax.swing.*;
import javax.swing.JComponent;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

// Importing data related Java libraries
import java.util.*;
import java.util.Date;
import java.util.Calendar;
import java.util.GregorianCalendar;


// Importing Quartz Scheduler related libraries
import org.quartz.CronTrigger;
import org.quartz.JobDetail;
import org.quartz.Scheduler;
import org.quartz.SchedulerFactory;
import org.quartz.SchedulerMetaData;
import org.quartz.SimpleTrigger;
import org.quartz.helpers.TriggerUtils;
import org.quartz.JobListener;

import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;
```

```
class SourceScheduler extends JFrame implements ActionListener
{
        // Declaring source meta data file
        String srcMetaDataFile = "DataSource_Metadata_Modified.xml";

        // Declaring the WEST panel in the frame and the components inside
        JLabel srcListLbl = new JLabel("List of sources in the DB");
        JPanel srcListLblPanel = new JPanel();


        JPanel srcsListPanel = new JPanel();
        JScrollPane scrollPane1; // Scrollpane 1 for listing all sources
        JList sourceList = new JList();
        DefaultListModel sourceListModel = new DefaultListModel();
        JButton selectSrcBtn = new JButton("Select the highlighted source");

        // Declaring the CENTER panel and the components inside
        JPanel srcLabelValuePanel = new JPanel();
        JLabel srcLinkLabel = new JLabel("Source Link - URL Of the source:");
        // will go as the resource URI
        JTextField srcLinkText = new JTextField();
        JLabel srcTypeLabel = new JLabel("Source Type - HTML Page, RSS Feed, RDF:");
        JTextField srcTypeText = new JTextField();
        JLabel srcContentLabel = new JLabel("Content Type - Hotels, Restaurants, Parks,
        Others:");
        JTextField srcContentText = new JTextField();
        JLabel srcFormatLabel = new JLabel("Source Format - Structure of the document (N/A
        for now):");
        JTextField srcFormatText = new JTextField();
        JLabel srcLastModifiedLabel = new JLabel("Source Last Modified on -
        (YYYYMMDDTHHMinMinSS):");
        JTextField srcLastModifiedText = new JTextField();
        JLabel srcLastVisitLabel = new JLabel("Source Last visited on -
        (YYYYMMDDTHHMMSS):");
        JTextField srcLastVisitText = new JTextField();
        JLabel srcFreqVisitLabel = new JLabel("Frequency of visits -
        (CCYYMMDDTHHMinMinSS):");
        JTextField srcFreqVisitText = new JTextField();

        //Declaring the SOUTH panel in the frame
        JPanel buttonPanel1 = new JPanel(new GridLayout(1,1)); //start gathering button
        JButton gatherSourcesBtn = new JButton("Start Gathering from known sources");

        /*****************************************
         *Construtor: Initialize all the panel elements
         *****************************************/

         public SourceScheduler()
         {
                //Title for screen
                super("RDF BASED CMS SYSTEM - SOURCE DESCRIPTION SCREEN");

                //Setting Frame Layout
                getContentPane().setLayout(new BorderLayout());
```

```
// Adding components to srcLableValuePanel and the srcLableValuePanel to the
frame

// CENTER PANE - srcLabelValuePanel
srcLabelValuePanel.setLayout(new
BoxLayout(srcLabelValuePanel,BoxLayout.PAGE_AXIS));
srcLabelValuePanel.add(srcTypeLabel);
srcLabelValuePanel.add(srcTypeText);
srcLabelValuePanel.add(srcContentLabel);
srcLabelValuePanel.add(srcContentText);
srcLabelValuePanel.add(srcFormatLabel);
srcLabelValuePanel.add(srcFormatText);
srcLabelValuePanel.add(srcLastModifiedLabel);
srcLabelValuePanel.add(srcLastModifiedText);
srcLabelValuePanel.add(srcLastVisitLabel);
srcLabelValuePanel.add(srcLastVisitText);
srcLabelValuePanel.add(srcFreqVisitLabel);
srcLabelValuePanel.add(srcFreqVisitText);
srcLabelValuePanel.setPreferredSize(new Dimension(600,500));
srcLabelValuePanel.setVisible(false);
srcLabelValuePanel.setBorder(BorderFactory.createLineBorder
(Color.black, 2));
Color clr = new Color(149,186,217);
srcLabelValuePanel.setBackground(clr);
srcLabelValuePanel.setFont(new Font("Serif", Font.BOLD,15));
getContentPane().add(srcLabelValuePanel,BorderLayout.CENTER);

// Adding components to model_srcsListPanel and model_srcsListPanel to the
frame
// WEST PANE - srcsListPanel

srcsListPanel.setLayout(new BoxLayout(srcsListPanel,
BoxLayout.PAGE_AXIS));

setLblProps(srcListLbl,15,srcListLblPanel,400,20);
srcsListPanel.add(srcListLblPanel);
scrollPane1 = new JScrollPane(sourceList);
scrollPane1.setPreferredSize(new Dimension(400,180));
sourceList.setSelectedIndex(0);
srcsListPanel.add(scrollPane1);
setBtnProps(selectSrcBtn,12,this,400,30);
selectSrcBtn.setAlignmentX(Component.CENTER_ALIGNMENT);
srcsListPanel.add(selectSrcBtn);
selectSrcBtn.setEnabled(true);

getContentPane().add(srcsListPanel,BorderLayout.WEST);
srcsListPanel.setPreferredSize(new Dimension(400,500));
srcsListPanel.setVisible(true);
srcsListPanel.setBorder(BorderFactory.createLineBorder (Color.black, 2));

// Adding the lower panel (button) panel to the frame
// SOUTH PANE - buttonPanel1

setBtnProps(gatherSourcesBtn,15,this,1000,30);
gatherSourcesBtn.setEnabled(false);
buttonPanel1.add(gatherSourcesBtn);
```

```
                buttonPanel1.setBorder(BorderFactory.createLineBorder (Color.black, 2));
                getContentPane().add(buttonPanel1,BorderLayout.SOUTH);

                //Setting Screen properties
                setSize(1000,600);
                setVisible(true);
                //setResizable(false);

                // Adding listener for windows
                addWindowListener (
                        new WindowAdapter() {
                                public void windowClosing(WindowEvent e) {
                                        System.exit(0);
                                 }} );

                // Initializing the source list with source names from the metadata file
                MetaDataFileInterface mdFI = new MetaDataFileInterface();
                String sourceNames = mdFI.getSourceNamesFromFile("source");
                System.out.println("SOurce:------>" + sourceNames);
                StringTokenizer sTokr = new StringTokenizer(sourceNames,"*");
                sourceListModel.removeAllElements();

                while(sTokr.hasMoreTokens() == true){
                sourceListModel.addElement(sTokr.nextToken().trim()); }
                sourceList.setModel(sourceListModel);
        }// SourceScheduler constructor

public void setBtnProps(JButton btn,int fontSize, ActionListener obj, int x, int y)
{
                Color clr = new Color(16,107,141);
                btn.setFont(new Font("Serif", Font.BOLD,fontSize));
                btn.setPreferredSize(new Dimension(x,y));
                btn.setBackground(clr);
                btn.addActionListener(obj);
}


public void setLblProps(JLabel lbl, int fontSize, JPanel lblPnl, int x, int y)
{
                Color clr = new Color(149,186,217);
                lbl.setFont(new Font("Serif", Font.BOLD,fontSize));
                lblPnl.setBackground(clr);
                lblPnl.setPreferredSize(new Dimension(x,y));
                lblPnl.add(lbl);
}

public void actionPerformed(ActionEvent e)
        {
                //If the button to select a source from list is clicked//
                if(e.getSource() == selectSrcBtn)
                {
                        handleSelectSource();
                }

                //If the button to gather data from a source is clicked//
                if(e.getSource() == gatherSourcesBtn)
```

```
                {
                        handleGatherSource();
                }
        }




        /*******************************************************
                Event handlers for
                *   1. Select source button
                *   2. Gather from source button
         *******************************************************/

public void handleSelectSource()
{
                 //System.out.println("Inside handleSelectSource()");
                srcLabelValuePanel.setEnabled(true);
                srcLabelValuePanel.setVisible(true);

  try
  {
                MetaDataFileInterface mdFI = new MetaDataFileInterface();
                String vals[] = mdFI.getSourceDetails("" + sourceList.getSelectedValue());
                srcTypeText.setText(vals[0]) ;
                srcContentText.setText(vals[1]);
                srcFormatText.setText(vals[2]);
                srcLastModifiedText.setText(vals[3]);
                srcLastVisitText.setText(vals[4]);
                srcFreqVisitText.setText(vals[5]);
  }
  catch(Exception e1)
  { System.out.println("Exception occured while selecting a source" + e1); }
 gatherSourcesBtn.setEnabled(true);
}

public void handleGatherSource()
{
        try
        {
                MetaDataFileInterface mdFI = new MetaDataFileInterface();
                String vals[] = mdFI.getSourceDetails("" + sourceList.getSelectedValue());
                // vals[]'s format =
                {"type","content","format","lastModifiedOn","lastVisitedOn","freqOfVisits"};

