Indexing agent gathered data in an e-travel system

Marcin Paprzycki, Austin Gilbert, Andy Nauli, Minor Gordon, Steve Williams, Jimmy Wright
Computer Science Department
Oklahoma State University
Tulsa, OK 74106, USA
(marcin, austirg, nauli, minorg, stw, jimmyww}@cs.okstate.edu

Keywords: content management, Internet, software agents, data indexing, ebXML registry/repository

Received: 12 October, 2003

In this paper we discuss the problem of indexing information available on the Internet, with the ultimate goal of delivering personalized content to users of an Internet-based travel support system. We introduce the form of index tokens that will be stored in the system and describe an agent-based subsystem designed to support the indexing function. Finally, we discuss the search agent that was developed to provide the system with index tokens and allow to experiment with the proposed design..

1 Introduction

In the past decade the travel services market has developed a hugely diverse presence on the Internet, in terms of both resources offered (hotel rooms, rental cars, dinner reservations, golf tee times, "general tourist information," etc.) and approaches to offering them (e.g. aggregation, personalization, mobile delivery). For instance, a simple search using the keyword *hotel* on Google search engine returns about 82,500,000 hits sorted by their rank. Thus, as in most other domains, the potential travel services user must often deal with one of the crucial problems inherent in information diversity: the lack of an encompassing catalogue through which the content **of interest** may be located. Most, if not all, Internet search engines provide only a non-categorized and mostly non-intuitive means of locating and representing data. Furthermore, search results in the travel domain (as well as any other domain) are likely to include too many hits unrelated to actual travel choices. The Google and Yahoo directories are representative attempts to organize access to and presentation of many types of data including travel data, however, for instance, they provide no organized booking interface for the data they offer. Additionally, they do not provide any realistic means of personalization of content delivery. Finally, the Google directory consists of a mixture of travel resource types and geographical categories (see [n18] for more details) that does not necessarily constitute the best way of supporting travelers. On the other hand, some of the major travel sites such as Expedia, Travelzoo, etc. organize and attempt to personalize a limited subset of travel data (typically airline, car, hotel reservation as well as cruise and vacation package arrangements), based on a limited number of large providers and content stored in tailor-made databases within the system. Here, the mass of information stored on independent Internet sites is completely ignored. Thus, we believe that neither search engines nor the large travel sites are currently capable of providing a complete support to a modern day traveler.

Ideally, a travel support system should act as a filtering and organizing intermediary between travel consumers and travel suppliers [15]. Its primary function [1] is to find the travel information that is most relevant to a given customer at a given moment and deliver it in a well-organized and intuitive way [2]. In order to support this content-delivery role, the system must explore the Internet and other sources dynamically constructing and managing a supply of travel content from known and previously unknown providers [1, 3, 17].

In exploring the potential of such a travel support system, we have followed a two-pronged approach. First, since travel support is a paradigmatic example of the application of agent technology [14, 16], we have decided to utilize software agents as the framework of our system [1, 6]. Second, as an information broker between travel content suppliers and end users (travelers) we must carefully consider the means by which we will structure the information within the system, in order to deliver the most relevant and accurate travel choices to the consumer [2, 5, 17]. We believe that one of the more promising approaches to structuring information from diverse sources is to apply index-based techniques similar to those described in [13] (with references available there). This approach should allow 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. Note that in our work we are primarily interested in personalized delivery of travel related informational content rather than booking of travel arrangements.

The aim of our paper is twofold. First, we describe an indexing method for storing the travel content. Second we present an agent subsystem that is devoted to management of index tokens in the central repository.

Finally, we briefly describe a simple search agent that has been developed to search the Internet for the travel content and to deliver index tokens to the system.

2 Content management problem

In order for an e-travel support system to accurately reflect available travel options and information, a robust strategy for obtaining this content from sources on the Internet and managing it within the system is required. Existing content provision systems typically approach this problem in one of two ways:

- *by aggregation*: retrieving beforehand **all** information that the system will possibly need in the future, and organizing it in databases in a predefined (by humans) format for future retrieval,

- *by selection*: indexing information to maintain a "map" as to what information (and where) is available on the Internet, and retrieving the actual content only as it becomes necessary to satisfy user's queries.

