USING SOFTWARE AGENTS TO INDEX DATA FOR AN E-TRAVEL SYSTEM

By

ANDY NAULI

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

2000

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
August, 2003

USING SOFTWARE AGENTS TO INDEX

DATA FOR AN E-TRAVEL SYSTEM

Thesis Approved:

Thesis Adviser

Dean of the Graduate College

Acknowledgments

I would like to express my sincere appreciation to Dr. Marcin Paprzycki for his continuous support and guidance throughout the completion of this thesis. His cheerful guidance and positive attitude have helped me overcome several obstacles during the course of my research.

Special thanks are also given to all the team members for this project: Austin Gilbert, Minor Gordon, Steve Williams, and Jimmy Wright. You guys are the greatest persons I have ever work with. I am looking forward to work with you guys in the future project.

I would like to thank Dr. Ajith Abraham and Dr. Martin Crossland for serving on my thesis committee and for their invaluable input.

And the last but not least, for the most part, I would like to thank my father and my mother for their continuous support (financially and spiritually). Without your support, any of this will not be possible. Thank you for the love, encouragement and belief to keep me going.

Table of Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

List of Listings

Chapter 1

Introduction

There were about 2.1 billion web pages on the Internet in July 2000 with predicted growth rate of 7.3 million web pages per day [39]. From these numbers we can simply conclude that there are enormous amount of information available on the Internet. There are advantages and disadvantages associated with such rapid growth of information. One advantage is that "almost" all types of information can be found on the Internet. These types include travel related information such as hotel, museum, flight, cruise, restaurant, amusement park, and etc. The disadvantage resembles the classic problem of finding a needle in the haystack. We know for sure that the information is out there somewhere but we have no clue as where it might reside. To solve this problem, we rely mostly on search engines to lead us to the location of the information. But with the current amount of information available on the Internet, search results returned by search engines will likely to include more unrelated hits that need further refinement by the user (manually filter through a bunch of search results). A simple search using the keyword *hotel* on Google search engine returns about 82,500,000 hits sorted by their rank (Google uses PageRank ranking system to order search results, see [42] for details on PageRank ranking algorithm).

Although ranking systems can provide help in locating the most related hits, they

do not always work. Without being able to specify the context of the search, ranking systems are not able to pinpoint the specific domain of the search (thus reducing the overall success of the algorithms to provide most related hits). This problem also inherent in the way search engines display the search results. Users are always presented with the search results organized in a non-intuitive order (no categorization or catalog). Some users find it easier to browse through the catalogs (e.g. Yahoo! Catalog or Google Directory) but they have to realize that the catalogs are not meant to be complete. Considering the way catalogs are created (manually added by human), their updates are usually slower compared with those of search engines.

In terms of coverage (number of collected web pages), the largest is provided by Google, with total collection of 3,083,324,652 web pages (April 2003). The problem of how accessible the web is imminent (see [31] for more information). This problem has resulted in the development of focused web crawlers or topical web crawlers [7, 43, 36]. Topical web crawlers scour web pages based on the specific context thus they are able to locate web pages that have not been discovered by search engines (e.g. Google). Topical web crawlers also designed to reduce the number of unrelated hits (e.g. search results returned by search engines) by filtering out the results that do not belong to the specified context of the search. The advantage of topical web crawler to search engines is the ability to specify domain of the keyword used during the search. Coupled with this knowledge (context of the search), ranking algorithms used in topical web crawlers are able to bring more related hits to the user (with embedded categorization or catalog).

In this thesis, an agent-based system for indexing and searching travel related

information from the Internet will be developed. It is also designed to support the Internet travel support system described in [20, 26, 2, 46] which requires constant feed of travel related information into the system in order to serve the needs of its users effectively. The functionality of this agent-based indexing system will be divided into two major parts: the search agent and the indexing agent. (the word *agent* refers to intelligent software agents as described in [19, 25]). The former is the web crawler designed to locate travel related information from the Internet. The latter is responsible for creating and storing an index of the information returned by search agents (please refer to Chapter 2 for the details of storage subsystem and the reason behind indexing). Both search agent and indexing agent will be developed according to intelligent software agent framework [11]. Search agent falls into the category of information gathering agents. Software agent is the chosen platform for developing entities that gather information from the Internet because their characteristics match those of the Internet. Software agent had been proven to be effective in handling diverse and dynamic nature of the Internet [23, 24]. It is not the purpose of this thesis to describe complete history in the development of information gathering agents but examples of intelligent information gathering agents can be found in [38, 27, 28, 10, 8]. There are some advantages in designing indexing agent as an agent as well. These advantages are derived from basic property of software agents [23, 24] e.g. cloning and migration. For example, indexing agent can clone itself to perform load-balancing.

1.1 History and Background

Current Internet travel support systems (e.g. Travelocity, Expedia, TravelZoo, etc.) support limited travel choices. They are only able to provide travelers with information about flights, hotels, cars, cruises, and vacation packages (as well as reservation support). While such systems have been able to identify most traveler's needs, they do not support other types of useful travel information (e.g. restaurant, movie theaters, national parks, historical sites and etc.) as well as content personalization. On the other hand, travel related information is readily available from the Internet. It is desirable to use this information in order to provide more travel choices and information to user.

This limitation can be overcome using a system that can effectively gather and store travel related information from the Internet as well as presenting them to user. One way to design such system is using software agent. There have been numerous attempts at applying agent paradigm to Internet travel support system [48, 40]. Such attempts are either failed or abandoned in their early stages of development [2].

The initial design of the e-travel system discussed in this thesis can be found in [2, 20, 26, 46]. To avoid the same pitfall faced by other Internet travel support systems, the e-travel system is devised from its business perspective [2]. The original design had gone through some minor changes but the concept of dividing the system into two major subsystems (content management and content delivery) remains the same. Content management subsystem is responsible for organizing incoming and outgoing content from and to the system. On the other hand, content delivery subsystem

handles delivery of information to the user. Please refer to [2] for the complete overview of the system. In this thesis, the concern is not on the overall e-travel system but on the content management aspect of it. The aim of this section is to provide the history and background behind the development of e-travel system.

1.2 Requirements for E-Travel System

Apart from the content management (developed in this thesis and also in [53]), the e-travel system [2, 20, 26, 44] requires several supporting subsystems to be able to perform effectively. Here are those requirements:

1. Content Delivery Subsystem

   This subsystem handles personalized delivery of content to the users as well as some other maintenance tasks. This subsystem is also mostly agent based. There are several agents involved such as personal agent, query agent, acquisition agent, data validation agent and deconfliction agent, and others. The complete descriptions on these agents can be found on [2].

2. Storage

   There are many choices on the implementation of storage subsystem. It must be chosen carefully so that the storage implementation does not limit the search options and result displays that this system offers [51]. Current Internet travel services have based their underlying storage system on traditional database which limits their search capability. On the contrary, the e-travel system [2, 46, 44] was designed with indexing in mind. Based on this idea, XML registry

is chosen as the underlying storage subsystem. The implementation of XML registry for Internet travel support system has been reported on [53].

3. GIS

   GIS subsystem is used to transform location information into latitude and longitude pair (reverse geo-coding). The latitude and longitude information will come in handy when performing spatial query (e.g. finding a hotel within the radius of 10 miles).

   There are two approaches to GIS subsystem. First, the GIS subsystem is implemented in house, in other words, it is part of the whole system. Second, GIS subsystem is provided by third party (external entity). In this thesis, we will be relying on external GIS subsystem (see page 35). During the development stage, such approach should be sufficient.

4. Expert System

   The expert system subsystem is used to mine user profiles and user behavior database to provide personalization. The expert system subsystem is comprised of several expert subsystems (e.g. travel expert, advertising expert, and etc.) and most of them are in the form of agents. The development of expert system for e-travel has been reported on [45].

5. User Interface

   User interface is the place for user to interact with the system. It comes in different format (e.g. HTML, WML, etc.) that is customized to any Internet

connected devices. Internet connected devices are ranged from full-blown computer system to small wristwatch with wireless connection. User interface must be designed for each of these devices but each user interface must provide the same functionality. The user interfaces are designed using using XUL and XUP [37] developed by Mozilla.

As previously mentioned in the preceding sections, this thesis deals with gathering and indexing travel related information from the Internet. Thus it only touches the content management part of the system. There are groups working on other parts of the system, and their development will be reported in subsequent papers.

1.3 Organization

This thesis is organized as follows: Chapter 1 is an introduction. Chapter 2 describes the system for indexing travel data. This chapter will establish the technical overview and thus provide better understanding of subsequent chapters. Chapter 3 introduces the indexing agent. It also explains the method of inserting an item into XML registry and/or performing additional lookup to the GIS subsystem in order to resolve latitude and longitude information. Chapter 4 deals with search agents. Search agents fall into the category of topical web crawler or focused web crawler [7, 43, 36]. In this chapter, several information retrieval techniques are also discussed because these techniques are required for efficient processing of document/information retrieved from the Internet. Chapter 5 summarizes the study and give direction for future works.

Chapter 2

Proposed Architecture

There are several key components required to index data for the e-travel system. Most of these components are depicted in Figure 2.1. Search agents gather travel re-

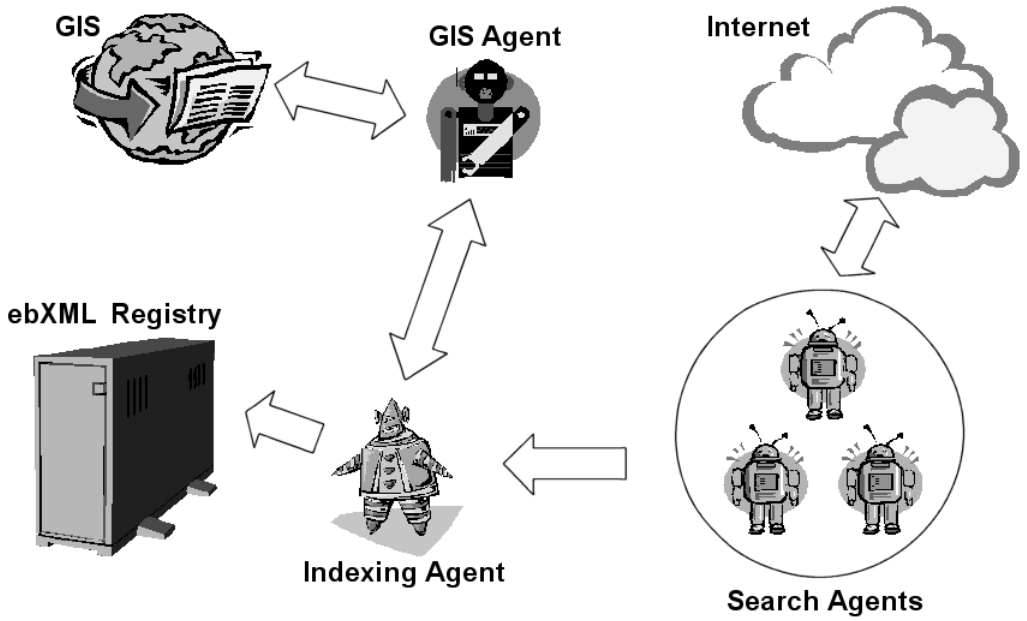

Figure 2.1: Proposed architecture for indexing travel data from the Internet.

lated data from the Internet and send this information to the indexing agent. Indexing agents on the other hand index this information into ebXML registry. The indexing agent and search agent will be exclusively discussed in Chapter 3 and Chapter 4 respectively. First, lets consider the approach to information or content gathering from the Internet. There are two approaches to information gathering as described

in [46, 53]:

1. Aggregation: Retrieving all information that the system will possibly need before hand and organizing it in a predefined (by humans) format within databases for later access.

2. Selection: Maintaining a general idea as to what content is available on the Internet, indexing it, and retrieving it only as becomes necessary to satisfy user's query.

Both approaches have their own advantages and disadvantages (summarized in Table 2.1). Most online travel providers (e.g. Travelocity, Expedia, Hotwire, and etc.) follow the first approach because it is simple and they have enough resources (processor and storage) in terms of utilization. The second approach is followed mostly by search engines. They gather only a limited amount of data (e.g. page title, page headers, or few selected texts) in order to support the search function.

The e-travel system fully embraces the indexing approach (the second approach) by keeping a well-organized and highly cross-referenced index of Internet-based content as described in [1]. This approach allows us to effectively deal with data available from multiple sources across the Internet in such a way that pertinent information may be efficiently and accurately selected and delivered to consumers [53].

2.1 Why Indexing?

To deliver the most relevant and most accurate travel choices to consumer [20, 26, 45], the structure of the information within the system must be carefully considered. In

designing the e-travel system, the advantages of accurate indexing and the avoidance of cache coherency issues outweighed the disadvantages of remotely stored content [46]. The idea is to focus on the classification of the content instead of the content itself. The index must be succinct enough so that it allows robust categorization and organization, yet verbose enough to satisfy all of the requirements of both content management and content delivery subsystems. This way we can make sure that limitations of current search engines (no categorization or catalog) do not get carried over into the e-travel system (Chapter 1 describes this problems). Also noted in [51], the search options and result displays of tourism information system have been limited by the database principles. With indexing, we can avoid this limitation and provides richer set of search options and result displays.

Method	Advantages	Disadvantages
1. Aggregation	Immediate availability of local content. Require no pre-processing. Implementation is simple.	Cache coherency problem. Amount of data generated is large. Resource intensive (processor time and bandwidth).
2. Selection	Amount of data generated is small. Resource effective (processor time and bandwidth). No cache coherency problem (fresh content).	Require pre-processing. Implementation is arduous. Availability cannot be guaranteed

Table 2.1: Two possible methods to content management.

2.2 The Index Token

Index token is the attempt to develop a *classification system* of the world of travel content [2]. The *site* is used as a basis for this classification scheme. A site is the real-world logical division upon which travel resources are modeled [53]. Each site contains three basic quantities: *provider*, *type*, and *location*. These three components and additional `<?notes?>` component together form an index token [46]:

`(<provider>,<type>,<location>,<?notes?>)`

This token is the bridge between logical site classification and implementation-level storage. The `<?notes?>` component is a special component that does not represent a site. It is used to provide administrative functions (e.g to mark the token as incomplete, see sub section 2.2.4). More on administrative functions in Chapter 4. The next three sub sections describes the details of these components. These three components together uniquely identify each travel resource.

2.2.1 The Travel `provider` Component

This component is represented in the form of a Uniform Resource Locator (URI) e.g. `http://www.eskimojoes.com` or `ftp://ftp.okstate.edu`. The URI describes the access method for the travel resource (e.g. http, https, ftp, etc.) and the location (e.g. www.eskimojoes.com, ftp.okstate.edu, etc.) where the travel resource resides. This component is used to identify where the original travel information resides. This component plays an important role for content acquisition subsystem (where the actual information is delivered to the user).

2.2.2 The Travel `type` Component

This component describes the position of a travel resource in the taxonomic hierarchy of all resources (e.g. Accommodations → Hotels → Chains). The taxonomic hierarchy is roughly modeled from Yahoo! Category of travel resources. The system will utilize this information to filter out information that is not pertinent to users needs. This taxonomy is not intended to be complete (as more are added later when deemed to be required).

2.2.3 The Travel `location` Component

The taxonomy in this component is modeled on the ISO-3166 standard which defines the continent, country, state or province, and city; and latitude and longitude for exact locations and proximity searches. Location information is very important for differentiating between different sites. It must be hierarchical so that organizational relationships between sites at different locations on different levels can be surmised [53].

2.2.4 Token Completeness

There is a possibility that search agents will return an incomplete tokens (more on this issues in Chapter 4). This issues is possible for instance because majority of travel providers do not support geospatial information (e.g. latitude and longitude information). To identify incomplete tokens, they are marked inside the `<?notes?>` component. These incomplete tokens will be processed later either by indexing agent (resolve the latitude and latitude information by communicating with GIS agent)

or by completion, validation, deconfliction (CVD) agents. CVD agents traverse the registry and process the incomplete/unverified tokens [46]. Token completion and deconfliction is another research topic and it progress will be described in future paper.

2.3 The ebXML Registry

The ebXML registry establishes the classification and catalog for storing the index of travel information. The base classification scheme (root) is called Agentlab. Agentlab has three child nodes and they match the components of index token (depicted in Figure 2.2 on page 17):

- Provider: How to access the content.
- Type: Classification of content type.
- Location: Classification of content location.

The bulk of the information in each of these nodes comes directly from the corresponding element of the index tuple. The external links field of provider node is populated using the URI information from the provider component of index token (see Figure 2.2 → Agentlab Provider Node). The type node is displayed in Figure 2.2 → Agentlab Type Node. The child nodes of the type node form the hierarchy of travel resources (currently based on Yahoo! Category). The location node in Figure 2.2 → Agentlab Location Node display location information according to the hierarchy. An interesting aspect of the construction of the location node is the use of the registry

slot object to contain the latitude/longitude of the site [53]. In the future, this slot can be used to store OpenGIS object for the site.

The ebXML registry supports three different access methods to perform update/-query to the registry. These methods are described below:

- Using the Registry Browser

  The ebXML registry repository provide the Graphical User Interface (GUI) to interact with the registry. This GUI is called registry browser. Registry browser supports all access types (e.g. update, query, authentication, etc.) supported by the registry repository. Registry browser shields user from the complexity of methods call in JAXR (see next access method) by using the more user friendly Swing GUI. This method is not always the desirable because it does not provide automation (e.g. automatic updates by indexing agent → see Chapter 3). Figure 2.3 on page 18 shows the main window of registry browser.

- Using Java API for XML Registry (JAXR)

  Java API for XML registry (JAXR) [16] is part of the Java Web Services Developer Pack [41] from Sun Microsystem, Inc. JAXR provide uniform sets of Application Programming Interface (API) to access many XML registries (e.g. UDDI and ebXML). There are two different category profiles of JAXR: category level 0 and category level 1. Category level 0 supports only UDDI registry. On the other hand, category level 1 is provided by ebXML and support both UDDI and ebXML registry. Indexing agent will utilize JAXR to access the registry, please see Chapter 3 for detail implementation and [16, 41] for references.

- Using SOAPSender

  Simple Object Access Protocol (SOAP) is the only access protocol supported by the ebXML registry. SOAP uses HTTP as the transport protocol thus it is not blocked by most firewall. SOAPSender provides the capability to send SOAP messages to ebXML registry. In fact, JAXR convert each method that accesses the registry into SOAP messages and send these messages using SOAPSender. Below is the basic format of SOAP message to access the registry:

```
<SubmitObjectRequest(parameters)>
  <LeafRegistryObjectList>
    <ClassificationScheme id="<uuid>"(other parameters)>
  </LeafRegistryObjectList>
</SubmitObjectRequest>
```

For more information on ebXML registry as well as access methods please refer to [53, 50]. As we can see in Chapter 3, the advantage of using JAXR is that we can avoid the need to create SOAP based messages (in XML format) in order to access the registry. JAXR provides set of methods to access the registry and automatically convert each of these method calls into the corresponding SOAP messages.

2.4 The GIS Subsystem and GIS Agent

The GIS plays an important role in the e-travel system. Without GIS, the index token will never be completed (because the location information needs to be converted to latitude and longitude information). During the development phase of the e-travel system, the GIS subsystem is provided by third party. There are some advantages and disadvantages following this approach. Their summary are described

in Table 2.2. Nonetheless, during the development phase of the e-travel system, all

GIS Approach	Advantages	Disadvantages
1. Internal	Availability can be guaranteed (99.99% uptime) Consistent format for GIS resource	Need resource to perform initial setup as well as maintenance Extra bandwidth for resolving GIS query
2. External	No extra resources required for setup and maintenance No extra bandwidth require when resolving GIS query	Inconsistent GIS format Availability cannot be 100% guarantee

Table 2.2: Internal vs. External GIS Subsystems

the disadvantages of relying on the third party GIS subsystem are considered to be insignificant.

GIS agent perform the communication with GIS subsystem in order to resolve the latitude and longitude information. GIS agent can be tailored to fit the requirement (e.g. query format) of the GIS Subsystem. If the current GIS subsystem is down, another GIS agent can be launch or created that match the requirement of the new GIS subsystem. This way, indexing agent does not need to adjust its method of resolving GIS query. Indexing agent will consistently utilize the Agent Communication Language (ACL) to communicate with GIS agent (more on this on Chapter 3).