                /*
                Steps to schedule a job using Quartz scheduler
                1. Create a SCHEDULER
                2. Create the JOBDETAIL for the job to be scheduled
                3. Create the TRIGGER for the job
                4. Associate JobListener/Trigger Listener with the job and trigger respectively(if
                needed)
                5. Schedule the job (using the scheduler)
                */
```

```
    // Creating a scheduler
    SchedulerFactory sf = new org.quartz.impl.StdSchedulerFactory();
    Scheduler sched = sf.getScheduler();
    sched.addJobListener(new MyJobListener("Listener :" +
    sourceList.getSelectedValue()));


  // Creating a jobDetail
  String jobName = vals[1] + "*" + sourceList.getSelectedValue() + "*" + "United States"
  + "*" + "us" + "*" + "Alabama" + "*" + "al" + "*";

  JobDetail job = new JobDetail(jobName, vals[1], PageManipulator_Modified.class);

  // Creating a Trigger
  //extracting fields from lastVisitedOn field
  int yyyy = Integer.parseInt(vals[4].substring(0,4));
  int mm   = Integer.parseInt(vals[4].substring(4,6));
  int dd   = Integer.parseInt(vals[4].substring(6,8));
  int hh   = Integer.parseInt(vals[4].substring(9,11));
  int min  = Integer.parseInt(vals[4].substring(11,13));
  int ss   = Integer.parseInt(vals[4].substring(13,15));

  // Setting values to the Calendar data type
  java.util.Calendar cal = new java.util.GregorianCalendar(yyyy,(mm-1) ,dd );
  // month starts with January = 0. Feb = 1,...
  cal.set(cal.HOUR, hh);
  cal.set(cal.MINUTE, min);
  cal.set(cal.SECOND, ss);
  cal.set(cal.MILLISECOND, 0);

  Date startTime = cal.getTime();
  System.out.println("Previously visited on : " + startTime);

  // Creating a trigger with the above time values
   SimpleTrigger trigger = new SimpleTrigger(
  "Trigger :" + sourceList.getSelectedValue(),
   vals[0],startTime, null, 0, 0);

  // Associating a listener with the job
  //MyJobListener mjl = new MyJobListener("Listener :" + sourceList.getSelectedValue());
  job.addJobListener("Listener :" + sourceList.getSelectedValue());


  // Scheduling the job using the scheduler (with jobdetails and trigger)
     Date ft = sched.scheduleJob(job, trigger);
     System.out.println("Job name is: " + job.getName());
     //System.out.println("Ready for scheduling...");
     sched.start();
     //System.out.println("Scheduled...");
  }
  catch(Exception e)
  {

  }
}
```

```
}

// Defining the class that listens to events associated with a scheduled job//
class MyJobListener implements JobListener
{
String name;
public MyJobListener(String name)
   { this.name = name; }

   public String getName()
   { return name; }

   public void jobToBeExecuted(JobExecutionContext context)
   {
      System.err.println(".......(" + name + "): jobToBeExecuted: "
            + context.getJobDetail().getFullName());
   }

   public void jobWasExecuted(JobExecutionContext context,JobExecutionException
   jobException)
   {
     System.err.println(".......(" + name + "): jobWasExecuted: "
       + context.getJobDetail().getFullName());
    try
     {
       String s1 = context.getJobDetail().getName();
       StringTokenizer sTokr = new StringTokenizer(s1,"*");
       String vals[] = new String[sTokr.countTokens()];
       int index = 0;
       while(sTokr.hasMoreTokens())
           {
            vals[index++] = sTokr.nextToken();
           }

      MetaDataFileInterface mdFI = new MetaDataFileInterface();
      String srcVals[] = mdFI.getSourceDetails("" + vals[1]);
                // srcVals[]'s format =
          {"type","content","format","lastModifiedOn","lastVisitedOn","freqOfVisits"};

      /*
          1. Get all the details from the context (like jobname, triggername, groupname and the
          date.time it was scheduled
          2. Calculate the next visit date (using the above details and freqOfVisits data from the
          DBase
          3. Create a new trigger with the 'next visit date' value
          4. Reschedule the job using the new trigger
      */

          System.out.println("Prev Scheduled Time was : " +
          context.getTrigger().getPreviousFireTime());
       GregorianCalendar cal1 = new GregorianCalendar();
       cal1.setTime(context.getTrigger().getPreviousFireTime());

       int cc1 = Integer.parseInt(srcVals[5].substring(0,2));
       int yy1 = Integer.parseInt(srcVals[5].substring(2,4));
       int mm1 = Integer.parseInt(srcVals[5].substring(4,6));
```

```
            int dd1 = Integer.parseInt(srcVals[5].substring(6,8));
            int hh1 = Integer.parseInt(srcVals[5].substring(9,11));
            int min1 = Integer.parseInt(srcVals[5].substring(11,13));
            int sec1 = Integer.parseInt(srcVals[5].substring(13,15));




                // adding the values from freqOfVisits field to the srcLastVisit field
                    cal1.add(Calendar.YEAR, yy1);
                    cal1.add(Calendar.MONTH, mm1);
                    cal1.add(Calendar.DATE, dd1);
                    cal1.add(Calendar.HOUR, hh1);
                    cal1.add(Calendar.MINUTE, min1);
                    cal1.add(Calendar.SECOND, sec1);

                     // Create a new trigger
                    Date startTime = cal1.getTime();
                    System.out.println("Updated Time is : " + startTime);
                    System.out.println("Job Name is     : " + context.getJobDetail().getName());
                    SimpleTrigger trigger = new SimpleTrigger("Trigger :" +
                   context.getJobDetail().getName() , srcVals[1], ""+context.getJobDetail().getName(),
                    srcVals[1], startTime, null, 0, 0);

                   // Reschedule the job with the updated date details
                   context.getScheduler().rescheduleJob(context.getTrigger().getName(),
                   context.getTrigger().getGroup(),trigger);
               } //try
               catch(Exception e)
                 {
                    System.out.println("Exception occured in : jobWasExecuted() of SourceScheduler");
                 }
        }

        public void jobExecutionVetoed(JobExecutionContext context)
        {
            // TODO Auto-generated method stub
        }
}
```

# Appendix C

## Java code for PageManipulator

```
/*                    PAGE MANIPULATOR

   This class implements the Job and Runnable Interface. Job Interface as defined in the
   "Quartz scheduler" domain is the actual task that needs to be schedules. In this case,
   it is the actual parsing of data from data sources in order to store them in the DB.
 */

package CMS;

// Importing General Usage Java libraries
import java.util.*;
import java.util.regex.*;

// Importing Cyberneko parser related libraries
import org.cyberneko.html.HTMLConfiguration;
import org.cyberneko.html.filters.ElementRemover;
import org.cyberneko.html.parsers.*;

// Importing Apache Xerces related libraries
import org.apache.xerces.xni.parser.XMLDocumentFilter;
import org.apache.xerces.xni.parser.XMLInputSource;
import org.apache.xerces.xni.parser.XMLParserConfiguration;
import org.apache.xerces.parsers.*;

// Importing JAXP & JAVA DOCUMENT related libraries
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;

// Importing JAXEN related libraries
import org.jaxen.dom.DOMXPath;

// Importing DOM related libraries
import org.w3c.dom.*;

// Importing Jena related libraries
```

```
import com.hp.hpl.jena.db.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;

// Importing quartz scheduler related libraries
import java.util.Date;
import org.quartz.Job;
import org.quartz.JobExecutionContext;
import org.quartz.JobExecutionException;


public class PageManipulator_Modified implements Job, Runnable
{
        Thread t;
        String tName;


        //Customizing the Cyberneko Parser
    org.cyberneko.html.filters.Writer writer =
      new org.cyberneko.html.filters.Writer();

  //setup filter chain
    XMLDocumentFilter[] filters =
    {
         writer,
    };

  // create HTML parser
  org.cyberneko.html.parsers.DOMParser par1 = new org.cyberneko.html.parsers.DOMParser();