Most online travel content gateways (e.g. Expedia, Travelersadvantage, etc.) employ the first method, storing the majority of browseable content locally and calling out to the primary source systems on the Internet (e.g. those run by travel providers such as airlines) for verification of locally-cached information (e.g. verified flight schedules, seat availability and ticket prices). The main advantage of this approach is the immediate local availability of content; interestingly, this is also a disadvantage, in that it leads to the problem of "data coherency." In addition, the amount of data that has to be necessarily stored locally and continuous local processing necessary for aggregation systems to operate makes them extremely resource intensive.

The majority of search engines (e.g. Yahoo, Interia, Lycos, etc.) take a hybrid approach, aggregating only a limited store of data (such as page headers and a few selected / cashed pages) necessary to support the search function. This approach attempts at striking a balance between the amount of content stored locally, frequency of local information updates and the precision of the search function. Rudimentary content organization and differentiation available in browsers combined with the relative freshness of data are a reasonable means for satisfying typical content searches (i.e. where the content changes infrequently); however, this approach is wanton when applied to travel-oriented services where the freshness of content is of paramount importance.

Our e-travel system fully embraces the second approach to content management (*by selection*) by attempting to develop a well-organized and highly cross-referenced index of Internet-based content (for a description of a number of similar systems see [1]). The proposed system dynamically utilizes remote content by referencing local indices – pointers. It focuses on the classification of content instead of the content itself, as in a library catalog (or in yellow pages), only storing enough information in indices to satisfy user queries. This approach eliminates the above mentioned problem of data coherency and is aimed at reducing the overall amount of data stored and managed locally. The downside of this approach is that the actual content must always be retrieved from a remote site. If a content provider becomes unreachable, the e-travel system is unable to retrieve the information and thus fails to fulfill the user's request. We have dealt with such a situation when a remote site providing reverse geo-coding based on an address has seized to exist leaving us with a much less desirable choice for our GIS subsystem (see section 5.1). More generally, any slowdown in reaching the primary content provider is reflected in the performance of the system. Nevertheless, in designing the e-travel system we felt that the advantages of accurate indexing combined with ability to deliver up to the minute updated information and possible optimization of local resource utilization (resources can be utilized to provide user with personalized content rather than to manage copious volumes of data) outweighed the disadvantages of remotely-stored content. We also expect that approach based on indexing will improve the limited queries options and result displays caused by traditional database logic and principles [1, 15].

3 Agent-based travel support system

3.1 History

The initial design of the travel support system was presented in [2, 5, 12, 13] and while it is being constantly modified (this paper represents such a modification), the general idea of dividing the functionality into two coordinated subsystems, one handling content management and the other content delivery [4], remains unchanged. In this paper we concentrate on the content management aspects of the system. Further details about the systems and in particular the proposed content delivery functionalities can be found in [2, 5, 12, 13]. The initial development of the system was initiated using the Grasshopper agent platform [???], however we have shortly realized that at that stage it did not fully supported the FIPA [???] standards. We have therefore switched to the JADE agent platform [???], which is build around these FIPA standards.

3.2 Sources of content indices

The travel options and information that is presented by the e-travel system originates from two types of sources on the Internet: verified and unverified. Verified sources are referred to as *Verified Content Providers* (*VCP*). This designation implies a degree of conformance to expected standards of accuracy, format, and availability of described travel options. Content from *VCP*s can be either fed directly to the system or gathered by search agents, as described in [2]. In the first case we assume that the incoming *index tokens* (pointers to available