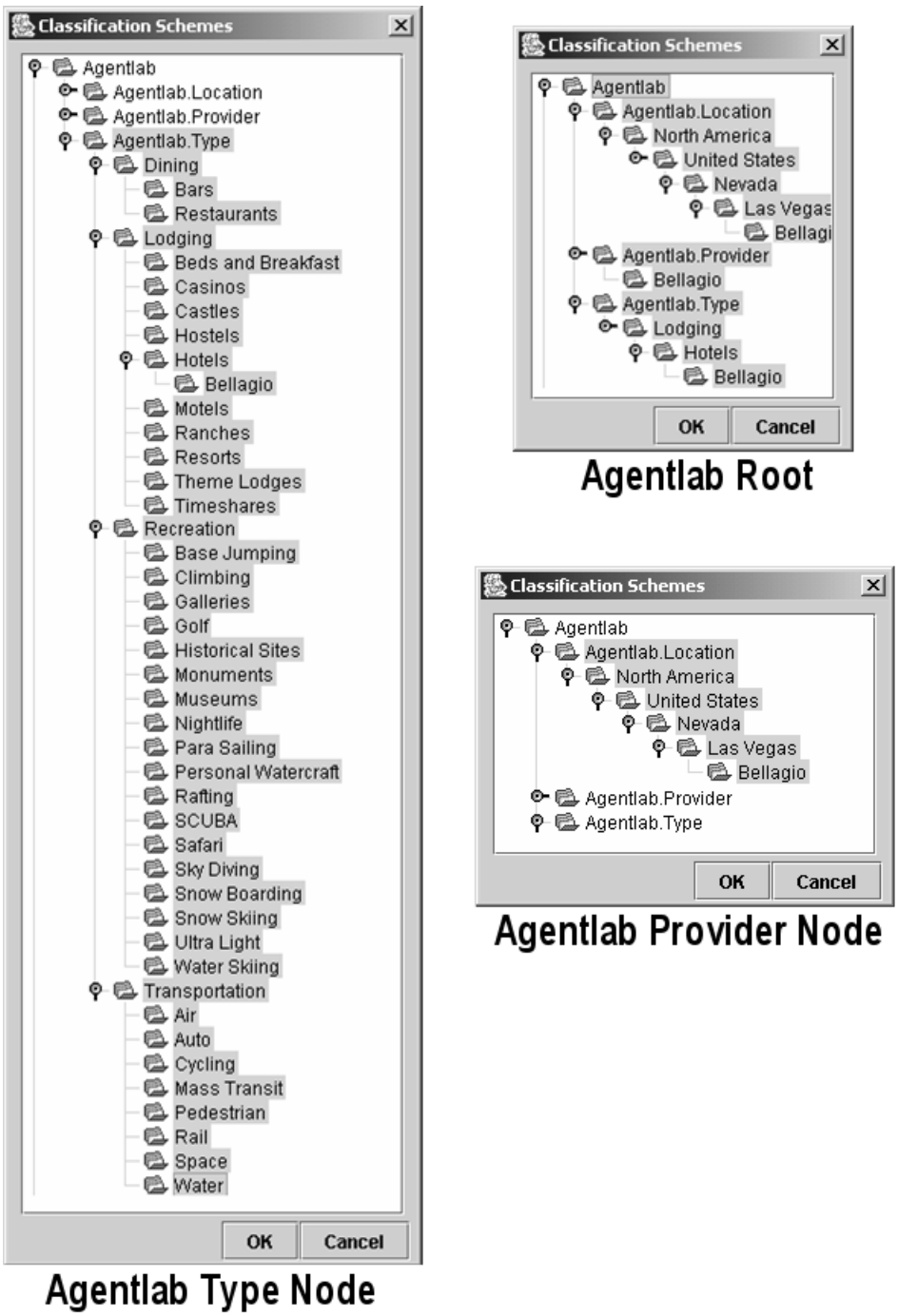

Figure 2.2: Agentlab Registry Classification

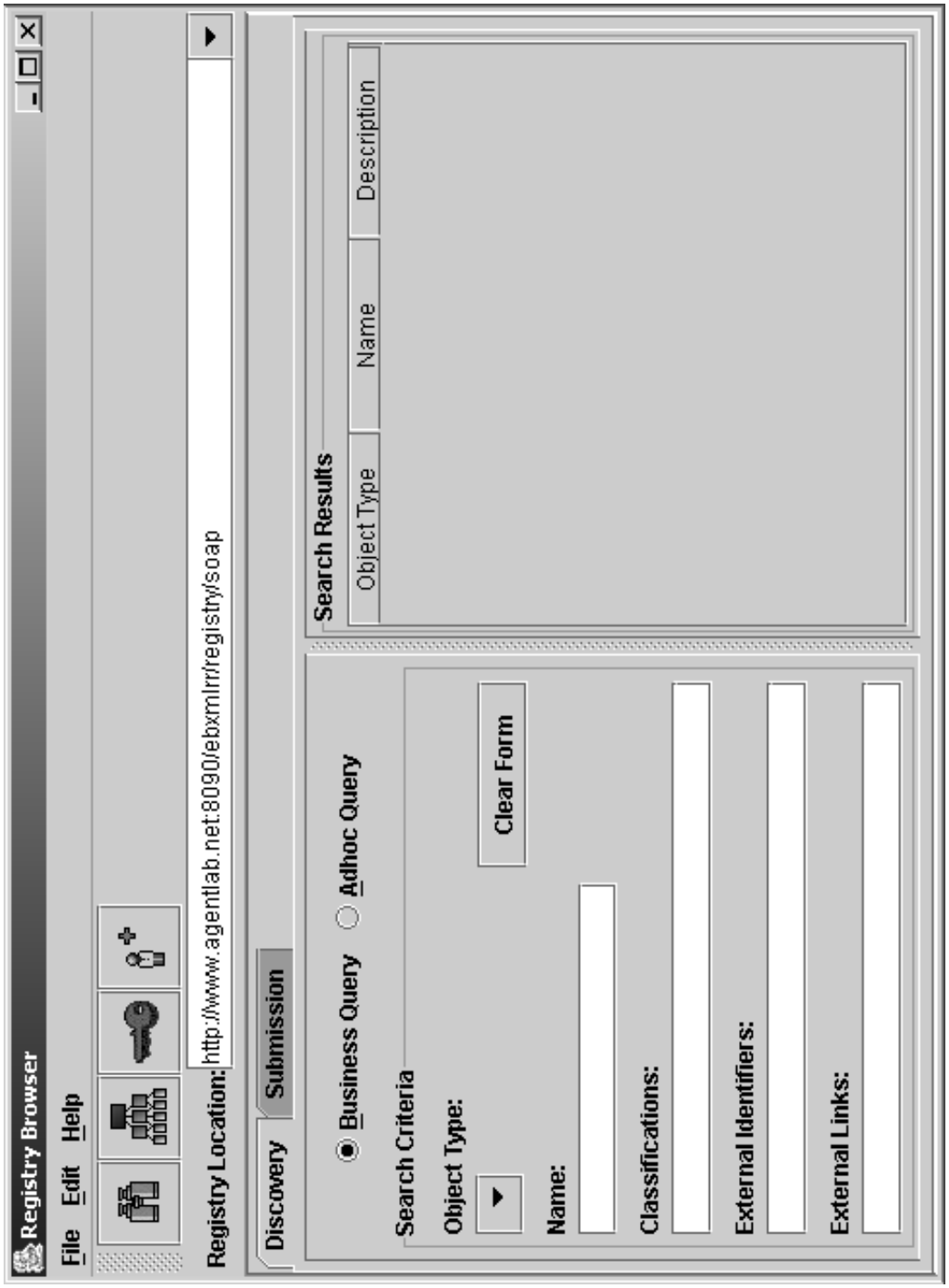

Figure 2.3: The Registry Browser

Chapter 3

Indexing Agent

Indexing agent is a software agent based on Java Agent Development (JADE) framework [13]. JADE provides set of tools and standards (according to FIPA [14]) to make the implementation of multi-agent system easier. Indexing agent also utilizes the JAXR [16] APIs to access the ebXML registry repository. As mentioned in the previous chapters, the indexing agent is responsible for:

- Inserting the index token (see section 2.2) into the ebXML registry repository
- Performing additional GIS lookup to convert location information into latitude and longitude pair.

In this chapter, these two responsibilities will be discussed in detail. The next section will introduce the concept of agent oriented programming with JADE.

3.1 Agent Oriented Programming with JADE

As suggested by the JADE Programmer's Manual [13], in order to understand JADE the reader must be familiar with FIPA [14] standards, such as *Agent Management* (FIPA no. 23), *Agent Communication Language*, and *ACL Message Structure* (FIPA no. 61). These standards are important to establish the concept of software agent architecture as well as communication layout. JADE comes with several main pack-

ages:

- `jade.core`:

  This is JADE core package. It includes `Agent` class and `jade.core.behaviours` sub-package. Any agent created within JADE must be subclass of this `Agent` class. An agent can have many behaviors and these behaviors must be subclass of the `jade.core.behaviours` sub-package.

- `jade.lang.acl`:

  This package provides the Agent Communication Language (ACL) standard according to the FIPA [14] specifications.

- `jade.content`:

  This package support user-defined ontologies and content language. Any expressions in the ACL is represented by content language (e.g SL Codec) and encoded in proper format.

- `jade.content`:

  This package support user-defined ontologies and content language. Any expressions in the ACL is represented by content language (e.g SL Codec) and encoded in proper format. This package is designed to avoid the need to perform conversion between different content expression and semantic checks. Please see Application Defined Content Languages and Ontologies at [13].

- `jade.domain`:

This package represents agent management entities such as Agent Management Service (AMS) and Directory Facilitator (DF). Agent management entities is required for any software agent platform that is compliant with FIPA [14] standards.

- `jade.gui`:

  This package provides set of GUI tools to display and edit many agent related attributes (e.g. identifiers, description, ACL Message, etc.).

- `jade.mtp`:

  This package represents the transport protocol. Transport protocol defines the protocol that carry the communication among agents. By default, JADE uses Internet Inter-ORB Protocol (IIOP) for communication. There several add-in that can be utilized to add more protocol into JADE (e.g. HTTP).

- `jade.proto`:

  This package defines standard interaction protocols according to FIPA [14] standards (e.g. fipa-request, fipa-query, fipa-contract-net, fipa-subscribe, etc.) as well as custom defined protocols.

- `jade.wrapper`:

  This package provides wrapper e.g. to use JADE as a library, such as launching JADE agents and agent containers from external Java applications.

Besides providing several core packages to the user/programmer, JADE also provides several tools (mostly in GUI) to simplify administration and application development.

These tools are contained in `jade.tools` package, and they are as follows:

- Remote Management Agent (RMA):

  Provides GUI for platform management and control. RMA also able to start other JADE tools.

- Dummy Agent:

  Dummy Agent is JADE GUI agent for debugging and monitoring. It is able to send ACL messages to other agents. Dummy Agent also displays received and sent messages (include the timestamps information).

- Sniffer:

  Sniffer intercepts ACL messages that travel within the system and displays them graphically using UML sequence diagram. Sniffer is also enhanced by Introspector Agent to inspect individual agent.

- Introspector Agent

  This agent monitors the lifecycle of an agent and its exchanged ACL messages.

- Socket Proxy Agent

  This agent acts as bidirectional gateways between JADE platform and ordinary TCP/IP connection. It is useful to handle network firewalls or to provide platform interactions with Java applets within a web browser.

- DF GUI

The DF GUI is graphical user interface for Directory Facilitator (DF) of JADE. It allows simple and intuitive way to control the DF e.g. federate DF with other DF.

3.2 Yellow Page Service

JADE provides a yellow page service called Directory Faciliator (DF). With DF, each agent can advertise their service and let other agents discover the services they provide. For example, indexing agent will advertise its service (it provides indexing service) and search agent can look up the DF for any agents that provides indexing service (so it can send the index token to that agent). This way, we can avoid hard coding agent's name, and additional indexing agents can be launch (to perform load-balancing). Search agents can simply find the indexing agents by querying the DF.

3.3 Access to ebXML Registry Repository Using JAXR

Indexing agent will utilize the Java API for XML registry [16] to interface with the registry. The JAXR provides the uniform APIs to connect to many XML registry (e.g. UDDI or ebXML). In order to access the registry, the first step is to create the `Connection` to the JAXR provider. The code snippet below establishes the connection to the registry (Listing 3.1):

```
...
import javax.xml.registry.*;
...
//create an instance of connection factory class
ConnectionFactory connFact = ConnectionFactory.newInstance();
...
//set the connection properties
Properties props = new Properties();
props.setProperty("javax.xml.registry.queryManagerURL",
  "http://xamot:8090/ebxmlrr/registry/soap/");
```

```
props.setProperty("javax.xml.registry.lifeCycleManagerURL",
  "http://xamot:8090/ebxmlrr/registry/soap/");
//set the property for connection and establish the connection
...
connFactory.setProperties(props);
Connection connection = connFactory.createConnection();
...
```

Listing 3.1: Establishing Connection to Registry

The Java `Property` class is used to store all the connection properties. For this purpose, we have set two properties for the connection: `queryManagerURL` and `lifeCycleManagerURL`. The former specifies the URL of the query manager service (e.g. to perform query to the registry) of the target registry. The latter specifies the URL of the life cycle manager service (e.g. to perform updates) of the target registry. There are other properties related to the connection and they can be found in JAXR [16] tutorial.

Once the connection to the registry is established, the `RegistryService` object can be obtained from the connection. From the `RegistryService` object, we can obtain the `BusinessQueryManager` and `BusinessLifeCycleManager`. The former provides an interface for querying the registry and the former provides interface for updating the registry. The code snippet below describes that purpose (Listing 3.2):

```
...
//obtain the RegistryService object from the connection
RegistryService rs = connection.getRegistryService();
//obtain the BusinessQueryManager and BusinessLifeCycleManager
//from the RegistryService
BusinessQueryManager bqm = rs.getBusinessQueryManager();
BusinessLifeCycleManager blcm = rs.getBusinessLifeCycleManager();
...
```

Listing 3.2: Obtaining BQM and BLCM from RegistryService

Using the `BusinessQueryManager`, the content of the registry can be queried. The `BusinessQueryManager` provides several methods to query the registry (e.g. `findOrganizations`, `findServices`, or `findConcepts`). These methods return ei-

ther `RegistryObject` or `BulkCollection` object. The example below queries the registry for the classification scheme, and iterates the classification scheme for any of its children concept (Listing 3.3):

```
...
//get an instance of BusinessQueryManager
BusinessQueryManager bqm = rs.getBusinessQueryManager();
...
//find classification scheme based on its name
ClassificationScheme cScheme = bqm.findClassificationSchemeByName(
  null, "Agentlab");
//get all the children nodes of this classification scheme
Collection children = cScheme.getChildrenConcepts();
//get the iterator
Iterator list = children.iterator();
while (list.hasNext()) {
  Concept con = (Concept) list.next();
  //prints the value of this node
  System.out.println(con.getValue());
}
...
```

Listing 3.3: Querying the ebXML Registry using JAXR

On the other hand, the `BusinessLifeCycleManager` provides a set of methods to update the registry (e.g. `saveConcepts`, `saveOrganizations`, `saveServices`). Notice that all these methods use the plural object names (Concepts, Organizations, etc.), it is because these methods accept `Collection` object as their parameter. The example below saves a new `Concept` object as the children of `Agentlab` classification scheme (Listing 3.4):

```
...
//get an instance of BusinessLifeCycleManager
BusinessLifeCycleManager blcm = rs.getBusinessLifeCycleManager();
//create a new concept
Concept newConcept = blcm.createConcept(cScheme, "Test", "Test");
//create an InternationalString object for the description
InternationalString desc = blcm.createInternationalString("Test");
//set the description
newConcept.setDescription(desc);
//new Collection object
Collection save = new ArrayList();
//add the new concept to the Collection
save.add(newConcept);
//save the concept
BulkResponse br = blcm.saveConcepts(save);
...
```

Listing 3.4: Saving Concept into ebXML Registry

Any methods that update the registry always return `BulkResponse` object. The `BulkResponse` object contains either `RegistryObject` objects as the result of the success update or `Exception` objects as the result of failure update.

3.3.1 Authentication Issues

Any updates (insert, delete, or edit) on the ebXML registry require authentication. Before any updates are committed into the registry, they must first be authenticated using user name and password. Authentication is usually assigned to the `Connection` object. In order to avoid having to authenticate everytime the indexing agent tries to save something into the registry, the source code of the ebXML provider will be slightly modified. Normally, if the connection to the ebXML registry is not authenticated, then ebXML provider will display a login dialog for user to type in the alias, keystore password, and private key password (Figure 3.1). We want to avoid having to type this information everytime indexing agent tries to update the ebXML registry. In order to do that, the source code for ebXML client must be modified. The `authenticate()` method in the `ConnectionImpl.java` file in the ebXML source code directory had been modified as follows (Listing 3.5):

Figure 3.1: The Authenticate Dialog Box

```
//The original authenticate method from ebXML has been modified
//No longer needs to display login dialog
SecurityUtil su = SecurityUtil.getInstance();
Set privateCredentials = new HashSet();
//the aliasToX500PrivateCredential() method will retrieve
//keystore password and private key password from
//jaxr-ebxml.properties file
privateCredentials.add(
  su.aliasToX500PrivateCredential("andy.nauli"));
...
```

Listing 3.5: Modifying ebXML Source Code

3.4 Inserting Index Token into the ebXML Registry

JADE agent example and JAXR APIs had been introduced, it is now the time to combine these two together to make "useful" indexing agent. In Chapter 2 Section 2.3, the concept of a site had been introduced. Each site in the ebXML registry is represented by three nodes (`provider`, `location`, and `type`). Based on this principle, for a successful insert, the indexing agent must create three new nodes and insert them at the appropriate places. Also in Chapter 2 Section 2.2, the concept of index token had been introduced. Index token is the bridge between the logical site classification and implementation level storage [53]. Thus indexing agent must convert index token received from the search agent into the nodes of ebXML registry.

3.4.1 Parsing the Content of an ACL Message

Agents communicate using ACL messages (based on FIPA [14] standards). ACL message supports several attributes (e.g. sender, receiver, performative, in-reply-to, conversation-id, and etc.) that can be accessed using `set/get<Attribute>()` methods. The content attribute of the ACL message is where the content of the message is stored. The content attribute support several content languages such as

SL, XML, as well as RDF. JADE also support Java objects as the content of the message as long as the object is `serializable`.

For the purpose of indexing agent, we will utilize Java object (e.g. `Properties`) as the content of the message. `Properties` is a `serializable` object and it can represent the index token easily. The following code shows how to represent the index token as the `Properties` object (Listing 3.6):

```
...
Properties p = new Properties();
//represent provider component
p.setProperty("provider","http://www.mountaindew.com");
//represent type component
p.setProperty("type","Camping");
//represent location component
p.setProperty("location","Stillwater,OK");
...
```

Listing 3.6: Parsing Content of an ACL Message

As you can see, the index token have been represented as the `Properties` object. The location component is not represented in the form of latitude and longitude. There are two reasons for this: (i) most travel sites do not include latitude and longitude information and (ii) with only latitude and longitude information, it will impossible for the indexing agent to decide where the location node will be created (e.g. under what state and city). `Properties` object can be inserted as the content of the message using the method `setContentObject(p)` and retrieved using the method `getContentObject()`. Both of these methods automatically activate the usage of `Base64` encoding.

Search agents will be responsible for constructing the `Properties` object to represent the index token and insert it as the content of the ACL message. Indexing agent will receive this message and extract the `Properties` object from the content

of the message. All the properties in `Properties` object can be listed using the `propertyNames()` method. Of course there are other ways to do this, but this way is deemed to be simple and it does not require any external Java classes. For example, JADE support the XML content languages by downloading an extra add-on from JADE [13] web site. To use the XML content language, you must create a class that represent the object as well as the associated ontologies. This simply leads to the creation of extra classes that must be carried by the agents. On the other hand, the `Properties` object is included in every Java distribution and it does not require any additional class to be defined.

3.4.2 The Naming Mechanism

The naming mechanism refers to the way to choose the content of **value** attribute of the nodes in the ebXML registry repository. There are other attributes (e.g. name, description, external links, slots, etc.) and they are listed in Figure 3.2. Because each

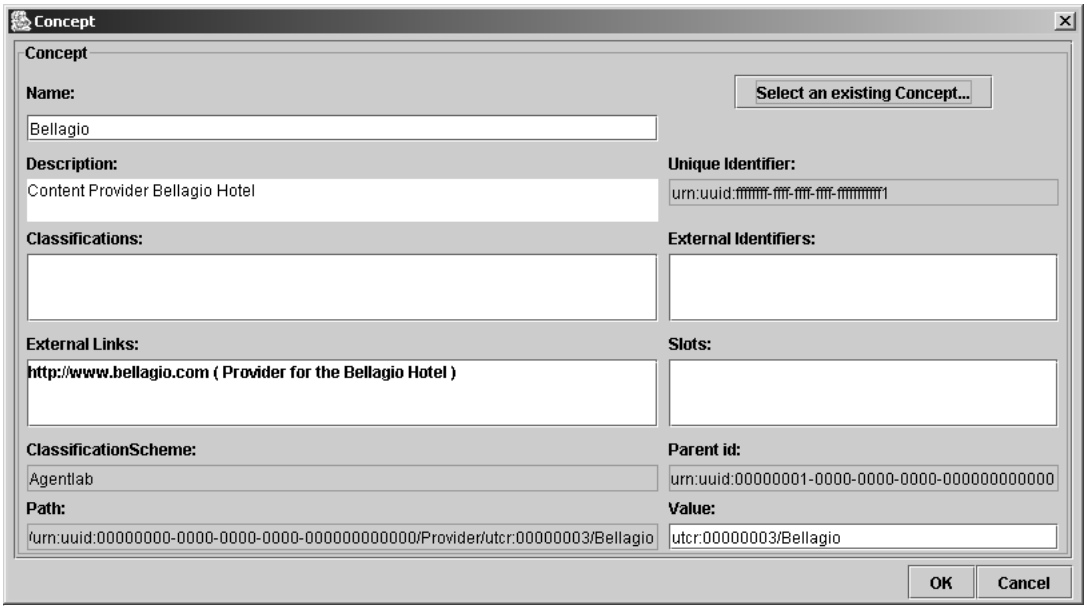

Figure 3.2: The Attributes for ebXML Nodes

site is represented by three nodes, there must be a way to linked them together. The **value** attributes is used to link these three nodes together to represent a site. The **value** attribute also plays an important role when the ebXML is queried for specific travel resource. For example, when the ebXML registry is queried for hotel Bellagio, the result of the query will return three nodes (representing the provider, type, and location). This is only possible because these nodes have the same content in their **value** attributes. For more information, refer to [53].

It is very important to establish the name mechanism for the **value** attribute. For this reason, the content of **value** attributes is the UUID (the globally unique identifier [5]) and the name of the site. The inclusion of the UUID in the content of the **value** attribute is to help differentiate the sites with the same name (e.g. Dancing Zorba restaurant in Conway, AR and Dancing Zorba pub in Muskogee, OK). When the ebXML is queried for Dancing Zorba, the three nodes that represent each site can be easily differentiated (because they have different UUID). Besides sending the index token, the search agent can send a little more information about the site to help the indexing agent decide what value to use for the name of the site. This information can be included in the `Property` object as well. It is hard for the indexing agent to decide what name to use for the site if the only information passed between indexing agent and search agent is the index token. The indexing agent can connect to the provider (from the provider component of the index token) and decide what name to use for the site.