  // InitialFullURL - URL string of the initial page for each source
  String initialFullURL = "";

  // A string variable to keep track of the URL of the page that is dealt at a given time
  private String currentWorkingURL = "";

  // A string variable to store the sourceHomeURL
  private String sourceHomeURL = "";

  // A boolean variable to indicate whether a source was modified or not
  boolean sourceWasModified = false;

  public void PageManipulator_Modified()
  {

        t = new Thread(this);
    try{
    //par1.setProperty("http://cyberneko.org/html/properties/filters", filters);
    par1.setFeature("http://cyberneko.org/html/features/balance-tags",true);
    }
    catch(Exception e) {
        System.out.println("Exception in PageManipulator Contructor:" + e);
    }
  }
```

```
	public void run()
	{
	 try
	 {
		//this.startThread(this.tName);
	 }
	 catch(Exception e)
	 {

	 }
	}


	/*
	 This function is called when the task(job) corresponding to a data source and
	 scheduled by the Quartz scheduler is triggered. This function initiates a series
	 of actions by calling the initiatePM() function
	*/
	public void execute(JobExecutionContext context) throws JobExecutionException
		{
		  try
		  {
			this.tName = context.getJobDetail().getName();

			String s1 = context.getJobDetail().getName();
			StringTokenizer sTokr = new StringTokenizer(s1,"*");
			String vals[] = new String[sTokr.countTokens()];
			int index = 0;
			while(sTokr.hasMoreTokens())
			{
				vals[index++] = sTokr.nextToken();
			}
			this.initiatePM(vals[0],vals[1],vals[2],vals[3],vals[4],vals[5]);
		  }
		  catch(Exception e)
		  {
			System.out.println("Exception occured in GatherData");
		  }
		}

	/*
	 This function initiates the process of page manipulation for a given source.
	 Details corresponding to the source are read from the meta data  file and
	 are used to parse the source pages to extract desired content.
	 As and when a node elements corresponding to the source becomes available,
	 they are handled using specified handlers
	*/
	public void initiatePM(String category, String srcHomeURL, String countryName, String
countryAbbr, String stateName, String stateAbbr)
	 {

	  sourceHomeURL = srcHomeURL;
	  currentWorkingURL = srcHomeURL;
```

```
String XPEs[][] = null;


// Results array returned from loadPages()
String resultsFromloadPagesArr[][] = null;
try
{
 boolean starter = true;
 MetaDataFileInterface mdFI = new MetaDataFileInterface();
 NodeList nl2 = mdFI.getNodeListForSource(srcHomeURL,category);
 NamedNodeMap nnoMap1 = mdFI.getCategoryAttributes(srcHomeURL,category);
      for(int i=0;i<nl2.getLength(); i++)
      {
            Node nd4 = nl2.item(i);

            if(nd4.getNodeName() == "citiesListPage")  // to reach page with list of cities
                  {
                   resultsFromloadPagesArr = handleCitiesListNode(nd4,
                  starter,srcHomeURL,nnoMap1,countryName,countryAbbr,stateName,
                  resultsFromloadPagesArr);
                     if(starter == true)
                       starter = false;
                   }

                   if(nd4.getNodeName() == "statesListPage")
                         // to reach page with list of states
                   {
                         resultsFromloadPagesArr =
                         handleStatesListNode(nd4,starter,srcHomeURL,nnoMap1,cou
                         ntryName,countryAbbr);
                         if(starter == true)
                           starter = false;
                   }

                   if(nd4.getNodeName() == "hotelsListPage")
                         // to get page with list of hotels in a city
                   {

                         handleHotelsListNode(nd4,starter,srcHomeURL,category,nno
                         Map1,countryName,countryAbbr,stateName,stateAbbr,results
                         FromloadPagesArr);
                    if(starter == true)
                      starter = false;
                   }
            }
      }//try
      catch(Exception e)
      {
            System.out.println("Exception occured while forming a Document
            initiatePM1");
      }
}// initiatePM fn()
```

```
/*
 Set Initial URL data to its complete value
*/

public void setInitialFullURL(String temp)
{
        initialFullURL = temp;
}

/*
 Get complete value of the InitialURL
 */
public String getInitialFullURL()
{
        return(initialFullURL);
}


/*
 This function handles the hotelsListPage in the source metadata file.
 */
        public void handleHotelsListNode(Node nd4, boolean isFirstNode,String srcHomeURL,
        String category, NamedNodeMap  nnoMap1, String countryName, String countryAbbr,
        String stateName, String stateAbbr, String[][] prevResults) throws Exception
{
        String[][] tempResults = null;
        NamedNodeMap nnoMap2 = nd4.getAttributes(); // Get the xPathExpressions for
        hotelsListNode
        String xps[] = new String[nnoMap2.getLength()];
        for(int i1=0;i1<nnoMap2.getLength();i1++)
        {
        Xps[i1] = nnoMap2.item(i1).getNodeName() + "-->" +
        nnoMap2.item(i1).getNodeValue();
        }

        // Get the list of cities from the citylist.xml file
                String [] cities = null;
                String [] argumentArr = null;
                cities = getCities(countryAbbr,stateAbbr);

                if(isFirstNode == true) // If this is the intial node for this source
                 {

                  /**            MODIFICATION  A
                   *
                   *    To gather hotel details for all cities....
                   *    replace 'for' loop statement below
                   *    with the following string....
                   *
                   *    for(int j =0;j<cities.length;j++)
                   *
                   */
                for(int j =0;j<1;j++)
```

```
                {
                initialFullURL = "" + formInitialFullURL(srcHomeURL, nnoMap1,
                countryName, countryAbbr, stateName, stateAbbr, cities[j]);
                        tempResults = loadPages(initialFullURL,xps,1,argumentArr);
                        for(int mm=0;mm<tempResults.length;mm++)
                        {
                        handleHotelDetailsNode(initialFullURL, countryAbbr, stateAbbr,
                        cities[j], tempResults[mm][1], tempResults[mm][0],srcHomeURL,
                        category);
                        }
                  }
                }

                else
                if(isFirstNode == false)
                 {

              /**            MODIFICATION  B
                   *
                   *    To gather hotel details for all cities....
                   *    replace 'for' loop statement below
                   *    with the following string....
                   *
                   *     for(int m =0;m<cities.length;m++)
                   *
                   */

                for(int m= 0;m<3;m++)//to iterate thru list of cities
                //for(int m =0;m<cities.length;m++)
                {
                        String t1 = "";
                        for(int n=0;n<prevResults.length;n++)
                        {
                        System.out.println("**"  + prevResults[n][1].toLowerCase());
                        System.out.println("**"  + cities[m].toLowerCase());


                        if(prevResults[n][1].toLowerCase().indexOf(cities[m].toLowerCase())!
                        =-1)
                         {
                                t1 = "" + prevResults[n][0];
                                String tempURL = "" + setCurrentWorkingURL(t1);
                                tempURL = tempURL.replaceAll(" ","");
                                tempResults = loadPages(tempURL,xps,1, argumentArr);
                                for(int mm=0;mm<tempResults.length;mm++)
                                {
                                handleHotelDetailsNode(initialFullURL, countryAbbr,
                                stateAbbr, cities[m], tempResults[mm][1],
                                tempResults[mm][0],srcHomeURL, category);
                                }
                                }
                                }
                        }
}

// Update the metadata file if the value of sourceWasModified is true
```

```
                    MetaDataFileInterface mdFI = new MetaDataFileInterface();
                    CurrentTime ct = new CurrentTime();
                    String strDT = ct.getCurrentDate_Time();
                    strDT = strDT.replaceAll(":","");
                    strDT = strDT.replaceAll(" ","T");
                    strDT = strDT.replaceAll("-","");

                if(sourceWasModified == true) // update all values: lastVisited
                                    // lastModified and freqOfVisits

                    {
                            mdFI.updateFreqOfVisitsNodeValue(srcHomeURL,2);
                            mdFI.updateTimeNodeValue(srcHomeURL,"lastVisitedOn",strDT);
                            mdFI.updateTimeNodeValue(srcHomeURL,"lastModifiedOn",strDT);
                    }
                    else
                    if(sourceWasModified == false) // update lastVisited and freqOfVisits
                    {
                      mdFI.updateFreqOfVisitsNodeValue(srcHomeURL,1);
                            mdFI.updateTimeNodeValue(srcHomeURL,"lastVisitedOn",strDT);
                    }