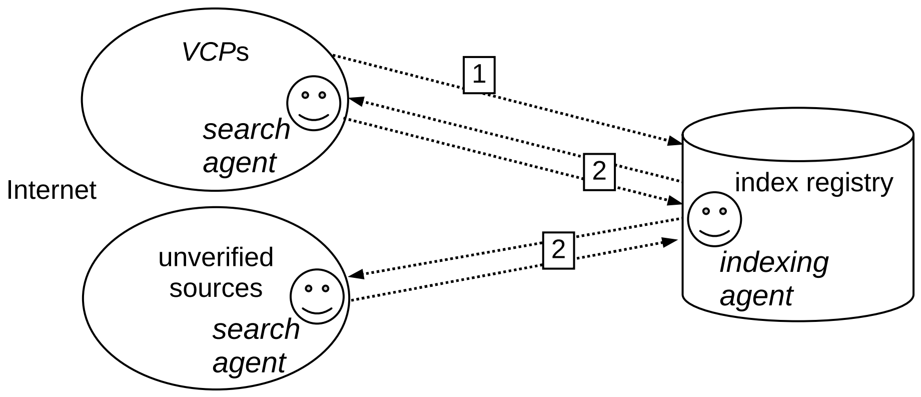

Figure 1: Information gathering and indexing; 1 – flow of index tokens originating from the *VCP*s, ready for insertion to the registry, 2 – flow of index tokens resulting from the Internet searches

information) are both in the required format and complete, and thus can be immediately stored in the system without further processing. In the second case, the acquired content indices may be incomplete and/or require further processing. When dealing with unverified sources the situation is similar to the latter case with an added component of necessary verification and deconfliction of remote information. At this stage of system design we will omit these last two issues of verification and deconfliction, assuming they have been successfully resolved. Let us note, that the proposed approach allows us to address one of the important research issues raised by Nwana and Ndumu in [11]; how to deal with dynamically changing content and form of the Internet-based information. Here, we assume that the *VCP*s are in contractual agreement with the travel agency and either, they will continue to deliver index tokens in prescribed format, or any changes in their site design will be communicated to our system, allowing it to be adjusted accordingly. Since the *VCP*s are the primary sources of the information, changes occurring in the remaining sites do not threaten the functioning of our system. Furthermore, this approach allows us to avoid most questions related to the reliability of Internet-based information. Finally, since the *VCP* provided information is assumed to be trustworthy, we can rely on them as the source of accurate information delivered to the user, while other, unverified, sources can be utilized only as supplementary resources. Regardless of source, the acquired indices are stored in the central registry for later access by the content delivery functions of the system. When the user requests information, a relevant content pointer is either found in the registry and the process of content extraction from the provider(s) is initiated (while additional search agents may be released to the Internet seeking additional content relevant to the query; in order to focus our presentation we will omit discussing this possibility), or a new index search and acquisition is forced in order discover relevant content (from both *VCP*s and unverified sources). Since the case of complete tokens being delivered directly by the *VCP*s is trivial (only an indexing agent is required to receive them and correctly store in the system), for the remaining part of this paper we will concentrate our attention on the tokens resulting from the Internet searches.

3.3 Semantics

Ideally, the content management subsystem should shield the rest of the e-travel system from the supply / retrieval mechanics of the travel content. Additionally, it should allow the content delivery functions of the systems to operate on the assumption that travel information is accurately classified. In theory, this would require the content management subsystem to semantically "understand" the information it keeps track of [5, 17, 18]. Here we have to acknowledge that currently available technology does not support this assumption of semantic "understanding" (its foundations are being developed, but are not widely accepted and thus cannot be assumed). In the absence of such technology, our system attempts the next best substitute. We apply a predefined categorical overlay to the travel information managed by the system, and allow the entire system to tune the accuracy of this overlay (e.g. with user, agent and supplier feedback, as described in [6]), with the ostensible goal of simulating real semantic classification. In addition, we pay close attention to the efforts initiated by the Open Travel Alliance that attempts at introducing a hierarchical description of the "world of travel" and most important processes taking place there [9] (see also [18n] for more details). Note that, while currently not operating on the semantic level, most of the functions of the proposed system can be adjusted to involve, for instance, RDF / OWL based ontology / semantics.

4 Structure of index tokens

The e-travel system relies heavily on the accuracy and completeness of local content indices. They must be succinct enough to be easily acquired and stored, yet verbose enough to satisfy all of the requirements of both content management and content delivery subsystems.