This requires an additional connection to be made to the provider site by the indexing agent, whereas this information can be gathered by search agent when it

was visiting the site. The details on how the search agent decide the name for the site will be described in Chapter 4. The information passed between indexing agent and search agent will be as follow (in the form of `Properties` object):

```
...
Properties p = new Properties();
p.setProperty("provider","http://www.moulinrouge.com");
p.setProperty("type", "opera");
p.setProperty("location","New Orleans,LA");
p.setProperty("name","Moulin Rouge Opera");
p.setProperty("zip","70112");
...
```

Listing 3.7: Properties Object Passed by Search Agent

As you can see from the above example, two additional properties have been added (e.g. name and zip). The name property will help indexing agent in deciding the name for the size. The zip code property plays an important role for performing GIS reverse geo-coding (see page 35).

3.4.3 Inserting the Index Token

There is a possibility that the index token returned by search agent is not complete (see Chapter 2 Section 2.2 Sub Section 2.2.4). For instance, This is true most of the time because most of the travel providers do not list their location in the form of latitude and longitude information. Using the `<?notes?>` component, this token can be marked as incomplete. The `<location>` component plays a special role in this case (see Sub Section 3.4.1) so it will not in the form of latitude and longitude. For the purpose of this thesis, the index token returned by the search agents will be considered to be complete. The procedure for inserting the index token into the registry follows the flowchart depicted in Figure 3.3.

The first thing before the index token is inserted into the registry is to check if it

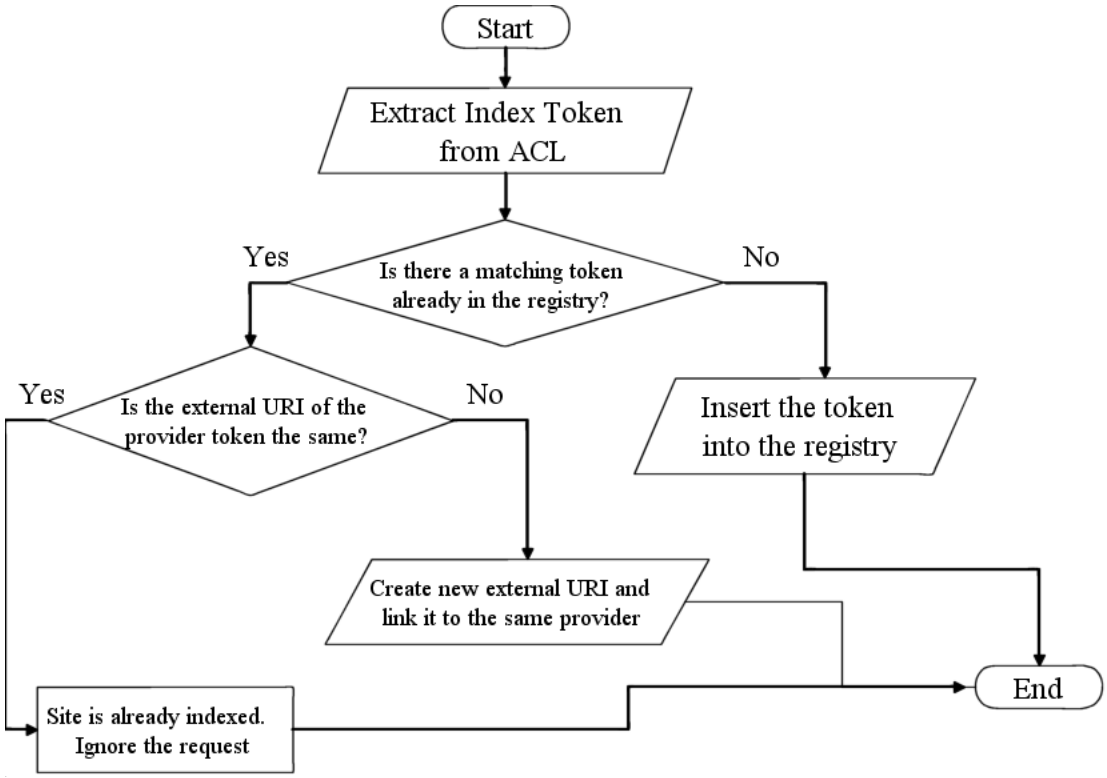

Figure 3.3: Flowchart for Inserting Index Token

has been inserted before. To compare the index token, it must be broken down into its components (e.g. `provider`, `type`, and `location`). We consider two sites to be the same if all three of their components match (their name attributes match). If the `type` and `location` component of two sites are equal but their `provider` component are different, then the new provider will be added to the list of provider for the site. Having more than one provider for a site means redundancy because if one of the provider goes down, there are other providers that can be reach to get the content. Also different sites mean different information.

In the ebXML registry repository, the URI of the provider is represented as `ExternalLink` object. There is no limit on how many URI a provider can have. The URI is linked to the provider component using `Association` object of ebXML

registry repository. When creating the `Association` object, the type must be set as `ExternallyLinks`. The ebXML registry has a pre-defined `AssociationTypes` classification scheme (see Figure 3.4). One of the child nodes of this classification schemes is the concept `ExternallyLinks`. This concept is then set as the `Association` type. The code snippet below describes this purpose (Listing 3.8):

```
...
//creating ExternalLink object
ExternalLink el = blcm.createExternalLink(URI,
 blcm.createInternationalString("Description"));
el.setName(blcm.createInternationalString("Name"));
//creating Association object
//the second argument is where we specify the
//association type (ExternallyLink)
Association ass = blcm.createAssociation(parent,
 bqm.findConceptByPath("%ExternallyLink%"));
ass.setName(blcm.createInternationalString("Name"));
ass.setDescription(blcm.createInternationalString(
 "Description"));
//now associating ExternalLink with the Provider
//component
el.addAssociation(ass);
....
```

Listing 3.8: Creating External Link Object

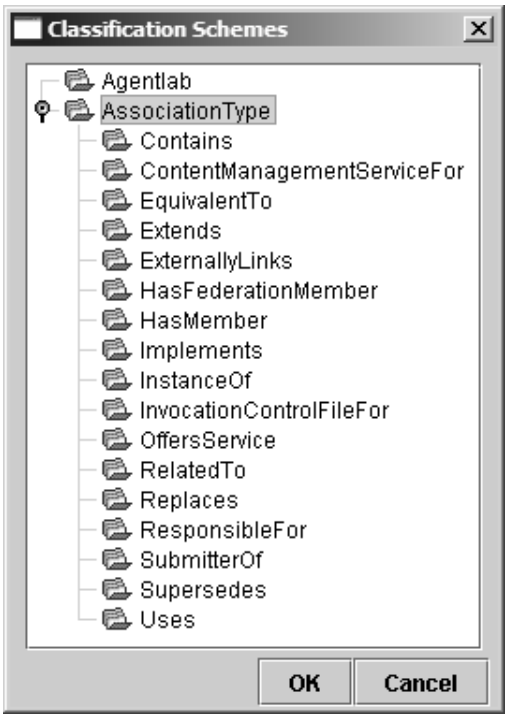

Figure 3.4: The Association Type in ebXML Registry

The `slot` object of ebXML registry will be utilized to hold GIS information (e.g. latitude and longitude). Indexing agent relies on GIS agent to convert location information (from `location` component) into the form of latitude and longitude. In order to do this, indexing agent send an ACL message containing the zip code of intended location. Using this zip code, the GIS agent will resolve this address into the form of latitude and longitude. The GIS agent is discussed in section 3.5. The code snippet below show how to add slot into `location` component (Listing 3.9):

```
...
//create collection of values
Collection values = new ArrayList();
values.add("Latitude=36,10,42");
values.add("Long=-115,10,24");
//now create slots and assign its values
Slot slot = blcm.createSlot("Name",values,"Type");
//assign this slot to a concept
aConcept.addSlot(slot);
...
```

Listing 3.9: Adding Slot into Location Component

The design of indexing agent is illustrated in Figure 3.5. From the figure, you

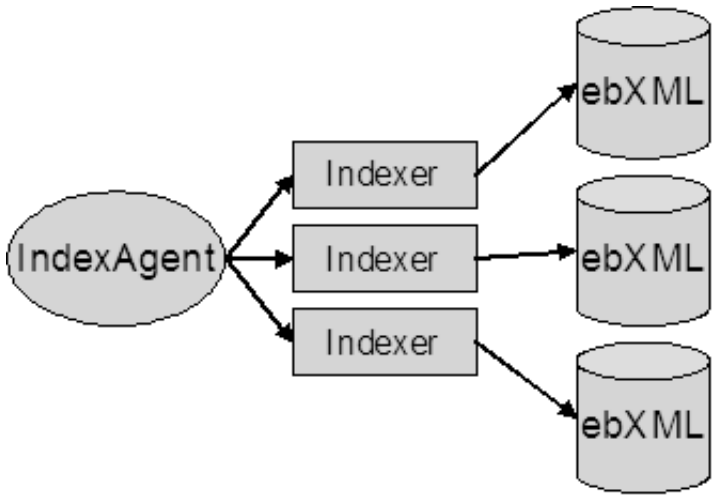

Figure 3.5: Indexing Agent Design

can see that the indexing agent utilizes `Indexer` class to interface with the ebXML registry. The advantage of this design is that indexing agent can interface with

different ebXML registry. It also provide fail over if one ebXML registry is down, then the insert operation can be switch to other registry easily.

The complete implementation of indexing agent can be found in Appendix A.

3.5 GIS Agent

The main function of GIS agent is to resolve location information into latitude and longitude pair. The latitude and longitude information are important for performing proximity search. For example, a person can perform a search on the ebXML registry for any restaurant within 10 miles radius of her current location. Without latitude and longitude information, such query would have not been possible. Current implementation of GIS agent relies on external party to provide reverse geo-coding functionality. The advantages and disadvantages of this approach have been presented in Table 2.2. In the future, another approach can be followed (e.g. relying on web services using JAX-RPC to query GIS information from Microsoft TerraServer [52] or implementing internal GIS subsystem).

For the purpose of this thesis, the GIS agent relies on the web site `http://mapper.acme.com` to resolve GIS queries. This website accepts the request for GIS queries interactively (e.g. using form and input box). This can be easily transformed programmatically using Java's `HttpURLConnection` class. The form is submitted using the `GET` method thus this can be represented as a URL by appending the base address of the website with parameter and value pair of intended queries. (e.g. `http://mapper.acme.com/find.cgi?zip=74075`).

The GIS agent obtain the ZIP code from the ACL message sent by indexing

agent. Indexing agent wraps the ZIP code inside the ACL message (in the content slot). When this message is received by GIS agent, the ZIP code can be extracted from the ACL message. Using the ZIP code, GIS agent will perform the reverse geo-coding and send back the result to indexing agent using ACL message.

When GIS agent is created the first time, it will register itself with the Directory Facilitator (see yellow page service on page 23) so that indexing agent can find it. This will also allow more than one GIS agent serving indexing agent (for load balancing). With yellow page service, the indexing agent does not need to know in advance the name and location of GIS agent. Instead, indexing agent can query the Directory Facilitator for any available GIS agent and thus establish communication directly.

The solution presented here for GIS subsystem is meant to be temporary. It hard to ensure reliability using third party subsystem especially if it is free. A better solution is needed. One solution is to create in house GIS subsystem.

The complete implementation of GIS agent can be found in Appendix B.

Chapter 4

Search Agent

The main function of search agent is to find travel related information from the Internet. Once the travel information is found, it then creates index token (see page 11) and send this token to indexing agent. In order to create index token successfully, semantic understanding must be introduced to search agents. The semantic understanding is crucial component of the search agents primarily because they need to decide if the given Internet resource are in fact travel related. This poses several problems like how to teach search agents to differentiate between different travel resources and how to locate necessary information in order to create index token. Many of information retrieval techniques can be used to help solving these problems. There is also another problem with information retrieval itself (afar from understanding semantic context of the information), especially information retrieval from the Internet. According to [4], the problems of information retrieval from the Internet are as follows (viewed from the data perspective):

- Distributed data

  Internet is a very dynamic and distributed network of computers. Data and information are scattered all over the interconnected network. Each computer or network has its own method of retrieval (or protocol in this case) that is not

necessarily compatible with each other. This problem is further discussed in [6].

- High percentage of volatile data

  Internet is a highly dynamic place where computers can come and go as they like. This means data that are available at one site may not be available again within seconds of time. The rate of how data change is also very fast, where data that were valid at the moment may not be valid in the next 10 seconds.

- Large volume

  There are more than 7 billion web pages (see Chapter 1) on the Internet that contains several terabytes of data. This number makes finding the desired information harder especially in locating the information resources.

- Unstructured and redundant data

  The data from the Internet are usually represented as HTML pages. HTML page are considered as semi-structured data. Semi-structured data are harder to be analyzed because their structure is unknown. It is hard to locate desired information from web pages when their structures are unknown.

- Quality of data

  With large amount of information available, there is no method to prove their validity and correctness. Information from one site may conflict with information from other sites. That is why in Chapter 1, the deconfliction agent was introduced.

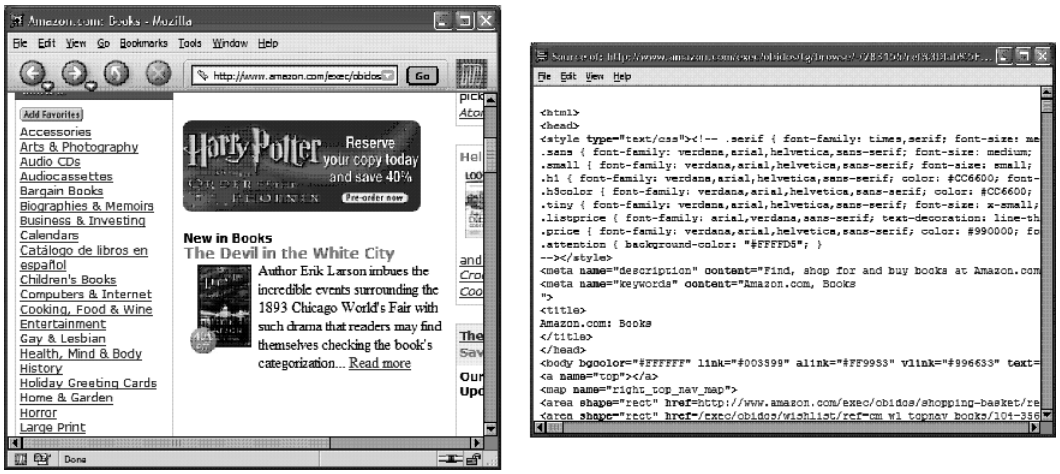

Figure 4.1: Human Eyes vs. Computer Eyes

To illustrate this problem, Figure 4.1 displays Amazon.com website in two different formats. The picture on the left displays the website the way human sees it and the picture on the right displays the website the way computer sees it. Based on this illustration, the following issues arrives (see [3] for further information):

- Information on the web page can be easily understand by human, whereas computer needs further processing of this document in order to extract semantic meaning.
- The page layout changes periodically thus computer needs to adjust to this changes in order to avoid extracting invalid information.
- Sometimes there are some intermediate steps required in order to reach the desired page (e.g. involves using password and user name).

Despite all these problems, there have been many researches in the area of information retrieval and semantic understanding. One particular emerging area of research is the Semantic Web [49]. Semantic web is discussed in the next section.

4.1 Introducing Semantic Understanding

The search agents must be designed so that they can understand the content of the information, e.g. for classifying a web page into the correct travel resource and finding necessary information to create index token. There are many ways this can be accomplished although their methods are not 100% foolproof. For example, some ranking model (using statistical method) can be used to calculate the similarity of a web page to a given keywords (or query).

Figure 4.2 illustrates a simple statistical ranking. First we identify the most common keywords that are used for each travel resource. For example, in the Figure 4.2, we have identified 7 keywords (e.g. room, double, single, view, floor, hotel, rates) that are commonly used to represent a hotel. Using these keywords, we will perform the statistical ranking on a web page. Before a web page can be compared, it must be indexed first to find its unique terms. In this example, we have web page A and web page B (with their own unique terms). Simple matching can be perform by adding 1 to the total rank value for each of the terms that match the keywords. On web page A, there are 4 matching term, thus the rank value is 4 (there are two matching terms in web page B). For weighted match, instead of adding 1 to total rank value, for each term that match the keywords, the number of times that terms appear on the web page are added to total rank value. For example, on web page A, the term "room" appears twice, thus the value of 2 are added to total rank value (and three times for term "single"). There are other advanced ranking models (e.g. Vector Space Model and Probabilistic Model). Their implementation will not be discussed in detail here

(please see [17] and [4] for very detailed discussion on this topic). The Vector Space Model ranks query against individual documents [17]. The search agent will utilize the Vector Space Model because it is very easy to be calculated. The Jakarta Lucene package [33] that is used by search agents to perform indexing of web pages provides implementation of Vector Space Model.

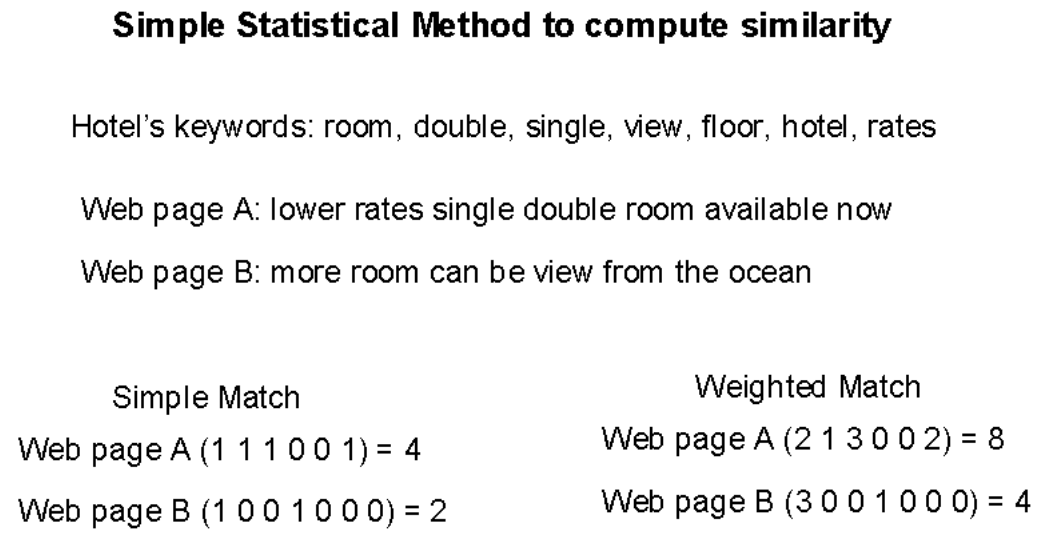

Figure 4.2: Statistical Method to Compute Similarity

4.1.1 Identifying Travel Resource Type

The simple and weight matching had been introduced previously. The same basic idea (with some improvements) will be used by search agents in order to categorize web pages into each travel resource type. The Jakarta Lucene package provides several convenience classes that can be used to perform the similarity calculation.

Before these web pages are fed into Lucene, they must be properly filtered to eliminate unnecessary information (e.g. Java Script codes, HTML Tags, URLs, stop words, etc.). The list below describes the filtering process and the tools involved:

1. HTML Filter

   This filter eliminates most HTML formatting commands from the web page. These formating commands are not valid keywords and their existence will jeopardize index terms produce by Lucene (they become index terms that represent the travel resource). Besides formatting commands, any scripting language must also be eliminated for the same reason. The HTMLParser package [22] is used for this job. Using HTMLParser package, the web page can be parsed and any desired information can be extracted (e.g. extracting links, images, text, title, etc.). To make text extraction easier, HTMLParser provides a `StringBean` class which is a Java Bean class. Using `StringBean` class, extracting text content from web page can be accomplished in just a few lines of code. Listing 4.1 shows how to accomplish this task.

```
import org.htmlparser.beans.StringBean;
...
StringBean sb = new StringBean();
//do not extract text from link
sb.setLinks(false);
//convert %20 to space
sb.setReplaceNonBreakingSpaces(true);
//multiple spaces become one space
sb.setCollapse(true);
//set the URL
sb.setURL("http://www.someurl.com");
//get the text
System.out.println(sb.getString());
...
```

Listing 4.1: String Bean Class of HTMLParser

   Using `StringBean` class, the HTML file will be stripped out and it will produce pure text output.

2. Lowercase Filter

   Lowercase filter converts the input text into lower case letter. This is useful because we do not want to differentiate between "Sun" and "sun". Lucene

[33] provides the implementation of Lowercase filter using `LowerCaseFilter` class. In fact, Lucene implementation allow us to implement multiple filter altogether using the `Analyzer` class. See Appendix C for the implementation of this `Analyzer` class.

3. Stop Words Filter

   Stop words filter removes stop words from the text. Stop words is basically words that are useless for semantic recognition (e.g. first, second, am, are, etc.). Their existence do not change the semantic meaning of the text. If these words are not removed from the text, they will be included in the index terms thus they increase the size of the index. Lucene has the `StopFilter` class that filters stop words. But its stop words list is not complete. For this purpose, the stop words listed in Appendix C are gathered from [32]. These stop words will be stored in XML file so they will be easier to be added or removed later. The `StopFilter` class read these stop words by using `MySAXParser` class that implements SAX parser for reading XML files.

4. Porter Stem Filter

   As its name implied, Porter Stem Filter was invented by Martin Porter [47]. According to [47], Porter stemming algorithm is "a process for removing the commoner morphological and inflectional endings from words in English". Porter Stem Filter basically convert words into its basic form (e.g. rooms → room, watches → watches, etc.). Lucene also come with `PorterStemFilter` class that implements Porter Stemming Algorithm. See Appendix C for the implementa-

tion of `Analyzer` class that uses `PorterStemFilter`.

The four steps of filtering process is illustrated in Figure 4.3. The HTML Filter is implemented by using HTMLParser Package [22], and the rest of the filtering process is implemented by using Lucene. The pure text can be fed to Lucene to create index terms and these index terms are then ready to be compared.