}

/*
 This function handles the citiesListNode()
*/

        public String[][] handleCitiesListNode(Node nd4, boolean isFirstNode,String
        srcHomeURL, NamedNodeMap  nnoMap1, String countryName, String countryAbbr,
        String stateName, String[][] prevResults) throws Exception
{
            String tempResults[][] = null;
                    NamedNodeMap nnoMap2 = nd4.getAttributes();
                    // Get the xPathExpressions for citiesListNode
                    String xps[]  = new String[nnoMap2.getLength()];
                    // initializing 2nd dim of XPEs array
                    for(int i1=0;i1<nnoMap2.getLength();i1++)
                    {
                            xps[i1] = nnoMap2.item(i1).getNodeName() + "-->" +
                            nnoMap2.item(i1).getNodeValue();
                    }

                    if(isFirstNode == true) // if this is the first node for this source
                    {
                             String tempURL = "" + formInitialFullURL(srcHomeURL, nnoMap1,
                            countryName, countryAbbr, "", "", "");
                      String [] argumentArr = null;
                      setInitialFullURL(tempURL);
                      tempResults = loadPages(initialFullURL,xps,1,argumentArr);
                    }
                    else
                    {
                            String t1 = "";
                            String [] argumentArr = null;
                            for(int k=0;k<prevResults.length;k++)
```

```
                        {
                                if(prevResults[k][1].toLowerCase().indexOf(stateName.toLow
                                erCase()) != -1)
                                {
                                        t1 = "" + prevResults[k][0];
                                        String tempURL = "" + setCurrentWorkingURL(t1);
                                        tempResults =
                                        loadPages(tempURL,xps,1,argumentArr);
                                }
                        }
                }
        return(tempResults);
}


/*
 This handles the statesListPage node in the source metadata file
*/
        public String[][] handleStatesListNode(Node nd4, boolean isFirstNode, String
        srcHomeURL, NamedNodeMap  nnoMap1, String countryName, String countryAbbr)
        throws Exception
{
   String tempResults[][] = null;
   String argumentArr[] = null;
                NamedNodeMap nnoMap2 = nd4.getAttributes();
                // Get the xPathExpressions for statesListNode
                String xps[] = new String[nnoMap2.getLength()];
                for(int i1=0;i1<nnoMap2.getLength();i1++)
                {
                        xps[i1] = nnoMap2.item(i1).getNodeName() + "-->" +
                nnoMap2.item(i1).getNodeValue();
                }

                if(isFirstNode == true) // if this is the first node for this source
                {
                        String tempURL =  "" + formInitialFullURL(srcHomeURL, nnoMap1,
                        countryName, countryAbbr, "", "", "");
                        tempResults = loadPages(tempURL,xps,1,argumentArr);
                }
                return(tempResults);
}


/*
 This handles the hotelDetailsPage node of source metadat file
 */


 public void handleHotelDetailsNode(String initialFullURL, String countryAbbr, String stateAbbr,
String cityName, String hotelName, String hotelPartURL, String srcHomeURL, String category)
throws Exception
   {
     MetaDataFileInterface mdFI = new MetaDataFileInterface();
     NodeList nl1 = mdFI.getNodeListForSource(srcHomeURL,category);

     String tempResults[][] = null;
```

```
for(int i=0;i<nl1.getLength(); i++)
  {
   Node nd4 = nl1.item(i);
   if(nd4.getNodeName() == "hotelDetailsPage")
                              {
                                        NamedNodeMap nnoMap2 = nd4.getAttributes();
                                        // Get the xPathExpressions for page2
                                        String XPEs[] = new String[nnoMap2.getLength()];
                                        for(int i1=0;i1<nnoMap2.getLength();i1++)
                                        {
                                                  XPEs[i1] = nnoMap2.item(i1).getNodeName() + "--
                                        >" +  nnoMap2.item(i1).getNodeValue();
                                        }

                              String tempURL = setCurrentWorkingURL(hotelPartURL);

                              // Visit the exact URL page of the hotel description
                                 // Use countryAbbr, hotelName ... to form a unique
                                 //Id for the hotel instance in the DB
                                   String tempStr[] = { category, srcHomeURL, countryAbbr, stateAbbr,
                                   cityName, hotelName};
                                   tempResults = loadPages(tempURL,XPEs,2,tempStr);
                                   }
        }
      }


/*
 This function  returns an URL that acts the basic URL by which the source is contacted first
 Only from the pages all other links (direct and indirect) to hotel details pages
 are gathered
 */

        public String formInitialFullURL(String srcHomeURL, NamedNodeMap nmap,String
        cnName, String cnAbbr, String stName, String stAbbr, String ciName )
           {
        Node ndSpace = nmap.getNamedItem("spaceFiller"); // part of URL
        String spaceFiller = ndSpace.getNodeValue();
        // part of initial page to be loaded for each source

        cnName = cnName.replaceAll(" ",spaceFiller);
        stName = stName.replaceAll(" ",spaceFiller);

        Node ndFormat = nmap.getNamedItem("format"); // part of URL
        String format = ndFormat.getNodeValue();
        // part of initial page to be loaded for each source

        if(format.compareTo("allSmall") == 0)
         {
                  cnName = cnName.toLowerCase();
                  stName = stName.toLowerCase();
                  stAbbr = stAbbr.toLowerCase();
                  ciName = ciName.toLowerCase();
         }

        Node ndPartURL = nmap.getNamedItem("initialPartURL"); // part of URL
```

```
        String initialPartURL = ndPartURL.getNodeValue();
        // part of initial page to be loaded for each source


                    initialPartURL = initialPartURL.replaceAll("CountryName",cnName);
                    initialPartURL = initialPartURL.replaceAll("CountryAbbreviation",cnAbbr);

                    initialPartURL = initialPartURL.replaceAll("StateName",stName);
                    initialPartURL = initialPartURL.replaceAll("StateAbbreviation",stAbbr);
                    initialPartURL = initialPartURL.replaceAll("CityName",ciName);

    currentWorkingURL = "" + sourceHomeURL + initialPartURL;
    return(currentWorkingURL);
}


/*
 This function is used to get list of cities from the cityList.xml file
 cityList.xml is a flat file that has sample cities listed for few states in USA.
 PageManipulator_Modified class gathers details of all hotels for all cities
 listed in the xml file from the chosen source.
 */

public String[] getCities(String countryID, String stateID)
{
 String cities[] = null;
 try
 {
 String xmlFile = "citylist.xml";
 Document doc = formDocument(xmlFile);
 Element srcElt = doc.getElementById(countryID); // Get the country element
 NodeList nl1 = srcElt.getElementsByTagName("State");

 for(int i=0;i<nl1.getLength();i++)
 {
        NamedNodeMap nnoMap1 = nl1.item(i).getAttributes();
   Node nd1 = nnoMap1.getNamedItem("id"); // Node from the Attribute nodes list

   if(nd1.getNodeValue().compareTo(stateID) == 0)
    {
        NodeList nl2 = nl1.item(i).getChildNodes();
        int cityCount = 0;
        for(int j=0;j<nl2.getLength(); j++)
        {
                if(nl2.item(j).getNodeName() == "City")
                  cityCount++;
        }

        cities = new String[cityCount]; int c = 0;

        for(int j=0;j<nl2.getLength(); j++)
        {
                if(nl2.item(j).getNodeName() == "City")
                 {
                        NodeList nl3 = nl2.item(j).getChildNodes();
                        String t1 = nl3.item(0).getNodeValue() ;
```

```
                        t1 = t1.toLowerCase();
                        cities[c++] = t1.trim();

                    }
                }

            }
        }
       }
       catch(Exception e)
        {
            System.out.println("Exception occured in getCities of PageManipulator" + e);
        }
        return(cities);
    }



    /*
     Function to form an DOCUMENT object given an XML source file
     */
    public Document formDocument(String xmlSourceFile) throws Exception
    {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document document = builder.parse(xmlSourceFile);
            return(document);
    }

/* This function sets the variable CurrentWorkingURL to the value of URL that is currently
Worked upon */
  public String setCurrentWorkingURL(String relativeURL)
  {
     System.out.println("Current Working URL ==>" + currentWorkingURL);
     System.out.println("Relative URL ==>" + relativeURL);

     relativeURL = relativeURL.replaceAll("\\s","");

     String temp = relativeURL;
     String tempWorkingURL = "";
     String tempTopURL = "";

     temp = temp.replaceAll("href","");
     temp = temp.replaceAll("=","");
     temp = temp.replaceAll("\"","");
     //temp = temp.replaceAll("\\s","");