Consider the following scenario: a user wishes to make travel arrangements to visit Mt. Rushmore, a historical monument. The user must first travel to South Dakota (requiring a means of transportation), and perhaps find a place to stay (hotels in the area). She may also wish to know about local restaurants or other places of interest. In order to satisfy the user's request for travel arrangements, the system must initially make two major distinctions based upon the query alone: location (South Dakota) and desired destination/attraction (Mt. Rushmore). In addition, the e-travel system must also be able to resolve multiple providers of content relating to Mt. Rushmore, in order to find those indices, which will eventually yield the most desirable response for the user (for the purpose of this paper we skip the question of content provider ranking, which is one of the possible ways of dealing with potential information overload).

Our current design of indices evolved from our early attempts to develop a classification system of the world of travel content [2], and was adapted to satisfy the above requirements. We now describe an index as a tuple consisting of:

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

Here, the *?notes?* component is added to the tuple to support of various administrative functions necessary when dealing with data delivered by the search agents (for more details see Section 5). Let us now look into the *provider, type* and *location* fields of the tuple in more detail.

4.1 The *provider* component

The *provider* component describes the means of accessing travel resources on the Internet. It is stored in the form of a Uniform Resource Identifier (URI). This URI describes the access method for the resource, the location of the resource, and any marker data that may be unique to this resource within the provider. In addition to explicitly identifying the transport protocol, the protocol section also (directly or indirectly) identifies the access methods of the server. For example, http:// and ota:// each have their respective access methods (hypertext and Open Travel Alliance protocols). Other possible protocols include edi:// and soap://. The URI also contains the host name to communicate with using this protocol. Let us also note that our system is capable of efficiently dealing with situation when multiple providers supply information pertinent to a given travel resource. In this case multiple index tokens varying only in the *provider* component will be "co-stored" in the repository for efficient retrieval (for more details see [17]).

4.2 The *type* component

The *type* component of a tuple describes the position of a travel resource in the taxonomic hierarchy of all resources (e.g. Accommodations -> Hotels -> Chains). The system will utilize this information to filter out content that for some reasons (i.e. in the context of a given query, or for a particular user) is not pertinent to a user's needs. Thus, it is the focal point for the proto-semantic division of travel information. For example: if the user is interested in hotels, an agent will be able to retrieve only hotel indices from the repository. Current version of our hierarchical taxonomy for the *type* component is derived from the modified Yahoo! directory of Travel and the Open Travel Alliance [9] XML Schemas (see also [n18] for more details). The content type is intended to define the relationships between travel resources.

4.3 The *location* component

Geography and location are key factors for determining the relevance of indexed travel resources to a particular user's travel plans. The *location* component must be flexible enough to support the multiple ways it may be utilized. Location information must be specific enough to differentiate between different sites. It must be hierarchical so that organizational relationships between sites at different locations on different levels (continent, country, state, city, et al.) can be surmised (e.g. the destination is in a different country). Given these criteria, our initial design of the *location* component consists of: a taxonomic description based on the ISO-3166 standard, which defines the continent, country, state or province, and city; and the latitude and longitude for exact locations and proximity searches. These are represented in the *ebXML* hierarchy. However, in the tuple itself we store the geographical information in the form of a (latitude, longitude) pair. This form as been selected due to the need of processing geospatial information beyond simple information that a given place of interest is, for instance, located in Claremore, Oklahoma, United States.

The *type*/*location*/*provider* tuple as described above, located in hierarchical structures representing resources and geospatial locations, is the basis of the classification scheme to be utilized by all of the functions of the travel support system, from the retrieval of content from travel suppliers on the Internet to the delivery of travel choices to the end user. It is with these functions in mind that we proceed to manifest the tuple on the implementation level, and, we hope, provide an efficient means of communicating travel content. Let us also observe that the proposed schematic solves the, above indicated, problem of the Google directory [n18]. In our approach we are able to untangle the geospatial information from the travel resource information by providing two separate but complimentary "looks" at our data. In this way, we are also making an initial step toward developing ontology of travel.