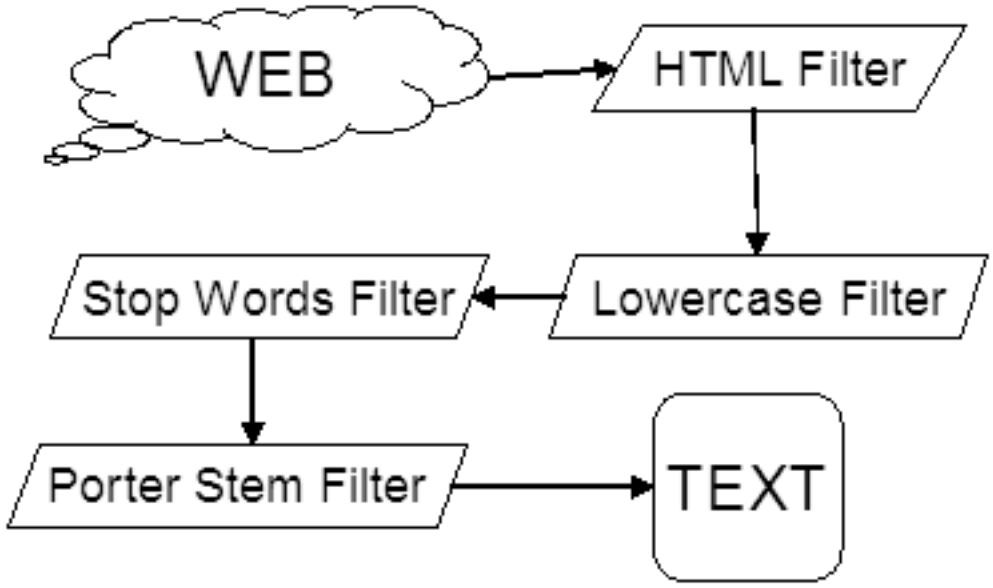

Figure 4.3: The Complete Filtering Process

The similarity value is calculated based on this formula (based on Vector Space Model):

$$similarity(d_j, d_k) = \frac{\sum_{i=1}^{n}(td_{ij} \times tq_{ik})}{\sqrt{\sum_{i=1}^{n} td_{ij}^2 \times \sum_{i=1}^{n} tq_{ik}^2}}$$

where td_{ij} = the i^{th} term in the vector for document j, tq_{ik} = the i^{ith} term in the vector for query k, and n = the number of unique terms in the data set. Luckily, `Lucene` comes with `Similarity` class that perform the same similarity calculation (using the same Vector Space Model). The `Similarity` class is very flexible, it allows user to

plug-in other similarity calculations (e.g. probabilistic model).

The filtered web sites (contain text and no HTML formatting) are then compared with the travel resource type keywords. Their similarity value are calculated using Vector Space Model. Lucene provides automatic calculation of the similarity value using `Hits` object. The XML file that contains the keyword can be parsed (using SAX or DOM). In this case, SAX is utilized because it is simpler and does not consume memory as much as DOM (refer to Java Web Services Tutorial [41] for more information on this topic). These keywords along with the `Query` object are used by Lucene to calculate the similarity. The code snippet below describe this purposes (Listing 4.2):

```
//create new Searcher object
Searcher searcher = new IndexSearcher("index name");
//create new Query object
Query query = queryParser.Parse("put keywords here",
  "field name", new MyAnalyzer());
//perform the search using the query
Hits hits = searcher.search(query);
//print out the score
System.out.println(hits.score(0));
...
```

Listing 4.2: Calculating Similarity Value

These are the clues that search agent can use to find the correct travel resource type for any web pages:

- Web Page Title

  Normally a web page will have the title that represents its content. For example, the web page title for Bellagio Hotel in Las Vegas is "Bellagio Hotel & Casino, Las Vegas". From this title, the search agent should be able to choose the correct travel resource type for this web page. Although we must also acknowledge that this is not always 100% true. There are some web sites that do not put travel

type on the title of its web site. Sometimes web sites titles are also misleading. Search agent will not only rely on the web page title to decide its travel resource category. The similarity value computed from the title with be matched against other parts of the web site.

- The <META> Tag of HTML Page

  The <META> tags are located in the header of HTML page. They are designed so that it is possible to insert additional text (e.g. information, description, author, etc.) without displaying it. Many search engines rely on the information described in the <META> when performing the searching indexing, and ranking. <META> tags have two possible attributes:

  1. <META HTTP-EQUIV="name" CONTENT="content">

     This tag control the action of the browser thus it does not related to our purposes. This tag is use for the equivalent of HTTP headers.

  2. <META NAME="name" CONTENT="content">

     In this attribute, the author of the page can add additional information (e.g. keywords, author, language, robot, description) into the web page. This attribute basically works like meta data in the database. Meta data describes the content of the database. For HTML page, the <META> tag with NAME attributes describes the content of the web page in a few simple words. Listing 4.3 gives <META> tags example:

```
<meta name="keywords" content="las vegas,vegas,Bellagio,
bellagio hotel,Bellagio casino,bellagio las vegas,
belagio,casino,luxury resort,restaurants,dining,
```

```
entertainment,weddings,shows,cirque du soleil,0,
meetings,conventions,banquets">
<meta name="description" content="Bellagio Hotel and
Casino in Las Vegas is the premiere vacation of elegance
and luxury.  Reserve your room online today!">
...
```

Listing 4.3: <META> Tag Example

Search agent can use the information in the <META> tag to map the web page to the correct type of travel resource. From the Listing 4.3, we can tell that Bellagio is hotel, casino, restaurant and resort. Using the HTML-Parser [22] package, the <META> tag can be extracted easily (Listing 4.4).

```
Parser parser = new Parser(
  "http://www.someurl.com", null);
parser.registerScanner();
Node[] nodes = parser.extractAllNodesThatAre(
  MetaTag.class);
for (int i=0;i<nodes.length;i++) {
  MetaTag meta = (MetaTag) nodes[i];
  System.out.println(meta.getMetaTagName()+
    "= "+meta.getMetaContent();
}
...
```

Listing 4.4: Extracting <META> Tag

- The Link (<A HREF>) Tag in HTML Page

  As described in [35], hypertext links in the HTML page can aid the understanding of semantic content of the web page. Since this topic is still under research, search agent will not exploit the links for semantic recognition.

- The URL of the Web Page

  Most of the time, the URL of the web page contains the type of travel resource it represents. For example, http://www.applesbedandbreakfast.com/ contains the word "bedandbreakfast". Since the URL address represents the front end of the business, we are able to rely on the URL address to match the web

page to its the travel type. Hilton Hotel will likely to register itself as `http://www.hiltonhotel.com` instead of `http://www.hiltonbar.com`. Of course this is not always true all the time. Thus search agent will not only relies on this information to decide the category of the web page.

- The Body of the Web Page

  The body of the web page is the place where it contains most information. It is the place where the search agent can rely on to find the correct travel resource category for a web page. The body of the web page will be indexed, their similarity to each travel keyword will be compared. The one with the highest value is retained.

4.1.2 Identifying Location Information

One of the index token (see Page 11) components is the location component. According to [53], location component is "the key factors for determining the relevance of indexed travel resources to a particular user's travel plans". Location component in the ebXML registry is designed to be hierarchical (based on ISO-3166 standard).

Location information can be in the form of city, state or zip code. The zip code is required in order to perform GIS reverse geo-coding (see GIS Agent on Page 35). The city and state is required to find the right location for inserting location nodes (see index token on Page 11). To find location information from a web page, we can use both HTMLParser [22] (to extract the text from the web page) and Lucene [33] (to tokenize the text). First of all, given a URL address, the location information is not necessarily located on the first page. It might be necessary to extract all the

links in the page and examines the web pages pointed by these links until we find a suitable location information. Using the HTMLParser [22] package, it is very simple to extract all the links from the web site. Listing 4.5 shows how to extract links:

```
Parser parser = new Parser("http://www.anyurlyouwant.com",null);
parser.registerScanner();
Node[] nodes = parser.extractAllNodesThatAre(
  LinkTag.class);
for (int i=0;i<nodes.length;i++) {
  LinkTag link = (LinkTag) nodes[i];
  System.out.println(link.getLink());
}
...
```

Listing 4.5: Extracting Links From Web Page

When the links are extracted, we do not want to include links that are located on a different host (e.g. `http://www.hotmail.com` and `http://www.google.com` are two different host). The reason is simply because the information on one host are totally irrelevant with information on the other host.

To locate location information, the filtered web page will be fed into Lucene [33]. From Lucene, the index term can be extracted in the form of tokens. These token will be examined for occurrences of any sequences of number that resembles a zip code. This can be accomplished by using regular expression (e.g. "\d{5}(-\d{4})?"). Java supports regular expression since version 1.4. Once possible zip codes have been retrieved, these zip codes will be compared to the United States Postal Service (USPS) database. USPS provides web interface where one can query the database using the zip code to get the associated city and state. Using the information from USPS database, we can increase the accuracy of finding the right city, state and zip code in the web pages.

4.2 Search Agent Implementation

There are several layers of search agent as depicted in Figure 4.4. The first layer is called the `DistributorAgent`. This agent accepts the start URL and then extracts all the links from this URL. Once the links are extracted, they will be passed to `SiteAgent`. SiteAgent will examine these links and launch specific search agents for each of this links. Most links are HTML links (they contain text) but some links point to non text content (e.g. `PDF`, `DOC`, and others). For now only text content is supported, and it is implemented in `SearchHTML` class. See Appendix C for the implementation of these agents.

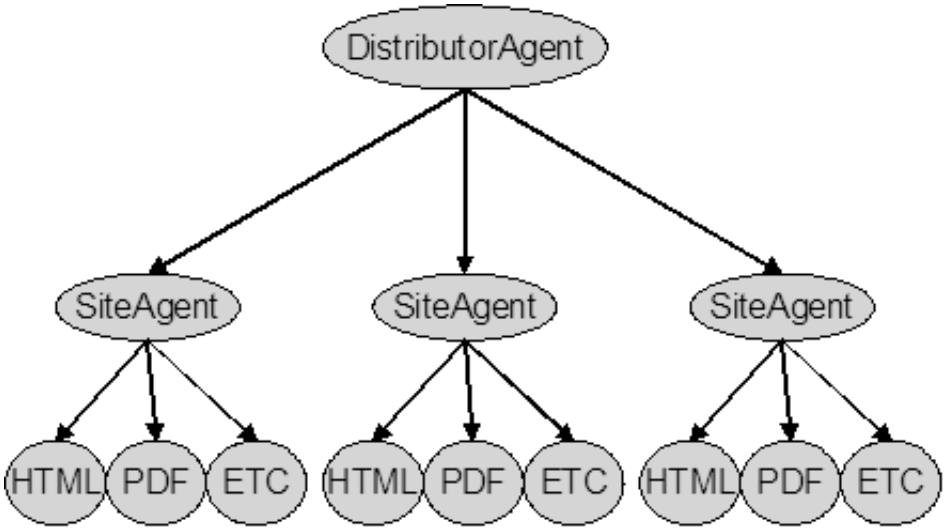

Figure 4.4: Search Agent Design

Search agent is designed to be similar like most web crawler. Given a start URL address, it will extract text and links from the web pages. The text will be examined and location information will be extracted. If location information can not be found on the start URL, the search agent will continue to extract text from the URL

pointed by the links. The procedure for extracting location information is described in Subsection 4.1.2. This procedure will fail if the location information is stored in a graphic files (e.g. `JPEG`, `GIF`, and etc.) because they simply do not produce pure text. If the location information can not be extracted from the start URL and any URL pointed by the links from the web page, this URL will be ignored.

After the location information is found (e.g. city, state and zip code), the web page will be classified to it's travel resource type. The procedure to classify a web page into their travel resource type is described in Sub Section 4.1.1. Most web sites describe their travel type in the `<TITLE>` tag or in the `<META>` tag of the `HTML` page although this is not always true. Search agent will examined these tags first. If examining these tags does not produce a valid classification, the keyword approached will be performed. If any of these approaches do not yield a travel resource, these URL will be ignored. The reason is simply because the web page is probably does not provide travel services although they have valid location information. We also must acknowledge that search agent is not always right. Sometimes it will fail to identify web pages with correct travel resource information. These conditions are likely to happen if location information is stored in non text format (inside a graphic files) or such information is very obscured thus search agents decide to ignore it.

For the name information, e.g. name of the sites. We can rely on the `<TITLE>` tag, `<META>` tag or the content of the web page itself. The procedure of extracting meaningful name will fall under the research for document summarizer e.g. in [29, 34], which will introduce another topic of research. For this thesis, the name will be simply the domain name of the URL (e.g. `hiltontulsa` for `http://www.hiltontulsa.com`).

After all the required information have been gathered, index token can be created and inserted to the ACL message. The search agent can look for the indexing agent by querying the Directory Facilitator (DF). This method is described in Listing 4.6:

```
AID index = new AID();
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("IndexingAgent");
dfd.addServices(sd);
try {
  while (true) {
    SearchConstraints c = new SearchConstraints();
    c.setMaxDepth(new Long(3));
    DFAgentDescription[] result =
    DFService.search(this,dfd,c);
    if ((result != null) && (result.length > 0)) {
      dfd = result[0];
      index = dfd.getName();
      System.out.println("Found␣Indexing␣Agent");
      break;
    }
  }
}
catch (FIPAException e) {
  System.out.println("Cannot␣Find␣Indexing␣Agent");
}
```

Listing 4.6: Querying the DF for Indexing Agent

When this method returns, the variable `index` will be pointing to the Agent Identified (`AID`) of the indexing agent.

4.2.1 User Interface for Search Agent

JADE [13] provides dummy agent (a GUI agent) that allows user to interact with other agents by sending ACL message (see Figure 4.5). Although dummy agent provides this convenience, its features does not satisfy our need for interacting with search agents. With dummy agent, we are only able to start search agent based on one seed URL (one ACL message at a time). We also want to start search agents based on the query results from Google, from set of URLs in a file, or from Yahoo! Search. In order to do that, a `GUIAgent` will be constructed. `GUIAgent` is depicted

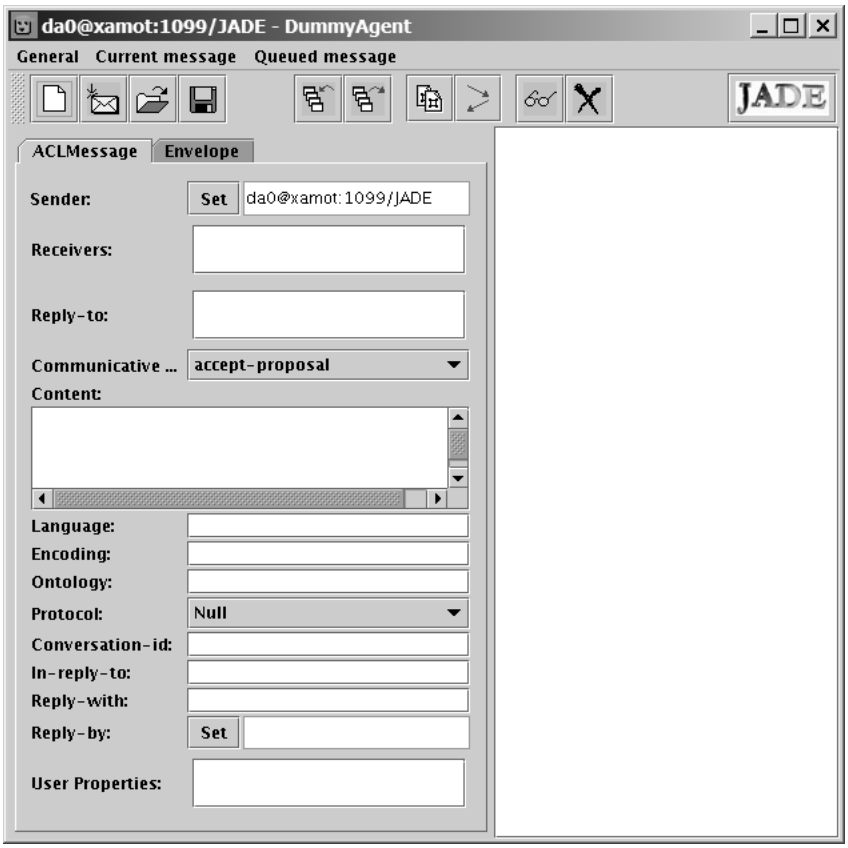

Figure 4.5: The JADE's Dummy Agent

in Figure 4.6. `GUIAgent` is another type of JADE's agent that provides graphical user interface (using Java Swing). JADE provides a convenience class `GuiAgent` for constructing agent with GUI. When designing GUI agent, we must be careful not to mix Swing event based thread and JADE agent thread. JADE uses one thread for each agent and Swing event dispatcher runs on its own thread. If these two are not synchronized, then one of the threads will not be running at all. JADE programmer's guide [13] recommends using `GuiAgent` when designing GUI agent.

The `GUIAgent` provides many conveniences for the user to initiate the search. As you can see from Figure 4.6, there are four possibilities for starting search agents. This features will be used when we conduct the performance measures on seach agents.

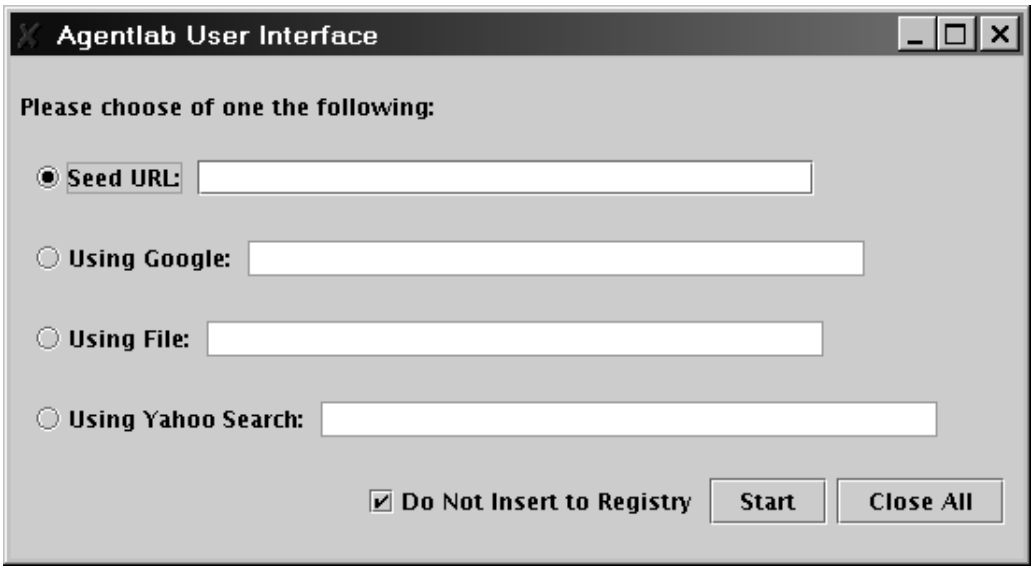

Figure 4.6: The `GUIAgent`

The GUI design of search agent is described in `GUIDesign` class which is the Swing `JFrame`. When `GUIAgent` starts, it will display the `GUIDesign` to the screen. The events are posted using `postEvent()` method. The GUI agent also provides a switch so that any results of the search will not be inserted into the ebXML registry (for some testing).

4.3 Performance Measures

In this section, the accuracy of search agents to identify travel resources type is examined. To be fair and exhaustive, the test must be complete. In other words, it must be as much as possible try to emulate the environment the search agent will encounter in the real world. In this case, the environment is the Internet where the amount of data is huge and dynamic.

For this test, we will be relying on Google and Yahoo! Search to generate random URLs. Google provides Web Services interface (known as Google API) to its search

engine allowing external program (e.g. Java, VB, C# and any other languages that support Web Services) to perform query directly. There are two limitations with Google API: the maximum results for each query is limited to 10 and total number query allowed for each day is 1000. Yahoo! on the other hand, does not provide any APIs that can query its search engine directly. For this case, `YahooParser` class is created to parse the search results return by Yahoo!.

In order to avoid overloaded test data and overcome some limitations (e.g. Google limit for daily queries), we will generate the test URLs based on the query for the cities in Oklahoma. For example, there are 691 cities in Oklahoma and 54 travel resource types which can be combined together to create query string (e.g. stillwater hotel, tulsa museums, and etc.) that can be fed into Google and Yahoo!. For this test, we will not be using all possible combination because with current limitation it will takes 40 days to finish the queries (691 cities $\times$ 54 categories = 37314 search queries). We will instead choose six categories ranging from obvious (e.g. hotel) to atypical (e.g. safari) for the this test and pair them with 10 random cities in Oklahoma.

For this experiment, we will define several attributes to be used in our test. The attributes are listed on Table 4.1 and 4.2. The search string attribute defines the keyword used for performing the search. Number of URLs attribute is total number of URLs produced by the search engine for that particular keyword. Identified site is the number or URLs that are identified by the search agents (may or may not be correct). Correct attribute represent the number of identified site that are correctly identified. Incorrect location attribute defines number of identified site with incorrect location information. Incorrect type defines number of identified site that are incorrectly

categorize to its type. And the last, failed attributes represent URLs that contain valid travel resources but failed to be identified by the search agents.

The result of the experiment using Google is shown in Table 4.1. For each six keywords chosen (ranging from obvious to weird), we choose 10 cities which makes total of 10 query strings for each keyword. Coupled with the 10 URLs from each result of the Google query, each keyword produces 100 URLs to be examined.

The experiment using Yahoo! produced more URLs because the search results are not confined by Google 10 search results limit. Although there are some advantages of not having this limitation, it also produces huge number of search results (e.g. simple search using using "stillwater hotel" as query string produces 49,200 URLs) which makes the experiment become infeasible. Not only that, since the results are also sorted by their rank value, the URLs at the end of the results becomes so far related from the original context of the query. To avoid that, we limit the Yahoo! search results to contain only 20 URLs. The experiment using Yahoo! is illustrated in Table 4.2: In this experiment, we perform the same type of experiment as yahoo whereas each here each query string produces 20 URLS thus make up the total URLs for each keyword becomes 200 URLs. Table 4.3 and Table 4.4 display the summary for the both experiments. The overall success rate is less than 25% which is contributed by several factors as described in the next paragraph.

After performing these experiments with thousand of URLs, there are several remarks that can be made. These remarks serve as author's conclusion for the experiment performed:

- the accuracy of the search agents depends on the quality of URLs examined. The quality in this context refers to how the page is arranged (location of essential keyword that represent the site e.g. travel type and location information).