     // get the toplevel URL part from the currentWorkingURL
     String REGEX = "http://.*\\." + ".*" + "\\." + "[^htl]{3}+";
     String INPUT = currentWorkingURL;
     Pattern pattern = Pattern.compile(REGEX);
     Matcher matcher = pattern.matcher(INPUT);
     while(matcher.find())
     {
         tempTopURL = matcher.group();
          int i = tempTopURL.lastIndexOf("/");
```

```
      if(i > 6)
        tempTopURL = tempTopURL.substring(0,i);
}

// Find part of reltiveURL to be concatenated to form absoluteURL
String temp2 = temp;

// Count number of preceding dots in relativeURL
String temper  = "";
if(temp.indexOf("/") != -1)
   temper = temp.substring(0,temp.indexOf("/"));
int dotCount = 0; int sPos = -1;
while(true)
{
  sPos = temper.indexOf(".",sPos);
  if(sPos == -1)
    break;
  else
   {
     dotCount++;
   sPos++;
   }
}

if(relativeURL.toLowerCase().indexOf("http") != -1)
            {
                  tempWorkingURL = relativeURL;
            }
else
if(dotCount > 0)
            {
            temp2 = temp2.substring(temp2.indexOf("/")+1,temp2.length());
            // Use number of dots in relativeURL to find a part for absoluteURL from
              // categoryHomeURL
            int p1 = -1;
            for(int i =0;i<dotCount+2;i++)
            {
                  p1 =  currentWorkingURL.indexOf("/",p1);
                  p1++;
            }
            // Concatenate parts from relativeURL and categoryHomeURL to form
              // absoluteURL
            String s1 = currentWorkingURL.substring(0,p1);
            s1 = s1 + temp2;
            tempWorkingURL =  s1;
            }
else
            if(relativeURL.indexOf("/") == 0)
            {
              tempWorkingURL =  tempTopURL + relativeURL;
            }

else
//if(relativeURL.indexOf("/") == -1)
  {
```

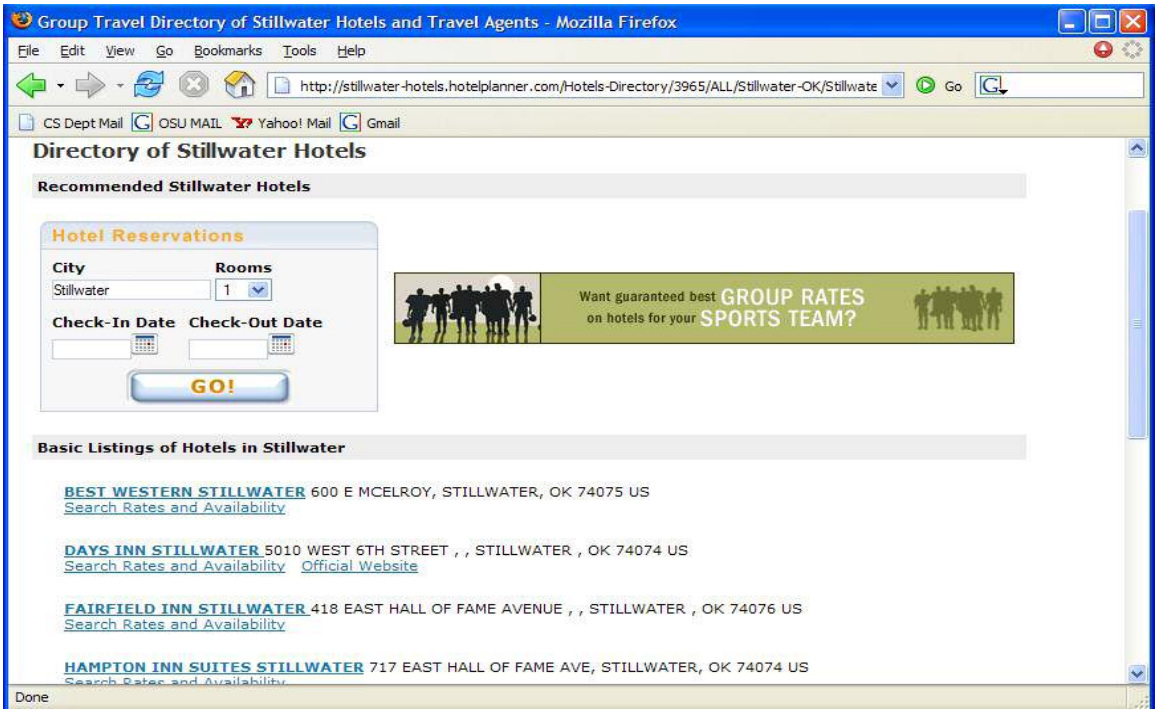
```
            tempWorkingURL =
            currentWorkingURL.substring(0,currentWorkingURL.lastIndexOf("/")+1) +
            relativeURL;
        }

      currentWorkingURL =  "" + tempWorkingURL;
      return(tempWorkingURL);
}

/*
 This function is used to Load a given URL using Cyberneko HTML parse
 and extract data from the loaded pages using JAXEN
*/
            public String[][] loadPages(String url, String[] xps,int page1page2, String[] tempValues)
            throws Exception
{

    System.out.println("URL of the page thatz gonna be loaded is:  " + url);

    String systemId = url;
    XMLInputSource source = new XMLInputSource(null, systemId, null);
    par1.parse(source);
    Document doc = par1.getDocument();
    String tempXps[] = new String[xps.length];
    for(int i=0;i<tempXps.length;i++)
    {
         tempXps[i] = xps[i];
    }

    try
     {
        // Find the total number of xpath expressions
        // Declare variables required for parsing
        int noXps = xps.length;
        DOMXPath dxp[] = new DOMXPath[noXps];
        List results[] = new List[noXps];
        ListIterator l_iter[] = new ListIterator[noXps];

        // Initialize and Populate variables required during parsing
         int index = 0;

        for(int i=0;i<noXps;i++)
        {
                    if(xps[i].indexOf("-->") != -1)
           index = xps[i].indexOf("-->")+3;
        xps[i] = xps[i].substring(index, xps[i].length());
        dxp[i] = new DOMXPath(xps[i]);
        results[i] = (List) dxp[i].evaluate(doc);
                    l_iter[i] = results[i].listIterator();
                      }

        String returnValues[][] = new String[results[0].size()][noXps];

        if(page1page2 == 1) {
        for(int i =0; i<results[0].size(); i++) {
                    for(int j = 0;j<noXps; j++) {
```

```
                        returnValues[i][j] = "" + l_iter[j].next();
                                }}

            for(int i=0;i<results[0].size();i++) {
                        for(int j=0;j<noXps;j++) {
                        if(returnValues[i][j].indexOf("href=") !=-1) {
                                returnValues[i][j] = returnValues[i][j].replaceAll("href=","");
                                returnValues[i][j] = returnValues[i][j].replaceAll("\"","");
                        }
                        else
                         System.out.print("\t" + returnValues[i][j]);
                }
            }
}
else
if(page1page2 == 2) // Implies that results of parsing has to be stored in the DB
{
            String clN = "", NS = "", insID = "", resN = "", propName = "";
            if (tempValues[0].compareTo("hotels") == 0)
             {
                        clN = "Hotel";
                        NS = "http://www.agentlab.com/schemas/hotel.rdf#";
             }
                        insID =
                        tempValues[1]+"_"+tempValues[2]+"_"+tempValues[3]+"_"+tempVal
                        ues[4];
             resN = tempValues[5].replaceAll("text","");
             resN = resN.replaceAll(" ","");
             insID = insID + "_"+ resN;

            PersistentOntology po = new PersistentOntology();
            Individual inst1 = po.createClassInstance(NS, clN,insID);

            // We now have the properties to be assigned for each instance created above
            for(int i=0;i<noXps;i++)
             {
                        String tempStr = "";
                        while(l_iter[i].hasNext() == true)
                         {
                                tempStr = tempStr + "" + l_iter[i].next();
                         }

                         if(tempXps[i].toLowerCase().indexOf("address") != -1)
                         {
                                propName = "address";
                                NS = "http://www.agentlab.com/schemas/location.rdf#";
                         }
                         else
                         if(tempXps[i].toLowerCase().indexOf("amenities") != -1)
                         {
                                propName = "hotelAmenity";
                                NS = "http://www.agentlab.com/schemas/hotel.rdf#";
                         }

                        boolean result =
                        po.addPropertyValuesToInstance(inst1,NS,propName,tempStr);
```