The following is an example of a complete index token that is ready to be stored in the system (the *?notes?* filed is omitted, but in this case it would contain information that the token is complete and no further processing is required):

(edi://www.drp_sushi_palace.com/, restaurant, (25'45'', 34'67''))

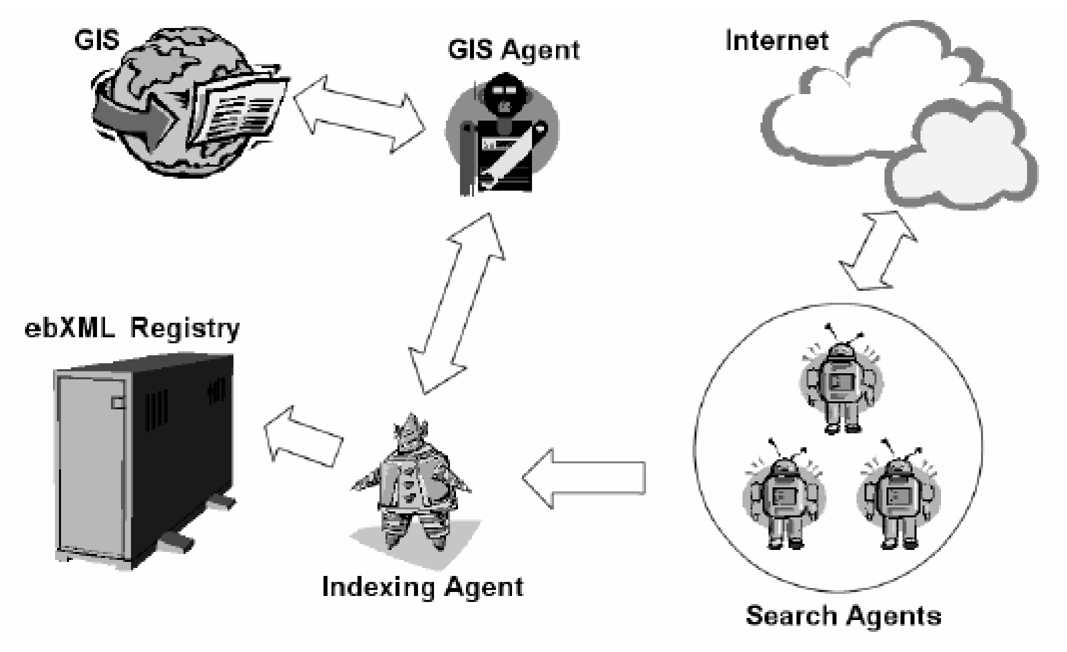

Figure 2: Proposed architecture for indexing travel data from the Internet.

Here, information about a *restaurant* is available at www.drp_sushi_palace.com and the communication protocol with that site is *edi://* and the location of the restaurant is *25'45'', 34'67''* (the details of the ISO-3166 location will be retrievable from the position of the token in the geo-tree structure in the registry, while the restaurant is positioned within a hierarchical structure of types). Once a complete index token is successfully inserted into the registry, it is ready for processing by the content delivery subsystem (as described in [2, 5, 12]) and can be utilized to prepare responses to user queries. To implement the storage of index tokens, we have decided to utilize the turned to the ebXML Registry / Repository (for an extended discussion of index storage see [17]).

5 Index acquisition

We now consider the actual process of index acquisition. As indicated above, there are two separable sources of index tokens: *VCP*s that feed complete indices directly to the system (this relationship is pre-defined by agreements between selected providers and the e-travel system); and *search agents*, which explore both the remaining *VCPs* and other repositories on the Internet. Tokens acquired by *search agents* may or may not be complete, and if their source is unverified, the content referred to should be validated and deconflicted (in the case when there is no way to verify the information, the *?notes?* field will be utilized to store such an information so that in the content delivery subsystem such information can be treated accordingly, when delivered to the user). Within the system all incoming tokens (from the *VCP*s and *search agents*) are received and handled by an *indexing agent*, which inserts them into the *registry*. Incomplete or not yet validated tokens are marked as such in the *?notes?* field of the index tuple. Furthermore, incoming tokens may have various priority levels, also indicated in the *?notes?* field. For instance, tokens acquired by the *search agents* for a user currently interacting with the system will have to be made ready for use (completed, and if necessary validated and deconflicted) as quickly as possible, while other tokens (it is assumed that in a fully operational system *search agents* continually traverse the Internet in search of travel-related information, similarly, for instance to the Google-bots) may be processed when the system is "idle." Thus the content management subsystem, as we described it so far, consists of index tokens being fed directly by the *VCP*s and *search agents* that find location of pertinent travel related content and generate index tokens; furthermore we have one or more *indexing agents* that store index tokens in the registry (number of *indexing agents* will depend on the scalability needs of the system). The JADE-based implementation of our system helps facilitating agent interactions. First, communication between the *search agents* and the *indexing agent(s)* (as well as all other inter-agent communication) is facilitated using ACL messages which are implemented in compliance of the FIPA standard. Second, *search agents* can locate *indexing agent* by simply querying the JADE Directory Facilitator. The following code snippet illustrates the method invoked to achieve this goal:

```
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");
}
```