- as the experiment moves from obvious to weird keyword, we noticed the decreasing of correctly identified site. This is caused by the inability of the search engines to locate appropriate site (URLs).

- the high number of identified site does not guarantee the increase of search agents accuracy. In fact, as number of identified site increase, they place a burden on the search agent to identified these sites correctly. Incorrectly identified sites decrease the accuracy of the search agents.

- the search result produced by Google and Yahoo! contributed to the small accuracy value of the search agents since the results do not represent appropriate site. For example, performing search on Google or Yahoo! for "tulsa safari" will produce many unrelated hit URLs for search agents to comprehend.

Above all, carefully examining the successful identifications reveal that the search agents work very well with URLs that represent their content informatively (no hidden information or gotchas). Search agents failed most of the time with site that represent their content with graphics and absurd tables.

Hotels

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	stillwater hotels	10	5	4	1	0	1
2	tulsa hotels	10	4	4	0	0	0
3	oklahoma city hotels	10	3	2	0	1	2
4	enid hotels	10	5	3	2	0	1
5	ponca city hotels	10	5	3	2	0	1
6	lawton hotels	10	6	4	1	1	0
7	muskogee hotels	10	5	5	0	0	0
8	edmond hotels	10	5	3	2	0	1
9	guthrie hotels	10	6	4	2	0	0
10	guymon hotels	10	6	4	2	0	1
	Total	**100**	**50**	**36**	**12**	**2**	**7**

Restaurants

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	tulsa restaurants	10	2	2	0	0	0
2	oklahoma city restaurants	10	8	8	0	0	1
3	stillwater restaurants	10	6	2	0	4	1
4	edmond restaurants	10	2	0	0	2	2
5	broken arrow restaurants	10	3	0	0	3	2
6	enid restaurants	10	4	3	0	1	0
7	del city restaurants	10	4	1	0	3	2
8	guthrie restaurants	10	4	1	0	3	1
9	shawnee restaurants	10	4	2	0	2	0
10	ada restaurants	10	3	0	0	3	0
	Total	**100**	**40**	**19**	**0**	**21**	**9**

Museums

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	tulsa museums	10	5	3	0	2	0
2	oklahoma city museums	10	4	3	0	1	1
3	guthrie museums	10	6	4	0	2	1
4	enid museums	10	4	3	0	1	0
5	norman museums	10	3	3	0	0	2
6	clinton museums	10	7	5	0	2	1
7	ardmore museums	10	5	5	0	0	1
8	bartlesville museums	10	3	2	0	1	0
9	claremore museums	10	4	4	0	0	1
10	duncan museums	10	5	2	0	3	2
	Total	**100**	**46**	**34**	**0**	**12**	**9**

Ranches

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	oklahoma city ranches	10	4	0	0	4	0
2	chandler ranches	10	2	2	0	0	0
3	tulsa ranches	10	4	2	0	2	1
4	muskogee ranches	10	1	0	0	1	0
5	claremore ranches	10	2	1	0	1	0
6	sayre ranches	10	2	1	0	1	0
7	lawton ranches	10	3	1	0	2	0
8	stillwater ranches	10	3	0	0	3	0
9	okmulgee ranches	10	1	0	0	1	0
10	guymon ranches	10	3	0	0	3	0
	Total	**100**	**25**	**7**	**0**	**18**	**1**

Rafting

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	tahlequah rafting	10	7	2	0	5	0
2	tulsa rafting	10	3	2	1	0	0
3	tecumseh rafting	10	3	1	0	2	1
4	ardmore rafting	10	3	1	0	2	0
5	cleveland rafting	10	3	1	0	2	0
6	miami rafting	10	3	2	0	1	0
7	enid rafting	10	4	3	0	1	0
8	broken arrow rafting	10	3	2	0	1	0
9	ponca city rafting	10	4	0	0	4	0
10	bartlesville rafting	10	5	0	0	5	0
	Total	**100**	**38**	**14**	**1**	**23**	**1**

Safari

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	tulsa safari	10	6	3	0	3	0
2	oklahoma city safari	10	3	3	0	0	1
3	edmond safari	10	1	0	1	0	0
4	enid safari	10	2	0	0	2	0
5	broken arrow safari	10	8	3	0	5	0
6	norman safari	10	2	0	0	2	0
7	claremore safari	10	6	0	0	6	0
8	owasso safari	10	1	0	0	1	1
9	weatherford safari	10	0	0	0	0	0
10	tahlequah safari	10	3	0	0	3	0
	Total	**100**	**32**	**9**	**1**	**22**	**2**

Table 4.1: Result of the Experiment Using Google

Hotels

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	stillwater hotel	20	9	6	2	1	0
2	tulsa hotel	20	13	10	3	0	0
3	oklahoma city hotel	20	7	5	0	2	0
4	edmond hotel	20	6	5	1	0	1
5	enid hotel	20	7	5	1	1	0
6	muskogee hotel	20	6	4	1	1	2
7	norman hotel	20	2	1	1	0	2
8	guthrie hotel	20	5	4	1	0	0
9	ponca city hotel	20	5	4	0	1	0
10	lawton hotel	20	6	5	1	0	0
	Total	**200**	**66**	**49**	**11**	**6**	**5**

Ranch

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	guymon ranch	20	10	2	0	8	0
2	tulsa ranch	20	9	2	0	7	0
3	chandler ranch	20	9	3	0	6	1
4	alva ranch	20	5	1	0	4	0
5	claremore ranch	20	7	0	1	6	0
6	broken arrow ranch	20	10	7	0	3	0
7	edmond ranch	20	11	4	0	7	0
8	sayre ranch	20	4	0	0	4	0
9	enid ranch	20	5	1	0	4	0
10	idabel ranch	20	7	1	0	6	1
	Total	**200**	**77**	**21**	**1**	**55**	**2**

Restaurant

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	tulsa restaurant	20	8	6	0	2	0
2	oklahoma city restaurant	20	9	6	0	3	0
3	norman restaurant	20	3	3	0	0	1
4	stillwater restaurant	20	8	6	0	2	0
5	edmond restaurant	20	9	4	0	5	0
6	enid restaurant	20	7	7	0	0	2
7	del city restaurant	20	7	5	0	2	0
8	lawton restaurant	20	8	6	0	2	0
9	shawnee restaurant	20	8	5	0	3	0
10	muskogee restaurant	20	8	7	1	0	1
	Total	**200**	**75**	**55**	**1**	**19**	**4**

Rafting

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	tahlequah rafting	20	11	4	0	7	0
2	catoosa rafting	20	7	2	0	5	1
3	tulsa rafting	20	6	2	1	3	0
4	ardmore rafting	20	6	2	0	4	0
5	yukon rafting	20	3	2	1	0	0
6	tecumseh rafting	20	6	1	0	5	1
7	pontotoc rafting	20	6	1	0	5	0
8	hennessey rafting	20	4	1	0	3	2
9	miami rafting	20	5	3	0	2	0
10	broken bow rafting	20	5	0	0	5	0
	Total	**200**	**59**	**18**	**2**	**39**	**4**

Museum

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	tulsa museum	20	12	8	0	4	1
2	oklahoma city museum	20	12	9	1	2	0
3	norman museum	20	3	3	0	0	2
4	edmond museum	20	7	5	0	2	0
5	clinton museum	20	11	5	0	6	2
6	ardmore museum	20	10	9	0	1	0
7	broken arrow museum	20	11	8	2	1	0
8	bartlesville museum	20	10	4	0	6	1
9	lawton museum	20	10	6	0	4	0
10	guthrie museum	20	5	2	0	3	1
	Total	**200**	**91**	**59**	**3**	**29**	**7**

Safari

	Search String	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
1	tulsa safari	20	8	2	0	6	0
2	norman safari	20	2	0	0	2	1
3	oklahoma city safari	20	6	4	1	1	0
4	broken arrow safari	20	13	4	0	9	0
5	enid safari	20	5	1	0	4	2
6	muskogee safari	20	5	0	0	5	0
7	lawton safari	20	5	2	0	3	0
8	ardmore safari	20	4	3	0	1	0
9	edmond safari	20	6	0	0	6	1
10	tecumseh safari	20	2	0	0	2	0
	Total	**200**	**56**	**16**	**1**	**39**	**4**

Table 4.2: Result of the Experiment Using Yahoo

Keyword	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
Hotels	100	50	36	12	2	7
Restaurants	100	40	19	0	21	9
Museums	100	46	34	0	12	9
Ranches	100	25	7	0	18	1
Rafting	100	38	14	1	23	1
Safari	100	32	9	1	22	2
Total	600	231	119	14	98	29

Table 4.3: Summary for Google Experiment

Keyword	Number of URLs	Identified Site	Correct	Incorrect Location	Incorrect Type	Failed
Hotels	200	66	49	11	6	5
Restaurants	200	75	55	1	19	4
Museums	200	91	59	3	29	7
Ranches	200	77	21	1	55	2
Rafting	200	59	18	2	39	4
Safari	200	56	16	1	39	4
Total	1200	424	218	19	187	26

Table 4.4: Summary for Google Experiment

Chapter 5

Conclusion and Future Work

This thesis has presented an agent framework for automatic indexing of travel resources from the Internet. The framework is divided into two major components: indexing agent and search agent, and GIS agent.

The simplest component is the GIS agent. GIS agent perform the GIS reverse geocoding. GIS agent relies on `http://mapper.acme.com` to perform the reverse geocoding. The website accept `HTTP GET` method. Java provides `HttpURLConnection` class that perform all standard `HTTP` operations. The web site takes the name of the city or zip code as an input.

Indexing agent connect to the ebXML registry using Java API for XML Registry (JAXR) [16]. There are other ways to connect to the registry, and JAXR is probably not one of the best. JAXR is designed so that it will work with many XML registries, thus most of its methods are abstract. This creates problem when we are trying to access the specific functionality of the registry (e.g. ebXML registry is superset of XML registry). Indexing agents expects index token in the from of `Properties` object from the search agents. The `Properties` object from the search agent must contain `provider`, `location`, `type`, `name`, and `zip code`. All these components are required to insert the index token into the registry. Zip codes are required in order to

perform the GIS reverse geo-coding. All location information must be converted into the form of latitude and longitude before they are stored in the registry. Indexing agent will communicate with GIS agent to resolve this conversion.

Search agent responsible for finding travel related resources on the Internet. For this project, travel resources are categorized according to Yahoo! and the open directory project (DMOZ). First the search agent must find location information in the web page. If this information is not available, then it will be impossible for indexing agent to index this information. After finding the location information, the next step is to categorize the web page into its travel resource type. This procedure is describe in Chapter 4. This procedure does not always procedure 100% correct answer all the time. If the travel resource type can not be found or decided, then this URL will also be ignored. The last part is to decide the name for this particular web site. In this thesis, the name will be simply the domain name.

5.1 Future Works

There are many aspects in thesis that can be improved. For example, the GIS agent still relies on external GIS resolver (e.g. `http://mapper.acme.com`) when performing the reverse geo-coding. We can not know how long this site will be available. In the future, using JAX-RPC [15], the GIS query can be resolved by calling remote method in Microsoft TerraServer [52]. The Microsoft TerraServer provides complete GIS functionality (e.g. address to latitude and longitude, population, and etc.).

For classifying web pages into their travel resource type, another popular classifying engine can be used, such as the Support Vector Machine (SVM) [9]. SVM has

been renounced recently as the most effective classifying engine. Using SVM, the search agent will be able to provide better accuracy in matching web sites to their travel resource types.

Another improvement can be made in the area of recognizing the content of graphics files (e.g. `JPEG`, `GIF`, or `SWF`). Although this topic is still heavily under research (Optical Character Recognition → OCR), it can be incorporated in the future for parsing web page that contains graphics files. This will become very important because more and more web pages are incorporating graphics and animations into their web sites.

Currently the search agent is also designed to recognize only travel resources located in US (both the zip code and the ebXML `location` component support only US location). In future, search agents can be developed to recognize location information outside US.

References

[1] Witold Abramowicz, Pawel J. Kalczynski, and Krzysztof Wecel. *Filtering the Web to Feed Data Warehouses*. Springer-Verlag New York, Inc., 2002.

[2] Rafal Angryk, Violetta Galant, Minor Gordon, and Marcin Paprzycki. Travel support system – an 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, 2001.

[3] Josuis Arjona, Rafael Corchuelo, and Miguel Toro. Automatic extraction of information from the web.

[4] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[5] DCE 128 bit Universal Unique Identifier. `http://www.opengroup.org`.

[6] Brian E. Brewington and George Cybenko. How dynamic is the Web? *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):257–276, 2000.

[7] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1623–1640, 1999.

[8] Liren Chen and Katia Sycara. Webmate: a personal agent for browsing and searching. In *Proceedings of the second international conference on Autonomous agents*, pages 132–139. ACM Press, 1998.

[9] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, March 2000.

[10] Robert B. Doorenbos, Oren Etzioni, and Daniel S. Weld. A scalable comparison-shopping agent for the world-wide web. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 39–48, Marina del Rey, CA, USA, 1997. ACM Press.

[11] Oren Etzioni and Daniel S. Weld. Intelligent agents on the internet: Fact, fiction, and forecast. *IEEE Expert*, 10(3):44–49, 1995.

[12] Simple HTML Ontology Extensions. http://www.cs.umd.edu/projects/plus/SHOE/.

[13] Java Agent for Development (JADE) Framework. http://jade.tilab.com.

[14] Foundation for Intelligent Physical Agent (FIPA). http://www.fipa.org.

[15] Java API for XMl-Based RPC (JAX-RPC). http://java.sun.com/xml/jaxrpc/index.html.

[16] Java API for XML Registry. http://java.sun.com/xml/jaxr/index.html.

[17] William B. Frakes and Ricardo A. Baeza-Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.

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

[19] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96)*, volume 1193, Berlin, Germany, 1996. Springer-Verlag.

[20] V. Galant and M. Paprzycki. Information personalization in an internet based travel support system. In *Proceedings of the BIS Conference*, pages 191–202, Poznań, Poland, April 2002.

[21] J. Hendler. Agents and semantic web. In *IEEE Intelligent Systems Journal*, pages 30–37. IEEE, 2001.

[22] HTMLParser. http://htmlparser.sourceforge.net.

[23] Nicholas R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

[24] Nicholas R. Jennings and Michael J. Wooldridge. *Agent technology: foundations, applications, and markets*. Springer-Verlag New York, Inc., 1998.

[25] Pawel Jan Kalczyński. *Software agents to filter business information from the internet to the data warehouse*. PhD thesis, University of Poznan, Poznan, Poland, 2001.

[26] Pawel Jan Kalczyński, Marcin Paprzycki, Iwona Fiedorowicz, Witold Abramowicz, and Maria Cobb. Personalized traveler information system. In *Proceedings of the* 5^{th} *International Conference Human-Computer Interaction*, pages 445–456, Gdansk, Poland, 2001. Akwila Press.

[27] Craig A. Knoblock and José-Luis Ambite. Agents for information gathering. In Jeffrey M. Bradshaw, editor, *Software Agents*, pages 347–374. AAAI Press / The MIT Press, 1997.

[28] Craig A. Knoblock, Yigal Arens, and Chun-Nan Hsu. Cooperating agents for information retrieval. In *Proceedings of the Second International Conference on Cooperative Information Systems*, Toronto, Ontario, Canada, 1994. University of Toronto Press.

[29] Julian Kupiec, Jan O. Pedersen, and Francine Chen. A trainable document summarizer. In *Research and Development in Information Retrieval*, pages 68–73, 1995.

[30] DARPA Agent Markup Language. `http://www.daml.org`.

[31] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.

[32] Stop Words List. `http://www.onjava.com/onjava/2003/01/15/examples/EnglishStopWords.txt`.

[33] Jakarta Lucene. `http://jakarta.apache.org/lucene/docs/index.html`.

[34] Extraction Using Machine. Text summarization by sentence segment.

[35] Filippo Menczer. Links tell us about lexical and semantic web content. arXiv:cs.IR/0108004.

[36] Filippo Menczer, Gautam Pant, Padmini Srinivasan, and Miguel E. Ruiz. Evaluating topic-driven web crawlers. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 241–249. ACM Press, 2001.

[37] Nikola Mitrović and Eduardo Mena. Adaptive user interface for mobile devices. In *9th International Workshop on Design, Specification and Verification of Interactive Systems (DSVIS 2002)*. Springer Verlag Lecture Notes in Computer Science LNCS, June 2002.

[38] Martin E. Müller. An intelligent multi-agent architecture for information retrieval from the internet.

[39] Brian H. Murray and Alvin Moore. Sizing the internet. White paper, Cyveillance, July 2000.

[40] Divine T. Ndumu, Jaron C. Collis, and Hyacinth S. Nwana. Towards desktop personal travel agents. *BT Technological Journal*, 16(3):69–78, 1998.

[41] Java Web Services Developer Pack. `http://java.sun.com/webservices/downloads/webservicespack.html`.

[42] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[43] G. Pant and F. Menczer. Myspiders: Evolve your own intelligent web crawlers, 2002.

[44] M. Paprzycki, R. Angryk, K. Kolodziej, I. Fiedorowicz, M. Cobb, D. Ali, and S. Rahimi. Development of a travel support system based on intelligent agent technology. In S. Niwiński, editor, *Proceedings of PIONIER Conference*, pages 243–255, Poznań, Poland, 2001. Technical University of Poznań Press.

[45] M. Paprzycki, M. Gordon, and A. Gilbert. Knowledge representation in the agent-based travel support system. In T. Yakhno, editor, *Advances in Information System*, pages 232–241. Springer-Verlag, Berlin, 2002.

[46] M. Paprzycki, M. Gordon, P. Harrington, A. Nauli, S. Williams, and J. Wright. Using software agents to index data for an e-travel system. In *The* 7^{th} *World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, Florida, 2003.

[47] Martin Porter. Porter stemming algorithm. `http://www.tartarus.org/~martin/index.html`.

[48] S. Poslad, H. Laamanen, R. Malaka, A. Nick, and P. Buckle. Crumpet: Creation of user-friendly mobile services personalised for tourism. In *Proceedings: 3G 2001 – Second International Conference on 3G Mobile Communication Technologies*, London, UK, March 2001.

[49] The Semantic Web Project. `http://www.semanticweb.org`.

[50] OASIS/ebXML Registry. `http://www.oasis-open.org/`.

[51] Steffen Staab and Hannes Werthner. Intelligent systems for tourism. *IEEE Intelligent System*, 17(6):53–66, November/December 2002.

[52] Microsoft TerraServer. `http://terraserver.microsoft.com`.

[53] Jimmy Wright, Steve Williams, Marcin Paprzycki, and Patrick Harrington. Using ebxml registry/repository to manage information in an internet travel support system. In *Proceedings of the* 6^{th} *BIS Conference*. To appear, June 2003.