```
                                if(result == true)
                                    sourceWasModified = true;
                            }
                        }
                        return(returnValues);
                    }
                    catch (Exception e)
                    {
                        System.out.println("Jaxen Exception:"  + e);
                        String dummy[][] = new String[1][1];
                        return(dummy);
                    }
                } // Fn() loadPages
            } // CLASS PageManipulator
```

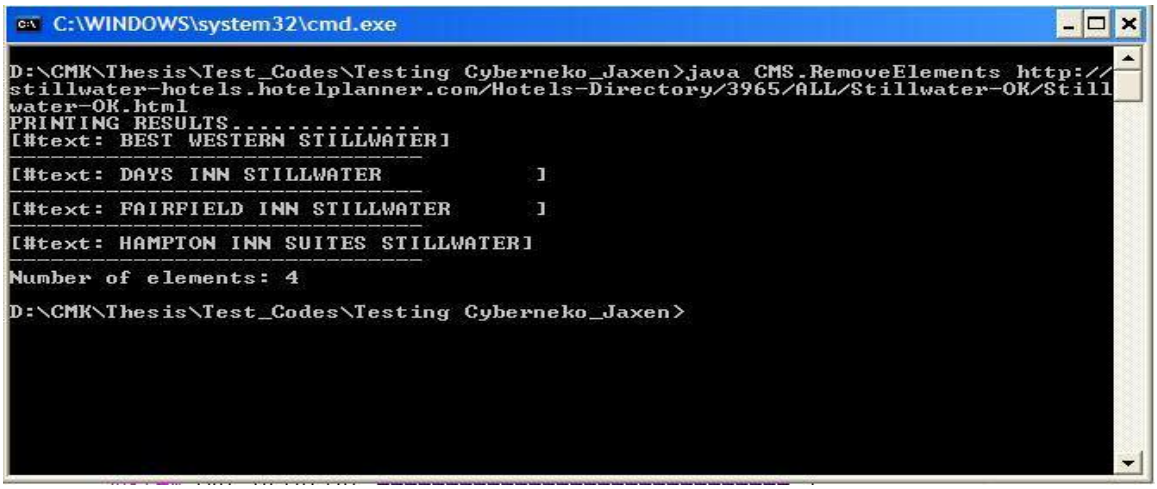
# Appendix D

Sample HTML page with list of cities, XPath expression and results




*XPath Expression to extract Hotel Name from the above page:*
//H1/following-sibling::TABLE[5]//TR//A/B/text ()

*Results from applying the above Xpath Expression using JAXEN on the Document object are shown in the screen shot below. This object is formed by parsing the HTML page above using CyberNeko HTML.*


C:\WINDOWS\system32\cmd.exe
D:\CMK\Thesis\Test_Codes\Testing Cyberneko_Jaxen>java CMS.RemoveElements http://stillwater-hotels.hotelplanner.com/Hotels-Directory/3965/ALL/Stillwater-OK/Stillwater-OK.html
PRINTING RESULTS.....
[#text: BEST WESTERN STILLWATER]
[#text: DAYS INN STILLWATER ]
[#text: FAIRFIELD INN STILLWATER ]
[#text: HAMPTON INN SUITES STILLWATER]
Number of elements: 4
D:\CMK\Thesis\Test_Codes\Testing Cyberneko_Jaxen>

# Appendix E

## cityList.xml file

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE Places[
<!ATTLIST Country id ID #REQUIRED>
<!ATTLIST State id ID #REQUIRED>
]>

<Places>
<Country id = "us" name = "United States of America">
<State id = "ok" name = "Oklahoma" >
<City> Ada </City>
<City> Agra </City>
<City> Ardmore </City>
<City> Bartlesville </City>
<City> Broken-Arrow </City>
<City> Chickasha </City>
<City> Claremore </City>
<City> Del-City </City>
<City> Edmond </City>
<City> Elk-City </City>
<City> Enid </City>
<City> Guthrie </City>
<City> Lawton </City>
<City> McAlester </City>
<City> Muskogee </City>
<City> Norman </City>
<City> Oklahoma-City </City>
<City> Okmulgee </City>
<City> Perry </City>
<City> Ponca-City </City>
<City> Sand-Springs </City>
<City> Sapulpa </City>
<City> Shawnee </City>
<City> Stillwater </City>
<City> Tulsa </City>
<City> Yukon </City>

</State>
<State id = "al" name = "Alabama" >
<City> Auburn </City>
<City> Dothan </City>
</State>
</Country>
</Places>
```