When this method returns, the variable index will be pointing to the Agent Identifier of the *indexing agent*.

5.1 The *GIS agent*

As noted above, the location component of the index token tuple is to be represented as a (latitude, longitude) pair. Typically, geospatial data available on the Internet is not represented in such form. Thus most index tokens delivered by the *search agents* will not have the correct form; typically an empty location field and a note specifying token's incompleteness in the *?notes?* field. To deal with this situation as well as to support a number of other important functions in the content delivery subsystem a *GIS agent* has been developed. In the context of this paper, the main role of the *GIS agent* is to fill the (latitude, longitude) data of the index token. This is the standard reverse geo-coding function, where the input is an address (found within a web resource by the *search agents*) and output is the (latitude, longitude) pair. Current implementation of the *GIS agent* relies on external party to provide reverse geo-coding functionality. As mentioned above, initially we were able to locate a service that provided free of charge the required functionality, but shortly afterwards this site seized to exist. Thus, in our subsequent experiments we have utilized the http://mapper.acme.com site. This website accepts the request for GIS queries interactively (e.g. using a form and an input box). This can be easily transformed using Java's HttpURLConnection class. The form is submitted using the get method and thus it 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.

While, obviously, this particular service is only of limited capability – it accepts only ZIP codes and only of locations in US – this level of detail provided by a free of charge system was satisfactory for our proof of concept system. Obviously, in a real system geospatial information would have to be more precise than one that is based solely on ZIP codes. Such information is available (including locations outside of United States) and can be easily incorporated into our system. Unfortunately, services delivering robust reverse geo-coding are not provided free of charge and we have decided to utilize ZIP-code-based service only.

Summarizing, in the current implementation of the content management subsystem, the *GIS agent* receives the ZIP code information from the *indexing agent* (send as an ACL message) and contacts the acme.mapper.com site to obtain the (latitude, longitude) pair (our implementation utilizes a slight shortcut as the search agents deliver also the ZIP code instead of a token with an empty location field; this latter solution that was postulated above requires implementation fo auxiliary agents; see Sections 5.2). The resulting information is send back to the *indexing agent* (again, as an ACL message) which then completes the token and inserts it in the *repository*. Overall, the simplified schema of the system has been depicted in Figure 2.

As noted earlier, in a travel support system, there is a need for a much broader support for geospatial data processing. For instance, it will be necessary to be able to respond to distance oriented queries of the type "how far is from a given restaurant X to a given movie theater Y," or "which restaurants are in a certain distance from a given hotel Z." These functions can be either implemented inside of the same *GIS agent* or each of the particular sub-functions can be implemented as a separate *GIS agent*. While the second solution seems to follow more closely the spirit of agent system development (where separate functions are represented by separate agents) and, furthermore, the particular GIS functions will be naturally separated between content management and content delivery subsystems, the final decision about the agent-based implementation of the complete set of required geospatial data processing functions will be made in the next iteration of system development.