Appendix A

The Indexing Agent

A.1 Indexer

```
//loading ebXML related library
import javax.xml.registry.*;
import javax.xml.registry.infomodel.*;

//loading java related library
import java.util.*;
import java.util.logging.*;

public class Indexer {

  Logger logMeBaby = Logger.getLogger("Indexer");

  public static String TYPE =
    "/urn:uuid:00000000-0000-0000-0000-000000000000/type";
  public static String PROVIDER =
    "/urn:uuid:00000000-0000-0000-0000-000000000000/provider";
  public static String LOCATION =
    "/urn:uuid:00000000-0000-0000-0000-000000000000/location";

  BusinessQueryManager bqm;
  BusinessLifeCycleManager blcm;
  DeclarativeQueryManager dqm;
  Connection connection;
  RegistryService rs;
  ConnectionFactory cFact;

  public Indexer () {

    //Making the connection to the ebXMl registry
    Properties props = new Properties();
    props.setProperty("javax.xml.registry.queryManagerURL",
      "http://xamot:8090/ebxmlrr/registry/soap");
      //no need to set the lifeCycleManagerURL because the
    //default will be the same as the queryManagerURL

    try {
      cFact = ConnectionFactory.newInstance();
      cFact.setProperties(props);

      connection = cFact.createConnection();
      rs = connection.getRegistryService();
      bqm = rs.getBusinessQueryManager();
      blcm = rs.getBusinessLifeCycleManager();
      dqm = rs.getDeclarativeQueryManager();
    }
    catch (Exception e) {
      logMeBaby.warning("Failed to established connection to ebXML");
    }
  }
```

```
/**
 * This method insert index token into the registry
 * but before it insert, it checks for any duplicates
 * because it is undesireable to have duplicates in
 * the registry
 *
 * @param token Properties object that contains index
 *              token
 */

public boolean insert(Properties token) {

  String provider = token.getProperty("provider");
  String name = token.getProperty("name");
  String type = token.getProperty("type");
  String zip = token.getProperty("zip");
  String loc = token.getProperty("location");

  boolean same = false, success = false;

  try {
    //find the matching type based on name component
    //can't use findConceptsByPath because it only
    //returns one concept
    Query query = dqm.createQuery(Query.QUERY_TYPE_SQL,
      "SELECT * FROM ClassificationNode " +
      "WHERE path LIKE '" +
        TYPE + "/%" + type + "%" + name + "%'");

    //the results is any concept that belong to
    //specified type and name
    BulkResponse br = dqm.executeQuery(query);
    Iterator iter = br.getCollection().iterator();

    while (iter.hasNext()) {
      Concept t = (Concept)iter.next();
      //find corresponding component easily
      //because they are bind by content of
      //value attribute
      String value = t.getValue();
      String city = loc.substring(0,loc.indexOf(","));
      String state = loc.substring(
        loc.indexOf(",")+1,loc.length());

      //here we can use findConceptByPath()
      //because it should
      //return only one result
      Concept l = bqm.findConceptByPath(
        LOCATION+"/%"+state+"/"+city+"/"+value);

      //we have found the same type (Hotels) and
      //the same location (Las Vegas, NV) and
      //same name (Bellagio), what does it tell us ???
      if (l != null) {
        logMeBaby.info("Site is already Indexed");
        same = true; //the site is already indexed

        //checking the provider
        //if the provider is not the same, we might
        //want to add this new provider to the site
        Concept p = bqm.findConceptByPath(
          PROVIDER+"/"+value);
        Iterator el = p.getExternalLinks().iterator();
        boolean match = false;
        while (el.hasNext()) {
          ExternalLink elink = (ExternalLink) el.next();
          if (elink.getExternalURI().equalsIgnoreCase(
```

```
          provider)) {
          match = true; //we found an equivalent URI
          logMeBaby.info("Found Matching Provider");
          break; //get out of this loop
        }
      }

      //if we can't any matching provider URI
      if (!match) {
        //create new external link
        ExternalLink newEl =
          blcm.createExternalLink(provider,
            blcm.createInternationalString(
              "Provider for " + name));
        newEl.setName(blcm.createInternationalString(
          "Content provider for " + name));

        //create new association
        //(between this external link and
        //the provider node
        Association ass = blcm.createAssociation(p,
          bqm.findConceptByPath("%ExternallyLink%"));
        ass.setName(blcm.createInternationalString(
          "Association for " + name));
        ass.setDescription(
          blcm.createInternationalString(
            "Association for " + name));

        newEl.addAssociation(ass);
        Collection coll = new ArrayList();
        coll.add(newEl); coll.add(ass);
        blcm.saveObjects(coll);
      }
      //no need to check other nodes
      //because we found match
      break;
    }
  }

//site has not been indexed
  if (!same) {
    logMeBaby.info("New site will be created");

    //find the provider parent
    Concept p = bqm.findConceptByPath(PROVIDER);

    //create new concept that point to provider
    Concept newP = blcm.createConcept(p,name,name);

    //this is the utrc that will be used for binding
    //three components together as a site
    Key utcr = newP.getKey();

    //now use this key with the name of the site
    newP.setValue(utcr.getId()+":"+name);

    newP.setDescription(blcm.createInternationalString(
      name+" "+type+" "+loc));

    //now creating external link
    //and of course, it association
    ExternalLink el = blcm.createExternalLink(provider,
      blcm.createInternationalString(
        "Provider for " + name));
    el.setName(blcm.createInternationalString(
      "Content Provider for " + name));
    Association ass = blcm.createAssociation(newP,
      bqm.findConceptByPath("%ExternallyLink%"));
```

```
    ass.setName(blcm.createInternationalString(
      "Association for " + name));
    ass.setDescription(blcm.createInternationalString(
      "Association for " + name));
    el.addAssociation(ass);

    //now creating type component

    //finding correct place to put this
    //new type component
    Concept t = bqm.findConceptByPath(TYPE+"/%"+type);

    //creating new type component
    Concept newT = blcm.createConcept(t,name,
      utcr.getId()+":"+name);

    //set the description (can't be null)
    newT.setDescription(blcm.createInternationalString(
      name+" "+type+" "+loc));

    //now creating location component
    String city = loc.substring(0,loc.indexOf(","));
    String state =
      loc.substring(loc.indexOf(",")+1,loc.length());

    //finding parent for this location component
    Concept l =
      bqm.findConceptByPath(
        LOCATION+"/%"+state+"/"+city);

    //creating new location component
    Concept newL =
      blcm.createConcept(l,name,utcr.getId()+":"+name);

    //setting the description
    newL.setDescription(blcm.createInternationalString(
      name+" "+type+" "+loc));

    //committing the changes to the registry
    Collection save = new ArrayList();
    save.add(newL); //adding location
    save.add(newP); //adding provider
    save.add(newT); //adding type
    save.add(ass); //adding association
    save.add(el); //adding external link

    //checking if GIS slot is available
    if (token.containsKey("gis")) {
      String gis = token.getProperty("gis");
      Slot slot = blcm.createSlot("Coordinate",
        token.getProperty("gis"),"GIS");
      //add this slot to location component
      newL.addSlot(slot);
      //adding slot component
      save.add(slot);
    }

    blcm.saveObjects(save);
    success = true;
    //close this collection
    connection.close();
    //run a garbage collector here??
    System.gc();
  }
}
catch (Exception e) {
  logMeBaby.warning("Insert operation to ebXML failed");
}
```

```
    return success;
  }
}
```

A.2 Indexing Agent

```
//loading jade related library
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;

//loading ebXML related library
import javax.xml.registry.*;
import javax.xml.registry.infomodel.*;

//loading java related library
import java.util.*;
import java.util.logging.*;

public class IndexAgent extends Agent {

  Logger logMeBaby = Logger.getLogger(getName());

  /**
   * <b>WaitForIndexToken</b> class defines the
   * behaviour of IndexAgent. This behavior
   * implements SimpleBehaviour.
   */

  class WaitForIndexToken extends SimpleBehaviour {

    private boolean finish = false;

    public WaitForIndexToken(Agent a) {
      super(a);
    }

    /**
     * Defines the action that will be performed by
     * IndexAgent when message that match the
     * INFORM performative arrives. It extracts the
     * Properties object from the ACL message and
     * call the Indexer to index the token
     *
     * @see Indexer
     */

    public void action() {

      //wait for an ACL message to arrive
      ACLMessage msg = blockingReceive(MessageTemplate.MatchPerformative(
        ACLMessage.INFORM));

      if (msg != null) {
        try {
          Properties token = (Properties) msg.getContentObject();

          //trying to resolve GIS reverse geo-coding
          //first need to find GIS Agent
          AID gisAgent = findGISAgent();
          if (gisAgent != null) {
            ACLMessage toGIS = new ACLMessage(ACLMessage.QUERY_REF);
            toGIS.addReceiver(gisAgent);
            toGIS.setContent(token.getProperty("zip"));
```

```
          send(toGIS);

          //waiting for 20 seconds, so that if GIS is unresponsive
          //then index agent will not block forever
          ACLMessage fromGIS = blockingReceive(
            MessageTemplate.MatchPerformative(
              ACLMessage.INFORM_REF),20000);
          if (fromGIS != null) {
            if (!fromGIS.getContent().equals("fail"))
              token.setProperty("gis", fromGIS.getContent());
          }
        }

        //the cool thing here is that, any indexer can be
        //passed to IndexAgent as an argument when the
        //agent is launch so that, if there's any changes
        //in the registry, the new one can be loaded
        //dynamically....I don't use that method here
        //because so we only have one ebXML registry
        Indexer store = new Indexer();
        boolean success = store.insert(token);
        if (success) logMeBaby.info("Insert successful");
        else logMeBaby.warning("Insert failed");
        System.gc();
      }
      catch (Exception e) {
        logMeBaby.warning("Failed to retrieve message content");
      }
    }
  }

  public boolean done() {
    return finish;
  }
}


/**
 * This method queries the DF for GIS Agent.
 * With this method, the name of GIS Agent does not
 * need to be known at first. Also there can be
 * multiple GIS Agent serving several Clients
 */

private AID findGISAgent() {
  AID gis = null;
  DFAgentDescription dfd = new DFAgentDescription();
  ServiceDescription sd = new ServiceDescription();
  sd.setType("GISAgent");
  dfd.addServices(sd);
  try {
    while (true) {
      SearchConstraints c = new SearchConstraints();
      c.setMaxDepth(new Long(3));
      DFAgentDescription[] result =
        DFService.search(this,dfd,c);
      if ((result != null) && (result.length > 0)) {
        dfd = result[0];
        gis = dfd.getName();
        logMeBaby.info("Found GIS Agent");
        break;
      }
    }
  }
  catch (FIPAException e) {
    logMeBaby.info("Cannot Find GIS Agent");
  }
  return gis;
```

```
  }

  /**
   * This method sorts of setting up all the required
   * action before running the agent. In this case,
   * this method add the behavior to the agent and
   * register this agent with the DF
   */

  protected void setup() {

    try {

      //add the behavior as agent behavior
      WaitForIndexToken behave = new WaitForIndexToken(this);
      addBehaviour(behave);

      //register the agent with the Directory Facilitator (DF)
      //so the search agents can find indexing agent easily
      //by querying the DF for agents that
      //provide indexing service
      DFAgentDescription dfd = new DFAgentDescription();
      ServiceDescription sd = new ServiceDescription();
      //this agent is of type indexing agent
      sd.setType("IndexingAgent");
      //set the name of this agent
      sd.setName(getName());
      //set the owner of the agent
      sd.setOwnership("Agentlab");
      dfd.addServices(sd);
      dfd.setName(getAID());

      //register this agent with the Directory Facilitator
      DFService.register(this,dfd);
    }
    catch (Exception e) {
      logMeBaby.warning("Problem Registering Agent with the DF");
      //cancel this agent
      doDelete();
    }
  }
}
```

Appendix B

The GIS Agent

```
//loading java related library
import java.net.*;
import java.io.*;
import java.util.*;
import java.util.regex.*;
import java.util.logging.*;

//loading jade related library
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;

public class GISAgent extends Agent {

  Logger logMeBaby = Logger.getLogger(getName());
  /**
   * <b>WaitAndReplyWithGISInfo is class that defines
   * the behaviour of this agent. It implements
   * SimpleBehaviour which models simple atomic behaviors.
   */

  class WaitAndReplyWithGISInfo extends SimpleBehaviour {

    private boolean finish = false;

    public WaitAndReplyWithGISInfo(Agent a) {
      super(a);
    }

    /**
     * Defines the action that the agent will perform
     * for this behavior. Since this behavior implements
     * SimpleBehaviour, it will be repeated forever.
     * It waits for an ACL Message with QUERY_REF
     * performatives, all other performatives will
     * be discarded
     */

    public void action() {

      ACLMessage msg = blockingReceive(
        MessageTemplate.MatchPerformative(ACLMessage.QUERY_REF));

      if (msg != null) {
        String zip = msg.getContent();
        try {
          URL url = new URL(
            "http://mapper.acme.com/find.cgi?zip="+zip);
          HttpURLConnection conn = (HttpURLConnection)
            url.openConnection();
```

```
        Pattern p = Pattern.compile(
          "/\\?lat=(-?\\d+.\\d+)\\&long=(-?\\d+.\\d+).*");
        Matcher m = p.matcher(conn.getHeaderField(2));
        if (m.matches()) {
          //creating reply message
          ACLMessage reply = msg.createReply();
          //set its performative
          reply.setPerformative(ACLMessage.INFORM_REF);
          //set its content with the value of
          //latitude and longitude
          reply.setContent("Lat="+m.group(1)+" "+
            "Long="+m.group(2));
          //send the message
          send(reply);
        }
        //if the zip code can not be found
        else {
          //create reply message
          ACLMessage reply = msg.createReply();
          //set its performative
          reply.setPerformative(ACLMessage.INFORM_REF);
          //set its content
          reply.setContent("fail");
          send(reply);
        }
      }
      catch (Exception e) {
        logMeBaby.warning("Error Establishing Connection");
        e.printStackTrace();
      }
    }
  }

  public boolean done() {
    return finish;
  }
}

/**
 * This method is called when agent is brought to live
 * the first time. Sort of setting up different kind
 * of stuff before the run() method of agent is called.
 * Here we add the SimpleBehaviour to agent's behaviors
 * and register this agent with the DF so that it
 * can advertise its service.
 *
 */

protected void setup() {

  try {
    //add the behavior as agent behavior
    WaitAndReplyWithGISInfo gis = new
      WaitAndReplyWithGISInfo(this);
    addBehaviour(gis);

    //register the agent with the
    //Directory Facilitator so the indexing
    //agent can find GIS agent easily by
    //querying the DF for agent that provide
    //GIS service
    DFAgentDescription dfd =
      new DFAgentDescription();
    ServiceDescription sd =
      new ServiceDescription();
    //this agent is of type GIS Agent
    sd.setType("GISAgent");
    //set the name of this agent
```

```
      sd.setName(getName());
      //set the owner of the agent
      sd.setOwnership("Agentlab");
      dfd.addServices(sd);
      dfd.setName(getAID());

      //registering the agent with DF
      DFService.register(this,dfd);
    }
    catch (Exception e) {
      logMeBaby.warning("Problem Registering Agent with the DF");
      //cancel this agent
      doDelete();
    }
  }
}
```

Appendix C

Travel Resource Keywords

C.1 Filtering HTML Page

`MyAnalyzer` is `Analyzer` class that is used to filter HTMLPage. This class is used by `IndexWriter` (see Section C.3) when creating index terms. The SAX parser class is in Section C.2.

```
/**
 * <b>MyAnalyzer</b> class is used by lucene for
 * tokenizing words. This class implements Porter
 * Stemming algorithm (to return words into its
 * basic forms> and Stop word filter to filter
 * out meaningless word.
 *
 * Stop words are stored in stopwords.xml file
 * and are loaded using MySAXParser class
 *
 * @author Andy Nauli
 * @since 1.4.1
 * @version %I%, %G%
 * @see MySAXParsser
 */

import org.apache.lucene.analysis.standard.*;
import org.apache.lucene.analysis.*;
import java.io.*;

public class MyAnalyzer extends Analyzer {

  public final TokenStream tokenStream(String fieldName, Reader reader) {

    MySAXParser parse = new MySAXParser(new File("stopwords.xml"));
    String[] SMART_STOP_WORDS = parse.getKeywords();

    return new PorterStemFilter(
      new StopFilter(new LowerCaseTokenizer(reader), SMART_STOP_WORDS));
  }
}
```

C.2 MySAXParser

```
import org.xml.sax.helpers.*;
import org.xml.sax.*;
import java.util.*;
import javax.xml.parsers.*;
import java.io.*;

import java.util.logging.*;

public class MySAXParser extends DefaultHandler {

  Logger logMeBaby = Logger.getLogger("MySAXParser");

  //this variable holds the result
  //of the parsing
  StringBuffer sb = new StringBuffer();

  /**
   * Default constructor for MySAXParser class
   * This constructor accept File object as
   * parameter. This File object is the name of
   * the file that will be parsed
   *
   * @param file the xml file to be parsed
   */
  public MySAXParser (File file) {
    try {
      //these are standard ways to invoke
      //SAXParser
      SAXParserFactory factory =
        SAXParserFactory.newInstance();
      SAXParser parser = factory.newSAXParser();
      parser.parse(file, this);
    }
    catch (Exception e) {
      //there's a problem reading and/or parsing
      //the XML file
      logMeBaby.warning("Problem Parsing XML file");
    }
  }

  /**
   * This method declaration overwrites the
   * abstract startElement() method of the
   * DefaultHandler. In here we gather the
   * attributes value and append it to
   * the StringBuffer
   *
   */
  public void startElement (String namespaceURI,
        String sName,
        String qName,
        Attributes attrs) throws SAXException {
    if (attrs != null) {
      //get all the value of the attributes
      //we split this keywords by using
      //semicolon
      for (int i=0;i<attrs.getLength();i++) {
        sb.append(attrs.getValue(i)+";");
      }
    }
  }

  /**
   * This method returns the keywords/stopwords
   * in the form of String[]
   */
  public String[] getKeywords() {
```

```
    String[] keywords = null;
    //perform conversion from string to
    //an array of string
    keywords = sb.toString().split(";");
    return keywords;
  }

}
```

C.3 Creating Index Terms

```
import org.htmlparser.beans.StringBean;

import org.apache.lucene.index.*;
import org.apache.lucene.document.*;
import org.apache.lucene.analysis.standard.*;
import org.apache.lucene.analysis.*;
import org.apache.lucene.store.*;

public class Keyword {
  public static void main(String[] args) throws Exception {
    StringBean sb = new StringBean();
    sb.setLinks(false);
    sb.setReplaceNonBreakingSpaces(true);
    sb.setCollapse(true);
    sb.setURL("http://www.someurl.com");

    RAMDirectory ram = new RAMDirectory();
    IndexWriter writer = new IndexWriter(ram,
      new MyAnalyzer(), true);
    Document document = new Document();
    document.add(Field.Text("body", sb.getStrings()));
    writer.close();

    IndexReader reader = IndexReader.open(ram);
    TermEnum te = reader.terms();
    int min = 0;
    PQ pq = new PQ(20);

    while (te.next()) {
      TermDocs td = reader.termDocs(te.term());
      while (td.next()) {
        if (td.freq() > min) {
          pq.put(new tfidf(te.term(),td.freq()));
          if (pq.size() > 20) {
            pq.pop();
            min = ((tfidf)pq.top()).freq;
          }
        }
      }
    }

    while (pq.size() != 0) {
      tfidf a = (tfidf) pq.pop();
      System.out.println(a.term.text()+":"+a.freq);
    }
    reader.close();
    }
  }
}
final class tfidf {
  Term term;
```

```
  int freq;

  public tfidf(Term term, int freq) {
    this.term = term;
    this.freq = freq;
  }
}

final class PQ extends PriorityQueue {
  public PQ(int size) {
    initialize(size);
  }

  protected final boolean lessThan(Object a, Object b) {
    tfidf one = (tfidf) a;
    tfidf two = (tfidf) b;
    return one.freq < two.freq;
  }
}
```

C.4 The Complete Keywords

```
<?xml version='1.0' encoding='utf-8'?>
<!-- This XML file contains keywords for
each travel resources-->
<keywords>
  <type name="currency exchange"/>
  <type name="bars"/>
  <type name="restaurants"/>
  <type name="beds and breakfast"/>
  <type name="casinos"/>
  <type name="castles"/>
  <type name="hostels"/>
  <type name="hotels"/>
  <type name="motels"/>
  <type name="ranches"/>
  <type name="resorts"/>
  <type name="theme lodges"/>
  <type name="timeshares"/>
  <type name="base jumping"/>
  <type name="climbing"/>
  <type name="galleries"/>
  <type name="golf"/>
  <type name="historical sites"/>
  <type name="monuments"/>
  <type name="museums"/>
  <type name="nightlife"/>
  <type name="parasailing"/>
  <type name="personal watercraft"/>
  <type name="rafting"/>
  <type name="scuba"/>
  <type name="safari"/>
  <type name="sky diving"/>
  <type name="snow boarding"/>
  <type name="snow skiing"/>
  <type name="ultra light"/>
  <type name="water skiing"/>
  <type name="charter airlines"/>
  <type name="commercial airlines"/>
  <type name="helicopter"/>
  <type name="hot air ballon"/>
  <type name="zeppelin"/>
```

```
  <type name="bus"/>
  <type name="camping"/>
  <type name="rental"/>
  <type name="touring"/>
  <type name="bicycle"/>
  <type name="motorcycle"/>
  <type name="bus"/>
  <type name="ferry"/>
  <type name="subway"/>
  <type name="taxi"/>
  <type name="trolley"/>
  <type name="backpacking"/>
  <type name="hitchhiking"/>
  <type name="rail"/>
  <type name="space"/>
  <type name="boating"/>
  <type name="cruises"/>
  <type name="ocean liner"/>
  <type name="sailing"/>
  <type name="inn"/>
  <type name="suites"/>
</keywords>
```

Appendix D

Search Agent

D.1 CheckZip

```
import org.apache.lucene.index.*;
import org.apache.lucene.document.*;
import org.apache.lucene.analysis.standard.*;
import org.apache.lucene.store.*;
import org.apache.lucene.search.*;
import org.apache.lucene.queryParser.*;

import java.util.*;
import java.util.regex.*;
import java.net.*;
import java.io.*;
import java.util.logging.*;

import org.htmlparser.beans.StringBean;

public class CheckZip {


  public static String check(String code, RAMDirectory ram) {

    Logger logMeBaby = Logger.getLogger("CheckZip");

    String result="";

    try {
      URL url = new URL("http://www.usps.com/zip4/zip_response.jsp");
      HttpURLConnection conn = (HttpURLConnection) url.openConnection();
      //this is a post method
      conn.setRequestMethod("POST");

      //usual stuff for post method
      conn.setDoOutput (true);
      conn.setDoInput (true);
      conn.setUseCaches (false);

      //connection header
      conn.setRequestProperty ("Accept-Charset", "*");
      conn.setRequestProperty (
        "Referer", "http://www.usps.com/zip4/citytown.htm");
      conn.setRequestProperty ("User-Agent", "Zip.java/1.0");

      StringBuffer buffer = new StringBuffer (1024);
      // 'input' fields separated by ampersands (&)
      buffer.append ("zipcode=");
      buffer.append (code);

      PrintWriter out = new PrintWriter(conn.getOutputStream());
      out.print(buffer);
      out.close();
```

```
      //getting the result back
      StringBean sb = new StringBean();
      sb.setLinks(false);
      sb.setReplaceNonBreakingSpaces(true);
      sb.setCollapse(true);
      sb.setConnection(conn);

      //parsing the result page
      Pattern p = Pattern.compile("([A-Z ]+[A-Z]) ([A-Z]{2}) ACCEPTABLE");
      Matcher m = p.matcher(sb.getStrings());

      //now checking the validity of the zip code
      //some zip code returns more than one city, need to check all of that
      float score = 0f;

      while (m.find()) {
        String city = m.group(1).trim();
        String state = m.group(2).trim();

        //looking for the index
        Searcher searcher = new IndexSearcher(ram);
        Query query = QueryParser.parse(city, "body",new StandardAnalyzer());
        Hits hits = searcher.search(query);
        if (hits.length() != 0) {
          //retain the one with the highest score
          if (score < hits.score(0)) {
            score = hits.score(0);
            result = city + "," + state;
          }
        }
        searcher.close();
      }
    }
    catch (Exception e) {
      logMeBaby.warning("Failed to check zip code");
    }

    //either null or contain location information
    if (result.length()>0) return result;
    else return null;
  }
}
```