# Appendix F

## Java code for PersistentOntology

```
/*

   This class provides an interface for the CMS to store data as Jena Models into the
   database. The database used is MySQL Server.
 */

// Package
package CMS;

// Import general Java libraries
import java.io.*;
import java.util.*;

// Import Jena related libraries
import com.hp.hpl.jena.db.*;
import com.hp.hpl.jena.ontology.*;
import com.hp.hpl.jena.rdf.model.*;


public class PersistentOntology
{
   // Initializing the database parameters
    public static final String DB_URL = "jdbc:mysql://localhost/cmk_cms";
    public static final String DB_USER = "CMSAGENT";
    public static final String DB_PASSWD = "CMSPASS";
    public static final String DB = "MySQL";

   // Naming the ontology model for storing the resource instances
    public static final String ONT_CMK = "world";

   //  Initializing the variable with the Hotels namespace
   String HotelNS = "http://www.agentlab.com/schemas/hotel.rdf#";

    /*
     This function is used to read the specified model present in the database
     into a file name passed as an argument
    */

    public void readSchema_DataModel(String schemaName, String opFileName)
    {
        FileOutputStream fostream; PrintStream p;
         try{
              fostream = new FileOutputStream(opFileName);
               p = new PrintStream(fostream);
```

```
        ModelMaker maker = getMaker();

        //open a previously created model
        Model prvModel = maker.openModel(schemaName);
        // list the statements in the Model
        StmtIterator iter = prvModel.listStatements();

        while (iter.hasNext())
        {
           Statement stmt      = iter.nextStatement();  // get next statement
           Resource  subject   = stmt.getSubject();     // get the subject
           Property  predicate = stmt.getPredicate();   // get the predicate
           RDFNode   object    = stmt.getObject();      // get the object
           System.out.print(subject.toString());

           System.out.print(" " + predicate.toString() + " ");
           p.print(subject.toString());
           p.print(" " + predicate.toString() + " ");

           if (object instanceof Resource)
           {
             System.out.print(object.toString());
             p.print(object.toString());
           }
           else
           {
              // object is a literal
              System.out.print(" \"" + object.toString() + "\"");
              p.print(" \"" + object.toString() + "\"");
           }
           System.out.println(" .");
           System.out.println("-------------------------------------------------------------------------");
           p.println(" .");
           p.println("-------------------------------------------------------------------------");
         }
       }
           catch(Exception e){System.out.println("There is an exception:" + e);}
} // read dataModel

/*
 This function reloads the model into the database. By this, it removes an old model
 with the same name(if one exists) and stores a new copy. The data for the model
 is read from the file name specified. This file name is the actual ontology
*/

protected OntModel reloadDB()
{
   ModelMaker maker = getMaker();
   // clear out the old
   if (maker.hasModel( ONT_CMK )) maker.removeModel( ONT_CMK );
   // create a spec for the new ont model
   OntModelSpec spec = new OntModelSpec( OntModelSpec.RDFS_MEM );
   spec.setModelMaker( maker );

   // create the base model as a persistent model
   Model base = maker.createModel( ONT_CMK );
```

```
    OntModel m = ModelFactory.createOntologyModel( spec, base );
   // Use the persistent model created in the DB via ModelMaker as the base model

    // tell m where to find the content for ontology
    m.getDocumentManager().addAltEntry( ONT_CMK, "file:world.rdf" );
   m.read( "file:world_Original.rdf" );
    return(m); }
/* This function lists the classes available in the model */

protected void listClasses()
{
    OntModel m = getCustomOntologyModel();

     for (Iterator i = m.listClasses(); i.hasNext(); ) {
       OntClass c = (OntClass) i.next();
       System.out.println( "Class " + c.getURI() );
     }
}

/* This function lists the properties specific to a class */
protected void listClassProperties(String NS, String className)
{
        OntModel m = getCustomOntologyModel();
        OntClass hotel = m.getOntClass( NS + className );

        for (Iterator i = hotel.listDeclaredProperties(); i.hasNext(); )
         {
          OntProperty c = (OntProperty) i.next();
          System.out.print( c.getLocalName() + " " );
         }
}

/* This function creates a class instance based on a specified class with the given
 namespace and name */

protected Individual createClassInstance(String NS, String className, String instanceID)
{
            OntModel m = getCustomOntologyModel();
            OntClass cl = m.getOntClass( NS + className );
            Individual inst = m.createIndividual( instanceID,cl);
            return(inst);
 }

/*
 This function adds values to properties of an instance of a class.
 It checks whether the given property wit the given value already exists.
 If it is true, it simple false. Otherwise, it returns TRUE
 denoting that the model was updated. This inturn means there were
 new data available from the source
 */

protected boolean addPropertyValuesToInstance(Individual inst, String NSpace,String
propertyName, String propertyValue )
{
        OntModel m = getCustomOntologyModel();
        OntProperty p1 = m.getOntProperty(NSpace + propertyName);
```

```
propertyValue = propertyValue.replaceAll(" ","");
propertyValue = propertyValue.replaceAll(":","");
propertyValue = propertyValue.replaceAll("text","");

if(inst.hasProperty(p1,propertyValue) == false)  {
            if(propertyName.compareTo("hotelAmenity") == 0)   {
          try{
           //OntResource range = (OntResource) p1.getRange();
           //OntClass rangeCl = (OntClass) range.as(OntClass.class);
          //OntResource instanceValsArr[] =
          printIndividuals(NSpace,rangeCl.getLocalName());
          //System.out.println("Hotel Amenity value is:==>" + propertyValue);
          inst.addProperty(p1,propertyValue);
            }
           catch(Exception e1) {
          System.out.println("OntProfile exception" + e1); }
         }
         else {
            inst.addProperty(p1,propertyValue); }
           return(true);
       }
      else {
            System.out.println("A property with the same value already exists");
            return(false);
     }
}

/*
 This function prints the individuals of a class or all the individuals
 present in the model
 */

protected OntResource[] printIndividuals(String NSpace, String className)
{
   OntResource returnValArr[] = null;
 try
 {
          OntModel m = getCustomOntologyModel();
          OntClass cl = null;
          boolean classNameIsNull = true;
          Iterator itr = null;

          if(className == "") // Implies list all individuals in the model
          {
                    itr = m.listIndividuals();
          }
          else // List only individuals of the given class name
          {
                    cl = m.getOntClass( NSpace+className);
                    itr = cl.listInstances();
                    classNameIsNull = false;
          }

// count no of individuals and initialize returnValues array accordingly
    int noIndi = 0;
```

```
            while(itr.hasNext())
            {
               noIndi++;
               itr.next();
            }
            returnValArr = new OntResource[noIndi];

            // reinitialize iterators
            if(classNameIsNull == false)
              itr = cl.listInstances();
            else
              itr = m.listIndividuals();

            // populate the returnValues array
            int index = 0;
            while(itr.hasNext())
            {
              int tempI = index;
              returnValArr[index++] = (OntResource)itr.next();
            }
      }
      catch(Exception e)
       {
          System.out.println("Exception in printIndividuals of PersistentOntology" + e);
       }

    // return the results
    return(returnValArr);
  }

   /* This function prints the values corresponding to a property for the given
  Individual */

protected void printPropertyValuesOfIndividuals(Individual ind, OntProperty p)
{
     // Form a statementIterator over the list of properties.
     // If a null value is passed for p, list all propeies and their values
     // for the specified individual
     StmtIterator iter = null;
     if(p == null)
        iter = ind.listProperties();
    else
      iter = ind.listProperties(p);

     while (iter.hasNext()) {
                Statement stmt      = iter.nextStatement();  // get next statement
                Resource  subject   = stmt.getSubject();     // get the subject
                Property  predicate = stmt.getPredicate();   // get the predicate
                RDFNode   object    = stmt.getObject();      // get the object

                if (object instanceof Resource) {
                   System.out.println(object.toString());
                }
                else {
                    System.out.println(" \"" + object.toString() + "\"");
                      }  }   }
```

```
/*This function returns the model object created using custom  parameters*/

 protected OntModel getCustomOntologyModel()
 {
  OntModelSpec spec = new OntModelSpec( OntModelSpec.RDFS_MEM );
  spec.setModelMaker( getMaker() );
        OntModel m = ModelFactory.createOntologyModel( spec, getMaker().createModel(
        ONT_CMK ) );
  return m;
 }

 /* Returns an instance of the model maker class used to
  create a persistent model in the database */

 protected ModelMaker getMaker()
 {
    try {
       // Load the Driver
       String className = "com.mysql.jdbc.Driver";
       Class.forName(className);

       // Create database connection
       IDBConnection conn  = new DBConnection
         ( DB_URL, DB_USER, DB_PASSWD, DB );

       // Create a model maker object
       return ModelFactory.createModelRDBMaker(conn);
    }
    catch (Exception e) {
       e.printStackTrace();
       System.exit( 1 );
    }
  return null;
 }
}
```