5.2 Auxiliary agents

As discussed above, one of the problems in indexing data originating from the Internet is the need of dealing with incomplete index tokens returned by the search agents. No agent can acquire information that is simply not available. As indicated above, the majority of content providers do not provide geospatial information in the form desired by our system. Rather they feature an address (complete or partial). Thus the system will have to properly manage incomplete index tokens (at this stage we will consider as incomplete also tokens gathered from unverified sources). To achieve this goal incomplete tokens are flagged as incomplete in the *?notes?* field, assigned priority and inserted into the registry. They are then processed by token completion, validation, deconfliction (*CVD*) *agents*. These agents traverse the registry and process the incomplete / unverified tokens. As an example let us consider the case of a token that is missing the location data. It is known who is the *provider* of the data, the *type* is also known (it is a hotel), while the *location* field contains no data and the *?notes?* field specifies an incomplete token with high priority. The *CVD agent* will therefore create an instance of a *query agent*. This agent will communicate with the content provider (using the specified protocol available from the *provider* field) and establish that the hotel in question is the Fairmont Hotel in San Francisco and recover its street address (from the provider, or from a different content provider discovered during separate web-searches). This information will be returned as an ACL message to the *CVD agent* responsible for managing this particular token. The *CVD agent* will then contact (via. An ACL message) the above described *GIS agent* (see [2, 12, 13] for more details) where reverse geo-coding will result in the (latitude, longitude) pair. This information will be then inserted it into the token and the incompleteness flag removed form the *?notes?* field, thus making it a full member of the *registry*. While, currently, this functionality is not yet implemented, its implementation is one of our next goals in the development of the system.

6 Content gathering

The difficult problems of content indexing and retrieval are representative of a crucial issue confronted in Internet-related research: how to introduce "understanding" to machine-web interaction. One of the reasons that many online content gateways choose the aggregation approach to content management is because it is easier to implement, despite its resource-intensiveness. The more "intelligent", selective approach of indexing content for later utility requires an in-depth, machine "understanding" of the content in order to reliably utilize it.

6.1 Interpreting sources

In recent years there has been a resurgence of interest in ontologies as a way of dealing with the problem of machines "understanding" the semantics of information on the web. Many claim that agents with ontologies will be the next breakthrough technologies for web applications [6]. This has been the thrust of the Semantic Web project [14] – the development of an ontology-described content infrastructure that will allow machines to interpret semantics as opposed to mere syntax. This capability has been realized in web pages hosted by several organizations. According to the DAML Crawler [3], as of the time of our writing, there are semantically 21,025 annotated web pages. Unfortunately, this number is negligible compared to the total of 7 billion web pages on the Internet. Therefore, today, it is not realistic to assume that agents can simply understand the web-content.

The design of our e-travel system takes into account the eventual existence of a semantically-described web; and, in particular, development of a complete and generally accepted ontology of travel, but it does not rely on it. Rather, we plan to implement an intermediate solution that allows us to depend on agent "understanding" only within the e-travel system, a working assumption which is supported by adapting the perimeter of the system (i.e. the index acquisition system) to simulate semantic gathering [6, 14].

One of the typical approaches to developing agents with the necessary functionalities is through topical web crawlers [9]. Topical web crawlers take advantage of knowing the context of the query to differentiate between the relevant and irrelevant web pages. Web pages are considered to be relevant if their similarity value satisfy a given threshold. Similarity value is calculated based on lexical analysis of the web page.

Another approach to semantic understanding of the web is through application of wrappers. For example, information agents in Heracles [15] are trained to locate meaningful information in the web pages by being shown examples consisting of web pages labeled with markers to indicate where the information is located. These examples are then used to develop a set of wrappers that are subsequently utilized in intelligent searches.

6.2 Simple search agent

While acknowledging that the above described techniques are already relatively sophisticated and new techniques are constantly being developed, for the purpose of our demonstrator system we have decided to pursue a more simplistic approach. Note, however, that we rely here on one of the important advantages of agent-based system design. Our *search agents* were implemented to verify the design of the system, to fill-in the *registry* with tokens, to pursue initial efficiency and scalability studies. As the system matures, our simple *search agents* will be replaced by more sophisticated *agents* and the system will continue its work without any additional changes.