D.2 GuessType

```
//loading lucene related library
import org.apache.lucene.index.*;
import org.apache.lucene.document.*;
import org.apache.lucene.analysis.standard.*;
import org.apache.lucene.store.*;
import org.apache.lucene.search.*;
import org.apache.lucene.queryParser.*;

//loading java library
import java.util.*;
import java.util.regex.*;
import java.net.*;
import java.io.*;
import java.util.logging.*;

//loading HTMLParser library
import org.htmlparser.beans.StringBean;
import org.htmlparser.*;
import org.htmlparser.tags.*;
import org.htmlparser.util.*;
```

```
public class GuessType {

  /**
   * This method resolve the current url
   * into its category type e.g. hotel,
   * restaurant, motel, etc.
   *
   * The keywords are loaded from the
   * keywords.xml file using MySAXParser
   * class
   *
   * @param url the url to be resolved
   * @see MySAXParser
   */

  public static String guess(String url) {

    String result = "";

    Logger logMeBaby = Logger.getLogger("GuessTypeName");

    //loading keywords
    MySAXParser parse = new MySAXParser(new File("keywords.xml"));
    String[] type = parse.getKeywords();

    String metaText="",title="";

    try {
      //create new parser without any parser feedback (null)
      Parser parser = new Parser(url, null);
      parser.registerScanners();
      NodeIterator iter = parser.elements();
      while (iter.hasMoreNodes()) {
        Node node = (Node) iter.nextNode();
        if (node instanceof TitleTag) {
          TitleTag titletag = (TitleTag) node;
          title = titletag.getTitle();
        }
        if (node instanceof MetaTag) {
          MetaTag metatag = (MetaTag) node;
          //we don't want to examine Meta Tag with HTTP-EQUIV
          if (metatag.getMetaTagName()!=null) {
            metaText += metatag.getMetaContent();
          }
        }
      }

      RAMDirectory ram = new RAMDirectory();
      IndexWriter writer = new IndexWriter(ram, new MyAnalyzer(), true);
      Document document = new Document();
      document.add(Field.UnStored("meta",metaText));
      document.add(Field.UnStored("title",title));

      StringBean sb = new StringBean();
      sb.setLinks(false);
      sb.setReplaceNonBreakingSpaces(true);
      sb.setCollapse(true);
      sb.setURL(url);

      document.add(Field.UnStored("body",sb.getStrings()));

      writer.addDocument(document);
      writer.close();

      Searcher searcher = new IndexSearcher(ram);
      float score = 0f; int key=-1;
      for (int i=0;i<type.length;i++) {
        Query query = QueryParser.parse(type[i],"meta",new MyAnalyzer());
```

```
        Hits hits = searcher.search(query);
        if (hits.length() != 0) {
          if (score < hits.score(0)) {
            score = hits.score(0);
            key = i;
          }
        }
        query = QueryParser.parse(type[i],"title",new MyAnalyzer());
        hits = searcher.search(query);
        if (hits.length() !=0) {
          if (score < hits.score(0)) {
            score = hits.score(0);
            key = i;
          }
        }
        query = QueryParser.parse(type[i],"body",new MyAnalyzer());
        hits = searcher.search(query);
        if (hits.length() !=0) {
          if (score < hits.score(0)) {
            score = hits.score(0);
            key = i;
          }
        }
      }
      searcher.close();

      if (score > 0) {
        //if it's categorized as inn or suite then
        //it belongs to hotel
        if (key > 54) key = 7;
        //the type
        result = type[key];
      }

      ram.close();
    }
    catch (Exception e) {
      logMeBaby.warning("Failed to guess type");
    }

    //garbage collector
    System.gc();

    if (result.length() > 0) return result;
    else return null;
  }
}
```

D.3 DistributorAgent

```
/**
 * <b>DistributorAgent</b> class defines
 * type of search
 * agent that distributes url to
 * SiteAgent. SiteAgent reprent search
 * agent that responsible for one site
 *
 * For this moment only text/* search
 * agent will be provided.
 *
 * @author Andy Nauli
 * @since 1.4.1
 * @version %I%, %G%
 * @see SiteAgent
 */

//loading jade related library
```

```
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import jade.wrapper.PlatformController;
import jade.wrapper.ControllerException;

//loading html parser related library
import org.htmlparser.*;
import org.htmlparser.tags.*;
import org.htmlparser.util.*;

//loading java built-in library
import java.util.*;
import java.net.*;
import java.io.*;

//loading java logging package
//per Minor's request
import java.util.logging.*;

public class DistributorAgent extends Agent {

  Logger logMeBaby = Logger.getLogger(
    getName());

  /*
   * Here we define the behavior that this
   * agent will have. You can define more
   * than one behaviour and they can be
   * added to list of agent's behaviors.
   */
  class Distribute extends SimpleBehaviour {
    private boolean finish = false;

    public Distribute(Agent a) {
      super(a);
    }

    /*
     * This method defines the action
     * that an agent will perform
     */
    public void action() {

      ACLMessage msg = blockingReceive(
        MessageTemplate.MatchPerformative(ACLMessage.REQUEST));

      if (msg != null) {
        try {
          URL url = new URL(msg.getContent());
          URLConnection conn = url.openConnection();
          //checking content type
          if (conn.getContentType().matches("text/.*")) {
            //start crawling this URL
            crawls(conn.getURL().toString());
          }
          //in the future we can have another if here that
          //can handle content besides text and call its
          //crawl method
          else logMeBaby.info(
            conn.getContentType() + " not yet supported");
        }
        catch (MalformedURLException e) {
          logMeBaby.warning(
            "MalformedURLException " + msg.getContent());
        }
```

```
      catch (IOException e) {
        logMeBaby.warning(
          "IOException " + msg.getContent());
      }
      catch (Exception e) {
        logMeBaby.warning(
          "connection to " + msg.getContent() + " failed");
      }
    }
  }

  public boolean done() {
    return finish;
  }
}

/*
 * This method crawls and extract links from the
 * start URL. Each of these links will be assigned
 * to specialized agents (e.g. agents that handles
 * pdf, words, image, etc)
 */
private void crawls(String startURL) {

  Set urls = new HashSet();

  //add this first url
  urls.add(startURL);

  Node[] nodes = null;

  try {
    Parser parser = new Parser(startURL,null);
    parser.registerScanners();
    nodes = parser.extractAllNodesThatAre(LinkTag.class);
  }
  catch (Exception e) {
    logMeBaby.warning("Cannot parse the URL at " + startURL);
  }

  //if we were able to parse the start URL
  if (nodes != null) {
    for (int i = 0; i<nodes.length; i++) {
      LinkTag link = (LinkTag) nodes[i];
      if (isSameHost(link.getLink(),startURL))
        urls.add(link.getLink());
    }
  }

  try {
    PlatformController pc = getContainerController();
    Set[] setArray = new Set[1];
    setArray[0] = urls;
    pc.createNewAgent(startURL, "SiteAgent", setArray).start();
  }
  catch (Exception e) {
    logMeBaby.warning("Failed to crawl " + startURL);
  }
}

/*
 * This method returns true if two URLs are in
 * the same host, returns false otherwise
 *
 * @param a current url
 * @param b the original url
 */
```

```
  private boolean isSameHost(String a, String b) {

    //i prefer not to compare by ignore case because
    //sometimes unix doesn't work this way

    boolean same = false;

    //if it ends with "/" then it's a directory
    if (b.endsWith("/"))
      same = a.startsWith(b);
    //if it does not end with "/" then more stuff
    //need to be taken care of
    else {
      String host = b.substring(0,b.lastIndexOf("/"));
      if (!host.endsWith("/")) same = a.startsWith(host);
      else same = a.startsWith(b);
    }

    return same;
  }

  /*
   * This method is executed when the agent
   * is brought up to live the first time.
   * Here we add the designated behavior
   * and register the agent to the DF so
   * that later on we can find it if
   * we need it without even knowing
   * its name
   */
  protected void setup() {
    try {

      //add the behavior as agent behavior
      Distribute search = new Distribute(this);
      addBehaviour(search);

      //register the agent with the Directory Facilitator (DF)
      //so we can find it later easily
      //by querying the DF for agents that
      //provide searching service
      DFAgentDescription dfd = new DFAgentDescription();
      ServiceDescription sd = new ServiceDescription();
      //this agent is of type search agent
      sd.setType("DistributeAgent");
      //set the name of this agent
      sd.setName(getName());
      //set the owner of the agent
      sd.setOwnership("Agentlab");
      dfd.addServices(sd);
      dfd.setName(getAID());

      //register this agent with the Directory Facilitator
      DFService.register(this,dfd);
    }
    catch (FIPAException e) {
      logMeBaby.warning(
          "Problem Registering Agent with the DF");
      //cancel this agent
      doDelete();
    }
  }

}
```

D.4 SiteAgent

```
//loading jade related library
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import jade.wrapper.PlatformController;
import jade.wrapper.ControllerException;

//loading html parser related library
import org.htmlparser.*;
import org.htmlparser.tags.*;
import org.htmlparser.util.*;

//loading java built-in library
import java.util.*;
import java.net.*;
import java.io.*;

//loading java logging package
//per Minor's request
import java.util.logging.*;

public class SiteAgent extends Agent {

  Logger logMeBaby = Logger.getLogger(
    getName());

  /*
   * Here we define the behavior that this
   * agent will have. You can define more
   * than one behaviour and they can be
   * added to list of agent's behaviors.
   *
   * OneShotBehaviour is an atomic behavior
   * that will be executed only once. This
   * agent will die after it crawl the
   * given sites. It correspond to one site
   * agent for each site.
   */
  class Distribute extends OneShotBehaviour {

    public Distribute(Agent a) {
      super(a);
    }

    /*
     * This method defines the action
     * that an agent will perform
     */

    public void action() {
      Set urls = (Set) getArguments()[0];
      Iterator iter = urls.iterator();
      while (iter.hasNext()) {
        String link = (String) iter.next();
        try {
          URL url = new URL(link);
          URLConnection conn = url.openConnection();
          //if content is text, then launch search agent
          //that can parse text
          if (conn.getContentType().matches("text/.*")) {
            Random rand = new Random();
            PlatformController pc = getContainerController();
            String name = "Search" +
              Integer.toString(Math.abs(rand.nextInt()));
            String[] argument = new String[2];
            //1st argument is URL address
```

```
          argument[0] = conn.getURL().toString();
          //2nd argument is name
          argument[1] = getName();
          pc.createNewAgent(name, "SearchHTML", argument).start();

          //waiting to see if this generates a success or not
          ACLMessage msg = blockingReceive();

          //site complete, get out of here
          if (msg.getPerformative()==ACLMessage.CONFIRM) {
            //without creating new ACLMessage
            //we just forward this message to index
            //agent
            msg.setPerformative(ACLMessage.INFORM);
            msg.clearAllReceiver();
            AID index = findIndexAgent();
            if (index != null) {
              msg.addReceiver(index);
              send(msg);
              break;
            }
          }
        }
        //if content is pdf, then launch search agent
        //that can parse pdf, etc
        //if (conn.getContentType().matchess("application/pdf");
        else logMeBaby.info(
          conn.getContentType() + " is not supported yet");
      }
      catch (Exception e) {
        logMeBaby.warning("Ignoring the URL " + link);
      }
    }

    //kill this agent
    doDelete();
  }

}

/**
 * This method queries the DF for Index Agent.
 * With this method, the name of Index Agent does not
 * need to be known at first. Also there can be
 * multiple Index Agent serving several Clients
 */

private AID findIndexAgent() {
  AID index = null;
  DFAgentDescription dfd = new DFAgentDescription();
  ServiceDescription sd = new ServiceDescription();
  sd.setType("IndexingAgent");
  dfd.addServices(sd);
  try {
    while (true) {
      SearchConstraints c = new SearchConstraints();
      c.setMaxDepth(new Long(3));
      DFAgentDescription[] result =
        DFService.search(this,dfd,c);
      if ((result != null) && (result.length > 0)) {
        dfd = result[0];
        index = dfd.getName();
        logMeBaby.info("Found Indexing Agent");
        break;
      }
    }
  }
  catch (FIPAException e) {
```

```
      logMeBaby.info("Cannot Find Indexing Agent");
    }
    return index;
  }

  /*
   * This method is executed when the agent
   * is brought up to live the first time.
   * Here we add the designated behavior
   *
   * There's no need to register this agent
   * with the DF because it will die after
   * performing its duties.
   */
  protected void setup() {
    try {

      //add the behavior as agent behavior
      Distribute search = new Distribute(this);
      addBehaviour(search);
    }
    catch (Exception e) {
      logMeBaby.info(
          "Problem Starting Agent " + getName());
      //cancel this agent
      doDelete();
    }
  }

}
```

D.5 SearchHTML

```
//loading jade related library
import jade.core.*;
import jade.core.behaviours.*;
import jade.lang.acl.*;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;
import jade.wrapper.PlatformController;
import jade.wrapper.ControllerException;

//loading html parser related library
import org.htmlparser.*;
import org.htmlparser.tags.*;
import org.htmlparser.util.*;
import org.htmlparser.beans.StringBean;

//loading java built-in library
import java.util.*;
import java.net.*;
import java.io.*;
import java.util.regex.*;

//loading lucene library
import org.apache.lucene.index.*;
import org.apache.lucene.document.*;
import org.apache.lucene.analysis.standard.*;
import org.apache.lucene.store.*;
import org.apache.lucene.search.*;
import org.apache.lucene.queryParser.*;

//loading java logging package
//per Minor's request
import java.util.logging.*;

public class SearchHTML extends Agent {
```

```
//ahhh..the long awaited logging
//class is here. This is just basic
//logging which prints log to the
//console, if need this can be
//changed later to any destination
//e.g file (in plain or xml format)
Logger logMeBaby = Logger.getLogger(
  getName());

/*
 * Here we define the behavior that this
 * agent will have. You can define more
 * than one behaviour and they can be
 * added to list of agent's behaviors.
 *
 * This agent implements OneShotBehaviour
 * because we want it to die after performing
 * its duties.
 */
class SearchText extends OneShotBehaviour {

  public SearchText(Agent a) {
    super(a);
  }

  /*
   * This method defines the action
   * that an agent will perform
   *
   * The URL is in agent's argument
   * This argument was inserted when
   * the agent was created
   */
  public void action() {

    Properties token = new Properties();
    String[] argument = (String[]) getArguments();

    try {
      //stringbean is the best way to convert
      //html into text -> thanks HTMLParser for this
      StringBean sb = new StringBean();
      sb.setLinks(false);
      sb.setReplaceNonBreakingSpaces(true);
      sb.setCollapse(true);
      sb.setURL(argument[0]);

      //here comes lucene, we will tokenize the text from stringbean
      RAMDirectory dir = new RAMDirectory();
      IndexWriter writer = new IndexWriter(dir, new StandardAnalyzer(), true);
      Document doc = new Document();
      doc.add(Field.UnStored("body",sb.getStrings()));
      writer.addDocument(doc);
      writer.close();

      //now performing the search
      IndexReader reader = IndexReader.open(dir);
      TermEnum enum = reader.terms();
      while (enum.next()) {
        //possible zip code found
        //there could be more than one location
        if (enum.term().text().matches("\\d{5}(-\\d{4})?")) {
          String result = CheckZip.check(enum.term().text().substring(0,5),dir);
          if (result!=null) {
            token.setProperty("location",result.toLowerCase());
            token.setProperty("zip",enum.term().text().substring(0,5));
          }
```

```
        }
      }
    }
    catch (Exception e) {
      logMeBaby.warning("Failed to parse " + argument[0]);
    }

    //don't proceed if location information is not found
    if (!token.isEmpty()) {
      String type = GuessType.guess(argument[0]);
      if (type != null) {
        token.setProperty("type",type);
        try {
        URL url =  new URL(argument[0]);
        Pattern p = Pattern.compile(
          "[.]([a-z]+)[.]([a-z]+|[a-z]+[.][a-z]{2})$");
        Matcher m = p.matcher(url.getHost());
        if (m.find()) token.setProperty("name",m.group(1));
        }
        catch (Exception e) {
          logMeBaby.warning("Failed to name a site so default will be used");
          token.setProperty("name",argument[0]);
        }
        //setting provider component
        token.setProperty("provider",argument[0]);
      }
    }

    ACLMessage msg = new ACLMessage(ACLMessage.FAILURE);

    //if token has 5 key, then it's complete
    if (token.size()==5)
      //letting now the SiteAgent agent we succeed
      msg.setPerformative(ACLMessage.CONFIRM);

    //receiver is the site agent
    msg.addReceiver(new AID(argument[1], true));
    try {
      msg.setContentObject(token);
    }
    catch (Exception e) {
      logMeBaby.warning("Failed to add index token to message");
      msg.setPerformative(ACLMessage.FAILURE);
    }

    //send the message
    send(msg);

    doDelete();
  }
}

/*
 * This method is executed when the agent
 * is brought up to live the first time.
 * Here we add the designated behavior
 */
protected void setup() {
  try {
    //add the behavior as agent behavior
    SearchText search = new SearchText(this);
    addBehaviour(search);

    //this agent does not need to register
    //with the DF because it's not a
    //permanent agents, it dies when it
    //finishes its jobs
  }
```

```
    catch (Exception e) {
      logMeBaby.info(
          "Problem adding behaviour to agent");
      //cancel this agent
      doDelete();
    }
  }

}
```