# Appendix G

## Java code for MetadataFileInterface

```
/*
   This class basically deals with the MetaData file. A meta data file is where
   all the details about a source are stored. MetaData file is an XML file
   describing each source with fields like type, content, last visited etc...
 */

package CMS;

// Importing Apache Xerces related libraries
import org.apache.xerces.xni.parser.XMLDocumentFilter;
import org.apache.xerces.xni.parser.XMLInputSource;
import org.apache.xerces.xni.parser.XMLParserConfiguration;
import org.apache.xerces.parsers.*;

// Importing JAXP & JAVA DOCUMENT related libraries
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.xml.sax.SAXException;

// To write a DOM to a file
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.FileOutputStream;

// Importing JAXEN related libraries
import org.jaxen.dom.DOMXPath;

// Importing DOM related libraries
import org.w3c.dom.*;

// Importing general Java libraries
import java.io.*;


class MetaDataFileInterface
{
   // Declaring instance variables
   String  metaDataFile = "DataSource_Metadata_Modified.xml";
```

```
public String getSourceNamesFromFile(String nodeName)
 {
   String sourceNames = "";
   try
   {
     // Forming a document object of the sourceMetadata file
     Document doc = formDocument(metaDataFile);
     NodeList nl1 = doc.getElementsByTagName(nodeName);
     int noNodes = nl1.getLength();
     System.out.println("Number of nodes..." + noNodes);
     for(int i =0;i<noNodes+1;i++)
     {
        Node nd = nl1.item(i);
        NamedNodeMap nmap = nd.getAttributes();
        Node id = nmap.getNamedItem("id");
        sourceNames = sourceNames + id.getNodeValue() + "*";
        //System.out.println("Source name ..." + sourceNames);
     }
   }
   catch(Exception e)
   {
        System.out.println("Exception occured in: getNodeListForSource()");
   }
   return(sourceNames);
 }

 /*This function will return details corresponsing to a source.
   The name of the source is passed as the parameter.
   Details are retunred as a string array
 */
 public String[] getSourceDetails(String sourceName)
 {

        String[] nodeNames =
        {"type","content","format","lastModifiedOn","lastVisitedOn","freqOfVisits"};

        String[] returnVals = new String[nodeNames.length];
             try
                        {
          // Forming a document object of the sourceMetadata file
                Document doc = formDocument(metaDataFile);
                Element srcElt = doc.getElementById(sourceName); // get src's root element
                for(int i=0;i<nodeNames.length;i++)
                {
                   NodeList nl1 = srcElt.getElementsByTagName(nodeNames[i]);//get all category
                   elements within the source
                    Node nd1 = nl1.item(0); // Get the first node in the nodeList
                    returnVals[i] = nd1.getFirstChild().getNodeValue().trim();
                }
             }
             catch(Exception e)
             {
                  System.out.println("Exception occured in: getSourceDetails()");
             }
        return(returnVals);
```

```
}



/*
 This function returns the list of nodes used to parse data from a source.
 Usually it represents the topology of the source.
 For example, if a source is parsed starting from the List of States in a
 country page,then
 <statesListPage> is the firstNode in the list of nodes returned followed by
 <citiesListPage>, <hotelsListPage> and <hotelsDetailsPage> nodes
*/

 public NodeList getNodeListForSource(String srcHomeURL, String category)
 {
   NodeList nl2 = null;
   try
   {
    // Forming a document object of the sourceMetadata file
     Document doc = formDocument(metaDataFile);

    // Getting the Xpath expressions for a given source and given category
     Element srcElt = doc.getElementById(srcHomeURL); // get src's root element
     NodeList nl1 = srcElt.getElementsByTagName(category);
        //get all category elements within the source
     Node nd1 = nl1.item(0); // Get the first node in the nodeList

      if(nd1.hasChildNodes() == true)
        {
         Node nd3 = nd1.getFirstChild();
         while(nd3.getNodeName() != "XPathExpressions")
          {
                    nd3 = nd3.getNextSibling();
          }

         if(nd3.getNodeName() == "XPathExpressions")
          {
                    boolean starter = true;
                              if(nd3.hasChildNodes() == true)
                    {
                              nl2 = nd3.getChildNodes();
                    }
          }
        }
   }
   catch(Exception e)
   {
         System.out.println("Exception occured in: getNodeListForSource()");
   }
   return(nl2);
 }
```

```
/* This function returns the attributes of a given node within the specified category.
   The result is a NamedNodeMap containg the list of attributes.
   For example, we can get the list of attributes for the hotel category node for a
   given source
 */

 public NamedNodeMap getCategoryAttributes(String srcHomeURL, String category)
 {
        NamedNodeMap nnoMap1 = null;
        try
        {
         // Forming a document object of the sourceMetadata file
        Document doc = formDocument(metaDataFile);

        // Getting the Xpath expressions for a given source and given category
        Element srcElt = doc.getElementById(srcHomeURL); // get src's root element
        NodeList nl1 = srcElt.getElementsByTagName(category);
        //get all category elements within the source
        Node nd1 = nl1.item(0); // Get the first node in the nodeList
     nnoMap1 = nd1.getAttributes(); // find the number of attributes in the firstNode
          // This nnoMap1 variable will be used to form the exact InitialFullURL for a
            // specific source
    }
    catch(Exception e)
    {
        System.out.println("Exception occured in: getCategoryAttributes()");
    }
     return(nnoMap1);
 }

 /*This function updates the value of FreqOfVisits node for a given source*/

 public  void updateFreqOfVisitsNodeValue(String sourceName)
 {
   try
   {
     // Forming a document object of the sourceMetadata file
     Document doc = formDocument(metaDataFile);
     Element srcElt = doc.getElementById(sourceName); // get src's root element
     NodeList nl1 = srcElt.getElementsByTagName("freqOfVisits");//get specific element
     Node nd1 = nl1.item(0); // Get the first node in the nodeList
     nd1.getFirstChild().setNodeValue("00000000T004500");
     //update changes
     updateFileUsingTransformer(doc);
   }
   catch(Exception e)
   {
        System.out.println("Exception occured in: updateFreqOfVisitsNodeValue()");
   }
 }
```

```
/*
  This function updates the node value of a specific node with the given value
  This function was meant to update lastModified and lastVisited nodes of a
  given source
 */

public void updateTimeNodeValue(String sourceName,String nodeName,String nodeValue)
{
  try
  {
    // Forming a document object of the sourceMetadata file
    Document doc = formDocument(metaDataFile);
    Element srcElt = doc.getElementById(sourceName); // get src's root element

    NodeList nl1 = srcElt.getElementsByTagName(nodeName);//get specific element
    Node nd1 = nl1.item(0); // Get the first node in the nodeList
       System.out.println("Before node value is updated :" + nodeName + ":" +
       nd1.getFirstChild().getNodeValue());
    nd1.getFirstChild().setNodeValue(nodeValue);
       System.out.println("After node value is updated :" + nodeName + ":" +
       nd1.getFirstChild().getNodeValue());

    //update changes
    updateFileUsingTransformer(doc);
  }
  catch(Exception e)
  {
       System.out.println("Exception occured in: updateTimeNodeValue()");
  }
}

/*
 This function is used to write the givee Document object into a given file.
 It was meant to update the changes made to the Document object
 (by updateTimeNodeValue() and updateFreqOfVisitsNodeValue()) by writing it
 back to the SourceMetaDataFile.
 */

public void updateFileUsingTransformer(Document doc)
{
  try{
   DOMSource source = new DOMSource(doc);
   StreamResult result = new StreamResult(new FileOutputStream(metaDataFile));
   TransformerFactory transFactory = TransformerFactory.newInstance();
   Transformer transformer = transFactory.newTransformer();
   transformer.transform(source, result);
   appendMissedDTDPart();
   }
   catch(Exception e)
   {
       System.out.println("Exception occured in updateFileUsingTransformer() of
       MetaDataFileInterface.");
   }
}
```

```
        /*
         This function appends the InternalDTD part of a file that is omitted while
         writing the Document object 'doc' to the file using updateFileUsingTrasformer().
         */
        public void appendMissedDTDPart()
        {
           try{
           //String  metaDataFile = "DataSource_Metadata_Modified.xml";
            FileInputStream  fistream = null;
            FileOutputStream fostream = null;
            BufferedReader br = null;
            PrintStream p = null;

            fistream = new FileInputStream(metaDataFile);
            fostream = new FileOutputStream("temp.xml");
            br = new BufferedReader(new InputStreamReader(fistream));
            p = new PrintStream(fostream);

            while(true) {
               String temp = br.readLine();
               if(temp!=null)  {
                 if(temp.toLowerCase().indexOf("<?xml version=\"1.0\" encoding=")!= -1) {
                 String replacingString = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
                             "<!DOCTYPE sourceDescriptions[<!ATTLIST source id ID #REQUIRED>]>";
                  temp = replacingString;
                 }
                 p.println("" + temp);
                }
               else
                 break;
             }
        br.close();
        p.close();

        fistream = new FileInputStream("temp.xml");
        fostream = new FileOutputStream(metaDataFile);
        br = new BufferedReader(new InputStreamReader(fistream));
        p = new PrintStream(fostream);

        while(true)
        {
          String temp = br.readLine();
          if(temp!=null) {
                  p.println("" + temp);
             }
           else
             break;
        }
        br.close();
        p.close();
       }
    catch(Exception e) {
        System.out.println("Exception occured in : appendMissedDTDPart() of MetaDataFileInterface");
         }
      }
```

```
    /* Function to form an DOCUMENT object given an XML source file */
    public Document formDocument(String fileName) throws Exception
    {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document document = builder.parse(fileName);
            return(document);
    }
}
```

# Appendix H

## Java code for CurrentTime

```
package CMS;
import java.util.*;

public class CurrentTime
{
public static String getCurrentDate_Time()
        {
           Calendar cal = Calendar.getInstance(TimeZone.getDefault());

           String DATE_FORMAT = "yyyy-MM-dd HH:mm:ss";
           java.text.SimpleDateFormat sdf =
                new java.text.SimpleDateFormat(DATE_FORMAT);
           sdf.setTimeZone(TimeZone.getDefault());
           return("" + sdf.format(cal.getTime()));
        }
}
```

# VITA

Karthik Chinnayan Muthukumaraswamy

Candidate for the Degree of

Master of Science

**Thesis:** SUPPLYING DATA FOR AN RDF BASED CONTENT MANAGEMENT SYSTEM

**Major Field:** Computer Science

**Biographical:**

**Personal Data:** Born in Coimbatore, Tamil Nadu, India, January 19, 1981, the son of Mr. C. Muthukumaraswamy and Mrs. M. Krishnaveni.

**Education:** Obtained Senior High School Diploma from S.B.O.A, Coimbatore, India, in May 1998; completed Bachelor of Engineering in Computer Science from Bharathiar University, Coimbatore, India, May 2002; fulfilled requirements for the Master of Science Degree at Oklahoma State University in December, 2005.

**Experience:** August 2003 – April 2005. Graduate Assistant and Technical support person, Career Services, Oklahoma State University, Stillwater, OK, USA.

# ABSTRACT

Name: Karthik Chinnayan Muthukumaraswamy Date of degree: December 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of study: Supplying data for an RDF content management system

Pages of Study: 89 Candidate for the Degree of Master of Science

Major Field: Computer Science

Abstract:

Personalized Internet Services have become an important topic of research and study. The concept of personalization is possible only with a system that can integrate and deliver data that are present in large number of repositories. An 'e-Travel system' being developed by the 'Agent Lab' group aims to provide personalized services to its customers in the travel domain. The main objective of this thesis was to propose an architecture for the content management system (CMS), which is an integral part of the e-Travel system. CMS's responsibilities include gathering and integrating data from varied sources (Web pages, Relational Databases, RDF dumps). In this attempt, many existing architectures for CMS were studied. Most of these architectures were based on either lazy (on demand) or eager (data warehousing) approach for data integration. Studies and requirements proved that an ideal architecture will be the one in which some data is pre fetched, processed, integrated (using a global ontology) and stored in the database, while other data, when not present in the database, is fetched and processed upon user queries. As a result, an architecture that uses Resource Description Framework (RDF) for representing the travel domain was developed. This architecture was implemented using Jena – a Java framework for building Semantic Web applications. The functionality of this architecture was studied with the help of 'Verified Content Providers' – sources of data that are considered reliable and whose structures are known beforehand. 'Time Adaptive' capability built within the architecture determined how often these sources are visited in search of updated content.

Dr. MARCIN PAPRZYCKI

Thesis Advisor: ----------