D.6 Stop Words

```
<?xml version='1.0' encoding='UTF-8'?>
<!--XML file that contains the stopwords-->
<stopwords>
  <word name="a"/>
  <word name="able"/>
  <word name="about"/>
  <word name="above"/>
  <word name="according"/>
  <word name="accordingly"/>
  <word name="across"/>
  <word name="actually"/>
  <word name="after"/>
  <word name="afterwards"/>
  <word name="again"/>
  <word name="against"/>
  <word name="all"/>
  <word name="allow"/>
  <word name="allows"/>
  <word name="almost"/>
  <word name="alone"/>
  <word name="along"/>
  <word name="already"/>
  <word name="also"/>
  <word name="although"/>
  <word name="always"/>
  <word name="am"/>
  <word name="among"/>
  <word name="amongst"/>
  <word name="an"/>
  <word name="and"/>
  <word name="another"/>
  <word name="any"/>
  <word name="anybody"/>
  <word name="anyhow"/>
  <word name="anyone"/>
  <word name="anything"/>
  <word name="anyway"/>
  <word name="anyways"/>
  <word name="anywhere"/>
  <word name="apart"/>
  <word name="appear"/>
  <word name="appreciate"/>
  <word name="appropriate"/>
  <word name="are"/>
  <word name="around"/>
  <word name="as"/>
  <word name="aside"/>
  <word name="ask"/>
  <word name="asking"/>
  <word name="associated"/>
  <word name="at"/>
  <word name="available"/>
  <word name="away"/>
  <word name="awfully"/>
  <word name="b"/>
```

```
<word name="be"/>
<word name="became"/>
<word name="because"/>
<word name="become"/>
<word name="becomes"/>
<word name="becoming"/>
<word name="been"/>
<word name="before"/>
<word name="beforehand"/>
<word name="behind"/>
<word name="being"/>
<word name="believe"/>
<word name="below"/>
<word name="beside"/>
<word name="besides"/>
<word name="best"/>
<word name="better"/>
<word name="between"/>
<word name="beyond"/>
<word name="both"/>
<word name="brief"/>
<word name="but"/>
<word name="by"/>
<word name="c"/>
<word name="came"/>
<word name="can"/>
<word name="cannot"/>
<word name="cant"/>
<word name="cause"/>
<word name="causes"/>
<word name="certain"/>
<word name="certainly"/>
<word name="changes"/>
<word name="clearly"/>
<word name="co"/>
<word name="com"/>
<word name="come"/>
<word name="comes"/>
<word name="concerning"/>
<word name="consequently"/>
<word name="consider"/>
<word name="considering"/>
<word name="contain"/>
<word name="containing"/>
<word name="contains"/>
<word name="corresponding"/>
<word name="could"/>
<word name="course"/>
<word name="currently"/>
<word name="d"/>
<word name="definitely"/>
<word name="described"/>
<word name="despite"/>
<word name="did"/>
<word name="different"/>
<word name="do"/>
<word name="does"/>
<word name="doing"/>
<word name="done"/>
<word name="down"/>
<word name="downwards"/>
<word name="during"/>
<word name="e"/>
<word name="each"/>
<word name="edu"/>
<word name="eg"/>
<word name="eight"/>
<word name="either"/>
```

```
<word name="else"/>
<word name="elsewhere"/>
<word name="enough"/>
<word name="entirely"/>
<word name="especially"/>
<word name="et"/>
<word name="etc"/>
<word name="even"/>
<word name="ever"/>
<word name="every"/>
<word name="everybody"/>
<word name="everyone"/>
<word name="everything"/>
<word name="everywhere"/>
<word name="ex"/>
<word name="exactly"/>
<word name="example"/>
<word name="except"/>
<word name="f"/>
<word name="far"/>
<word name="few"/>
<word name="fifth"/>
<word name="first"/>
<word name="five"/>
<word name="followed"/>
<word name="following"/>
<word name="follows"/>
<word name="for"/>
<word name="former"/>
<word name="formerly"/>
<word name="forth"/>
<word name="four"/>
<word name="from"/>
<word name="further"/>
<word name="furthermore"/>
<word name="g"/>
<word name="get"/>
<word name="gets"/>
<word name="getting"/>
<word name="given"/>
<word name="gives"/>
<word name="go"/>
<word name="goes"/>
<word name="going"/>
<word name="gone"/>
<word name="got"/>
<word name="gotten"/>
<word name="greetings"/>
<word name="h"/>
<word name="had"/>
<word name="happens"/>
<word name="hardly"/>
<word name="has"/>
<word name="have"/>
<word name="having"/>
<word name="he"/>
<word name="hello"/>
<word name="help"/>
<word name="hence"/>
<word name="her"/>
<word name="here"/>
<word name="hereafter"/>
<word name="hereby"/>
<word name="herein"/>
<word name="hereupon"/>
<word name="hers"/>
<word name="herself"/>
<word name="hi"/>
```

```
<word name="him"/>
<word name="himself"/>
<word name="his"/>
<word name="hither"/>
<word name="hopefully"/>
<word name="how"/>
<word name="howbeit"/>
<word name="however"/>
<word name="i"/>
<word name="ie"/>
<word name="if"/>
<word name="ignored"/>
<word name="immediate"/>
<word name="in"/>
<word name="inasmuch"/>
<word name="inc"/>
<word name="indeed"/>
<word name="indicate"/>
<word name="indicated"/>
<word name="indicates"/>
<word name="inner"/>
<word name="insofar"/>
<word name="instead"/>
<word name="into"/>
<word name="inward"/>
<word name="is"/>
<word name="it"/>
<word name="its"/>
<word name="itself"/>
<word name="j"/>
<word name="just"/>
<word name="k"/>
<word name="keep"/>
<word name="keeps"/>
<word name="kept"/>
<word name="know"/>
<word name="knows"/>
<word name="known"/>
<word name="l"/>
<word name="last"/>
<word name="lately"/>
<word name="later"/>
<word name="latter"/>
<word name="latterly"/>
<word name="least"/>
<word name="less"/>
<word name="lest"/>
<word name="let"/>
<word name="like"/>
<word name="liked"/>
<word name="likely"/>
<word name="little"/>
<word name="look"/>
<word name="looking"/>
<word name="looks"/>
<word name="ltd"/>
<word name="m"/>
<word name="mainly"/>
<word name="many"/>
<word name="may"/>
<word name="maybe"/>
<word name="me"/>
<word name="mean"/>
<word name="meanwhile"/>
<word name="merely"/>
<word name="might"/>
<word name="more"/>
<word name="moreover"/>
```

```
<word name="most"/>
<word name="mostly"/>
<word name="much"/>
<word name="must"/>
<word name="my"/>
<word name="myself"/>
<word name="n"/>
<word name="name"/>
<word name="namely"/>
<word name="nd"/>
<word name="near"/>
<word name="nearly"/>
<word name="necessary"/>
<word name="need"/>
<word name="needs"/>
<word name="neither"/>
<word name="never"/>
<word name="nevertheless"/>
<word name="new"/>
<word name="next"/>
<word name="nine"/>
<word name="no"/>
<word name="nobody"/>
<word name="non"/>
<word name="none"/>
<word name="noone"/>
<word name="nor"/>
<word name="normally"/>
<word name="not"/>
<word name="nothing"/>
<word name="novel"/>
<word name="now"/>
<word name="nowhere"/>
<word name="nyse"/>
<word name="o"/>
<word name="obviously"/>
<word name="of"/>
<word name="off"/>
<word name="often"/>
<word name="oh"/>
<word name="ok"/>
<word name="okay"/>
<word name="old"/>
<word name="on"/>
<word name="once"/>
<word name="one"/>
<word name="ones"/>
<word name="only"/>
<word name="onto"/>
<word name="or"/>
<word name="other"/>
<word name="others"/>
<word name="otherwise"/>
<word name="ought"/>
<word name="our"/>
<word name="ours"/>
<word name="ourselves"/>
<word name="out"/>
<word name="outside"/>
<word name="over"/>
<word name="overall"/>
<word name="own"/>
<word name="p"/>
<word name="particular"/>
<word name="particularly"/>
<word name="per"/>
<word name="perhaps"/>
<word name="placed"/>
```

```
<word name="please"/>
<word name="plus"/>
<word name="possible"/>
<word name="presumably"/>
<word name="probably"/>
<word name="provides"/>
<word name="q"/>
<word name="que"/>
<word name="quite"/>
<word name="qv"/>
<word name="r"/>
<word name="rather"/>
<word name="rd"/>
<word name="re"/>
<word name="really"/>
<word name="reasonably"/>
<word name="regarding"/>
<word name="regardless"/>
<word name="regards"/>
<word name="relatively"/>
<word name="respectively"/>
<word name="right"/>
<word name="s"/>
<word name="said"/>
<word name="same"/>
<word name="saw"/>
<word name="say"/>
<word name="saying"/>
<word name="says"/>
<word name="second"/>
<word name="secondly"/>
<word name="see"/>
<word name="seeing"/>
<word name="seem"/>
<word name="seemed"/>
<word name="seeming"/>
<word name="seems"/>
<word name="seen"/>
<word name="self"/>
<word name="selves"/>
<word name="sensible"/>
<word name="sent"/>
<word name="serious"/>
<word name="seriously"/>
<word name="seven"/>
<word name="several"/>
<word name="shall"/>
<word name="she"/>
<word name="should"/>
<word name="since"/>
<word name="six"/>
<word name="so"/>
<word name="some"/>
<word name="somebody"/>
<word name="somehow"/>
<word name="someone"/>
<word name="something"/>
<word name="sometime"/>
<word name="sometimes"/>
<word name="somewhat"/>
<word name="somewhere"/>
<word name="soon"/>
<word name="sorry"/>
<word name="specified"/>
<word name="specify"/>
<word name="specifying"/>
<word name="still"/>
<word name="sub"/>
```

```
<word name="such"/>
<word name="sup"/>
<word name="sure"/>
<word name="t"/>
<word name="take"/>
<word name="taken"/>
<word name="tell"/>
<word name="tends"/>
<word name="th"/>
<word name="than"/>
<word name="thank"/>
<word name="thanks"/>
<word name="thanx"/>
<word name="that"/>
<word name="thats"/>
<word name="the"/>
<word name="their"/>
<word name="theirs"/>
<word name="them"/>
<word name="themselves"/>
<word name="then"/>
<word name="thence"/>
<word name="there"/>
<word name="thereafter"/>
<word name="thereby"/>
<word name="therefore"/>
<word name="therein"/>
<word name="theres"/>
<word name="thereupon"/>
<word name="these"/>
<word name="they"/>
<word name="think"/>
<word name="third"/>
<word name="this"/>
<word name="thorough"/>
<word name="thoroughly"/>
<word name="those"/>
<word name="though"/>
<word name="three"/>
<word name="through"/>
<word name="throughout"/>
<word name="thru"/>
<word name="thus"/>
<word name="to"/>
<word name="together"/>
<word name="too"/>
<word name="took"/>
<word name="toward"/>
<word name="towards"/>
<word name="tried"/>
<word name="tries"/>
<word name="truly"/>
<word name="try"/>
<word name="trying"/>
<word name="twice"/>
<word name="two"/>
<word name="u"/>
<word name="un"/>
<word name="under"/>
<word name="unfortunately"/>
<word name="unless"/>
<word name="unlikely"/>
<word name="until"/>
<word name="unto"/>
<word name="up"/>
<word name="upon"/>
<word name="us"/>
<word name="use"/>
```

```
  <word name="used"/>
  <word name="useful"/>
  <word name="uses"/>
  <word name="using"/>
  <word name="usually"/>
  <word name="uucp"/>
  <word name="v"/>
  <word name="value"/>
  <word name="various"/>
  <word name="very"/>
  <word name="via"/>
  <word name="viz"/>
  <word name="vs"/>
  <word name="w"/>
  <word name="want"/>
  <word name="wants"/>
  <word name="was"/>
  <word name="way"/>
  <word name="we"/>
  <word name="welcome"/>
  <word name="well"/>
  <word name="went"/>
  <word name="were"/>
  <word name="what"/>
  <word name="whatever"/>
  <word name="when"/>
  <word name="whence"/>
  <word name="whenever"/>
  <word name="where"/>
  <word name="whereafter"/>
  <word name="whereas"/>
  <word name="whereby"/>
  <word name="wherein"/>
  <word name="whereupon"/>
  <word name="wherever"/>
  <word name="whether"/>
  <word name="which"/>
  <word name="while"/>
  <word name="whither"/>
  <word name="who"/>
  <word name="whoever"/>
  <word name="whole"/>
  <word name="whom"/>
  <word name="whose"/>
  <word name="why"/>
  <word name="will"/>
  <word name="willing"/>
  <word name="wish"/>
  <word name="with"/>
  <word name="within"/>
  <word name="without"/>
  <word name="wonder"/>
  <word name="would"/>
  <word name="would"/>
  <word name="x"/>
  <word name="y"/>
  <word name="yahoo"/>
  <word name="yes"/>
  <word name="yet"/>
  <word name="you"/>
  <word name="your"/>
  <word name="yours"/>
  <word name="yourself"/>
  <word name="yourselves"/>
  <word name="z"/>
  <word name="zero"/>
</stopwords>
```

D.7 GUIAgent

```
//loading jade related library
import jade.core.*;
import jade.core.behaviours.*;
import jade.gui.GuiAgent;
import jade.gui.GuiEvent;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.domain.*;
import jade.domain.FIPAAgentManagement.*;

//loading java library
import java.util.logging.*;
import java.io.*;
import java.util.*;

//loading google API library
import com.google.soap.search.*;

public class GUIAgent extends GuiAgent {

  /* these constant are used to identify
   * event in the GUI. Their functionality
   * are very similar to event id in AWT
   * but here we define our own id number
   */

  //user click close button
  public static final int END = 1001;
  //user choose seed url
  public static final int URL = 1002;
  //user choose google search
  public static final int GOOGLE = 1003;
  //user choose url from file
  public static final int FILE = 1004;
  //user choose Yahoo directory search
  public static final int YAHOO = 1005;

  //logging using standard java.util.logging
  Logger logMeBaby = Logger.getLogger(
    getName());

  //this is the class the define the GUI
  //in other words, this is what GUIAgent
  //will look like
  transient protected GUIDesign gd;

  /*
   * This method is called when agent is
   * about to be ran. It's more like a place
   * where you initialize several mandatory
   * variables, etc.
   * But here we establish the GUI and
   * call its setVisible() method to display
   * the graphics
   */

  public void setup() {
    gd = new GUIDesign(this); //get instance of gui
    gd.setVisible(true); //show the gui
  }

  /*
   * This is where the event passing actually
   * happened. This event are created by
   * postEvent() method in the GUIDesign
```

```
 * class. Here we catches that event
 * and examine their type. Their type
 * are then matched using the constant
 * (e.g. FILE, YAHOO, GOOGLE, etc) so
 * that we know what user has selected
 * in the GUI and perform appropriate
 * action
 */

protected void onGuiEvent(GuiEvent ev) {
  switch (ev.getType()) {
    case END: //user click Close All button
      gd.dispose();
      gd = null;
      doDelete();
      System.exit(0);
      break;
    case URL: //user click start and select seed url
      handleURL((String)ev.getParameter(0));
      break;
    case GOOGLE: //user click start and select google
      handleGoogle((String)ev.getParameter(0));
      break;
    case FILE: //user click start and select file
      handleFile((String)ev.getParameter(0));
      break;
    case YAHOO: //user click start and select yahoo
      handleYahoo((String)ev.getParameter(0));
  }
}

/**
 * This method is called when the agent is about
 * to be killed. All it does is just to
 * dispose the gui and make it not visible
 */
public void takeDown() {
  if (gd != null) {
    gd.dispose();
    gd.setVisible(false);
  }
}

/**
 * This method read set of urls from file. For
 * each url in the file, it will create an
 * ACLMessage that contain this url. It then
 * tries to locate distributor agent. Once
 * distributor agent is located, this acl
 * message will be send there so that search
 * agent can begin its task
 */

public void handleFile(String file) {
  try {
    BufferedReader read = new BufferedReader(
      new FileReader(file));
    String line;
    while ((line = read.readLine()) != null) {
      AID distro = findDistributorAgent();
      if (distro != null) {
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.addReceiver(distro);
        msg.setContent(line);
        //instead of using send() method
        //we use SenderBehaviour because it's thread
        //safer this way
        addBehaviour(new SenderBehaviour(this, msg));
```

```
      }
    }
  }
  catch (IOException e) {
    logMeBaby.warning("Cannot read file " + file);
  }
}

/*
 * This method queries google for the search result
 * using GoogleAPI which is pretty neat. As usual
 * we need to set the key first. GoogleAPI will only
 * return top 10 results. Each of this result will
 * be send to distributor agent. Distributor agent
 * will then spawn search agent to locate travel
 * information
 */

public void handleGoogle(String keyword) {
  GoogleSearch gs = new GoogleSearch();
  //set key (need to get the key first from
  //google web site, limit is 1000 queries per day
  gs.setKey("6FYCYP5QFHIC/Lx4FE0KalflD1rQAaWc");
  gs.setQueryString(keyword);
  try {
    GoogleSearchResult gsr = gs.doSearch();
    GoogleSearchResultElement[] elem = gsr.getResultElements();
    for (int i = 0; i<elem.length; i++) {
      AID distro = findDistributorAgent();
      if (distro != null) {
        ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
        msg.addReceiver(distro);
        msg.setContent(elem[i].getURL());
        //thread safer this way
        addBehaviour(new SenderBehaviour(this, msg));
      }
    }
  }
  catch (GoogleSearchFault gsf) {
    logMeBaby.warning("Failed to initiate google search");
  }
}

/*
 * This method perform queries using specified
 * keyword to yahoo directory. Since there are no
 * API for yahoo (like the one for Google) this task
 * have to be handle manually. Here we rely on
 * <code>YahooParser</code> to perform the parsing.
 * <code>YahooParser</code>'s result() method returns
 * set of urls as the result of the directory search
 */

public void handleYahoo(String keyword) {
  //YahooParser for performing the Yahoo Directory Search
  YahooParser yp = new YahooParser(keyword);
  Set result = yp.result();
  Iterator iter = result.iterator();
  while (iter.hasNext()) {
    AID distro = findDistributorAgent();
    if (distro != null) {
      ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
      msg.addReceiver(distro);
      msg.setContent((String)iter.next());
      //thread safer this way
      addBehaviour(new SenderBehaviour(this, msg));
    }
  }
```

```
  }

  /*
   * This is a very standard method. It just take the
   * seed url and then send it to the distributor agent
   * Distributor agent will then do its job of finding
   * links from seed url
   */

  public void handleURL(String url) {
    AID distro = findDistributorAgent();
    if (distro != null) {
      ACLMessage msg = new ACLMessage(ACLMessage.REQUEST);
      msg.addReceiver(distro);
      msg.setContent(url);
      addBehaviour(new SenderBehaviour(this, msg));
    }
  }

  /*
   * This method relies on JADE's AMS to query find
   * any Distributor Agent. Any agent that register
   * itself as type DistributeAgent is a distributor
   * agent. The result of the query is the AID of
   * the distributor agent.
   * This method is used when we composed an ACLMessage
   * and we want to send it to the distributor agent
   */

  private AID findDistributorAgent() {
    AID distro = null;
    DFAgentDescription dfd = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType("DistributeAgent");
    dfd.addServices(sd);
    try {
      while (true) {
        SearchConstraints c = new SearchConstraints();
        c.setMaxDepth(new Long(3));
        DFAgentDescription[] result =
          DFService.search(this,dfd,c);
        if ((result != null) && (result.length > 0)) {
          dfd = result[0];
          distro = dfd.getName();
          logMeBaby.info("Found Distributor Agent");
          break;
        }
      }
    }
    catch (FIPAException e) {
      logMeBaby.info("Cannot Find Distributor Agent");
    }
    return distro;
  }
}
```

D.8 GUIDesign

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import jade.gui.GuiEvent;

public class GUIDesign extends JFrame implements ActionListener {

  private GUIAgent myAgent; //the agent that will use this GUI
```

```
private JRadioButton url, google, file, yahoo; //radio button
private JButton start, end; //button
//text field to hold the keyword or url
private JTextField urlText, googleText, fileText, yahooText;
//button group for mutually exclusive radio button
private ButtonGroup rbg;

/*
 * This is the default constructor. Here we set the
 * title of the window, set the size and setup the
 * actual GUI (buttons and radio buttons)
 *
 * Since GUIDesign also is an <code>ActionListener</code>,
 * we will also register itself an the listener for these
 * components
 */

public GUIDesign(GUIAgent a) {
  super(); //calling parent's constructor
  myAgent = a; //referencing the calling agent
  setTitle("Agentlab User Interface"); //setting title of the frame
  setSize(550,300); //setting the size
  setResizable(false); //make it un-resizable

  //the label component
  JLabel label = new JLabel("Please choose of one the following:");

  //the buttons
  start = new JButton("Start");
  end = new JButton("Close All");

  //setting the layout for buttons
  JPanel buttons = new JPanel(new FlowLayout(FlowLayout.RIGHT));
  buttons.add(start); buttons.add(end);
  start.addActionListener(this);
  end.addActionListener(this);

  rbg = new ButtonGroup();

  //setting layout for seed url
  JPanel byURL = new JPanel(new FlowLayout(FlowLayout.LEFT));
  url = new JRadioButton("Seed URL:");
  url.setSelected(true); url.addActionListener(this);
  rbg.add(url);
  urlText = new JTextField(30);
  byURL.add(url); byURL.add(urlText);

  //setting layout for google search
  JPanel byGoogle = new JPanel(new FlowLayout(FlowLayout.LEFT));
  google = new JRadioButton("Using Google:");
  rbg.add(google); google.addActionListener(this);
  googleText = new JTextField(30);
  googleText.setEnabled(false);
  byGoogle.add(google); byGoogle.add(googleText);

  //setting layout for reading url from file
  JPanel byFile = new JPanel(new FlowLayout(FlowLayout.LEFT));
  file = new JRadioButton("Using File:");
  rbg.add(file); file.addActionListener(this);
  fileText = new JTextField(30);
  fileText.setEnabled(false);
  byFile.add(file); byFile.add(fileText);

  //setting layout for querying Yahoo Directory
  JPanel byYahoo = new JPanel(new FlowLayout(FlowLayout.LEFT));
  yahoo = new JRadioButton("Using Yahoo Directory:");
  rbg.add(yahoo); yahoo.addActionListener(this);
  yahooText = new JTextField(30);
```

```
    yahooText.setEnabled(false);
    byYahoo.add(yahoo); byYahoo.add(yahooText);

    //combining these area together
    JPanel selectionArea = new JPanel();
    selectionArea.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
    selectionArea.setLayout(new GridLayout(6,1));
    selectionArea.add(label);
    selectionArea.add(byURL);
    selectionArea.add(byGoogle);
    selectionArea.add(byFile);
    selectionArea.add(byYahoo);
    selectionArea.add(buttons);

    getContentPane().add(selectionArea);
  }

  /*
   * This method tracks which action has been performed
   * and on what component. Once the event has been
   * tracked down, it will then execute appropriate
   * action. For example, when user select seed
   * url radio button, all other text field (e.g.
   * in yahoo, google, and file) must be disabled
   * because it does not make sense to allow user
   * to select one radio button but type the text
   * in others
   *
   * This method also generate <code>GuiEvent</code>
   * as result of some action (e.g. clicking start
   * button). This event will be catched by
   * <code>GuiAgent</code>.
   */

  public void actionPerformed(ActionEvent e) {

    //get the source object
    Object source = e.getSource();

    //checking the source object and
    //perform appropriate actions
    if (source == url) {
      urlText.setEnabled(true);
      googleText.setEnabled(false);
      fileText.setEnabled(false);
      yahooText.setEnabled(false);
    }
    else if (source == google) {
      urlText.setEnabled(false);
      googleText.setEnabled(true);
      fileText.setEnabled(false);
      yahooText.setEnabled(false);
    }
    else if (source == file) {
      urlText.setEnabled(false);
      googleText.setEnabled(false);
      fileText.setEnabled(true);
      yahooText.setEnabled(false);
    }
    else if (source == yahoo) {
      urlText.setEnabled(false);
      googleText.setEnabled(false);
      fileText.setEnabled(false);
      yahooText.setEnabled(true);
    }
    //user click start button
    else if (source == start) {
      GuiEvent ev = null;
```

```
      //figuring out which radio button
      //is actually selected, and then
      //get the text from the text field
      //right next to the radio button
      if (url.isSelected()) {
        ev = new GuiEvent(null, myAgent.URL);
        ev.addParameter(urlText.getText());
      }
      else if (google.isSelected()) {
        ev = new GuiEvent(null, myAgent.GOOGLE);
        ev.addParameter(googleText.getText());
      }
      else if (file.isSelected()) {
        ev = new GuiEvent(null, myAgent.FILE);
        ev.addParameter(fileText.getText());
      }
      else if (yahoo.isSelected()) {
        ev = new GuiEvent(null, myAgent.YAHOO);
        ev.addParameter(yahooText.getText());
      }

      //generating event so that GUIAgent can
      //catch this
      myAgent.postGuiEvent(ev);
    }
    //user click close all button
    else if (source == end) {
      //creating GuiEvent
      GuiEvent ge = new GuiEvent(null, myAgent.END);
      //posting this event
      myAgent.postGuiEvent(ge);
    }
  }
}
```

D.9 YahooParser

```
//loading java built-in library
import java.util.regex.*;
import java.util.logging.*;
import java.net.*;
import java.util.*;
import java.io.*;

public class YahooParser {

  String keyword; //the search keyword

  //the standard java.util.logging
  Logger logMeBaby = Logger.getLogger("YahooParser");

  /*
   * Default constructor which takes the keyword
   * and convert any occurences of space into
   * the "+" sign
   */

  public YahooParser(String keyword) {
    //convert any space to "+" sign
    this.keyword = keyword.replace(' ','+');
  }

  /*
   * Produce a <code>Set</code> object that
   * contain urls from the result of Yahoo
   * Directory search.
```

```
    *
    * As usual we establish the connection
    * using URLConnection and parse the page
    * using regular expression. If the result
    * page layout changes, this method may
    * not work
    */

  public Set result() {

    Set urls = new HashSet();

    try {
      //the search url, using get method
      URL url = new URL(
        "http://search.yahoo.com/search/dir?p="+this.keyword+"&h=c");
      //opening the connection
      URLConnection conn = url.openConnection();
      //i like java way of handling I/O, see how you connect
      //one input to another input (like a pipe)
      BufferedReader br = new BufferedReader(new
        InputStreamReader(conn.getInputStream()));
      String line;
      //the regular expression
      Pattern p = Pattern.compile("^<big>\\d+.*<a href=.*[*][-](.*)\">.*$");
      while ((line = br.readLine()) != null) {
        Matcher m = p.matcher(line);
        if (m.matches()) urls.add(m.group(1)); //adding url to set
      }
    }
    catch (Exception e) {
      logMeBaby.warning("YahooParser failed to open search page");
    }

    return urls;
  }
}
```

VITA

Andy Nauli

Candidate for the Degree of

Master of Science

Thesis: USING SOFTWARE AGENT TO INDEX DATA FOR AN E-TRAVEL SYSTEM

Major Field: Computer Science

Biographical:

Personal Data: Born in Sibolga, Sumatra Utara, Indonesia, November 19, 1976, the son of Marsada Nauli and Shirley Limawan.

Education: Graduated from Tarakanita High School, Jakarta, Indonesia, in July 1995; received Bachelor of Science in Computer Science from Oklahoma State University, Stillwater, in May 2000; completed the requirements for the degree of Master of Science in Computer Science at the Computer Science Department of Oklahoma State University in August 2003.

Experience: Employed by University Extension International and Economic Development, Oklahoma State University as a Graduate Assistant from January 2001 to August 2003.