The *search agent* must be designed so that it can classify a web page into the correct travel resource and finding necessary information to create its index token. Our approach is to use a simple statistical method to calculate the similarity of a web page to a set of given keywords (or query). This statistical method compares the content of the web page with keywords that represent a travel resource. The similarity value of the web page and the keywords are then computed. This value is then used to decide if the given web resource matches the travel resource. In implementing this functionality we have utilized existing software.

In designing our *search agent*, we utilized several software packages. Assume that our agent accessed a web page. First, the HTML Parser [7] was used to strip out all HTML-based formatting instructions. The stripped-out HTML page was then fed to the Apache Lucene [2a] for statistical analysis. The statistical analysis process begins with applying a lower case filter, which turns all words into lower case. The second step consists of removing the stop-words (words with no meaning e.g. the, than, of, which, were, are, etc.). This allows us to reduce the size of the index file. The list of stop words was based on [19n]. In the third step, Porter Stem Filter [9a] was applied to convert words into their basic form (e.g. running into run, watches into watch). The final step was to compute the statistical similarity of the filtered content to the travel resource keywords provided to the system. The Apache Lucene package includes all these steps. It also implements the vector space model to calculate the similarity value. The vector space model works by comparing the frequency of words that appear in the document with a set of given keywords using the formula:

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

where td_{ij} denoted the i^{th} term in the vector for the document j, tq_{ik} denotes the i^{th} term in the query vector k and n number of unique terms in the data set.

We have experimented with the above described simple search agent in two ways. First, to obtain some indication of its accuracy. For this purpose we have implemented a GUI front end to communicate with JADE agents. As expected, due to the simplicity of the search agent, the overall results were rather disappointing (correct identification of about 25% of visited web pages, when a single keyword was used). At the same time, these results seem promising, as a much better recognition rate can be easily obtained (more details about our experiments and their results can be found in [20]). More importantly, we were able to develop a working system in which *search agents* searched the web and produced index tokens. These index tokes were completed through interactions between the *indexing agent* and the *GIS agent*. Finally, the *indexing agent* was able to utilize the Java API for XML Registry (*JAXR*) to insert completed tokens into the ebXML Reiztry /Repositiry.

7 Concluding remarks

In this paper we have reported on our progress in developing an agent-based travel support system. Our principal motivation is an attempt at implementation of a realistic agent system that can be used to establish potential and limitations of a more general class of agent-based systems. In this we follow the methodological lead of Nwana and Ndumu [11] who have stressed the importance of the implementation and experimentation phases of agent system development. We are also challenged by the fact that all the past projects have been limited in scope [10, 16] or abandoned in early stages of development.

At the time of writing of this paper we have implemented (1) the hierarchical classification schemes for the *type* and the *location* components and instantiated them in the *ebXML registry / repository* [17]; (2) the *simple search agent*, the *indexing agent* and the *GIS agent*; (3) communication between them. In this way we were able to perform initial experiments with inserting and storing tokens in the *registry*.

These initial experiments indicate that we will have to rethink the way in which the index tokens are stored and operated on. For still unknown reasons we have run into a number of problems with the ebXML Registry/Repository. While all necessary operations worked well when its native GUI interface was used, we were constantly running into problems when combining the Repository with the JAXR or other insertion techniques. Some of these problems were of technical nature e.g. hanging registry, runaway threads etc., but also related to scalability e.g. when running into a brick wall of impossibilities when attempting to instantiate in the Registry a complete ISO-3166 classification for the United States. Since the latter problems could not have been related to the computer hardware (we have been using P4, 2.4 GHz based server with 1 Gbyte of memory), we tend to believe that this may be a problem with the currently existing implementation of the Registry. Establishing this fact was one of the important lessons learned from our experiments. This will force us to re-evaluate the token storage technology before the next step in system design and implementation. Observe, however, that while the token storage technology may change, this will no affect other parts of the system.

Obviously, we recognize the drawbacks of relying on third-party GIS subsystem as the primary source for latitude and longitude information (limited to US and Canadian address only). However, as indicated above, we consider such drawbacks to be insignificant during the system development time. Finally, our categorization of the world of travel (the hierarchy used to structure the information stored in the *type* field) is primarily based on the Yahoo! catalog and the work of the Open Travel Alliance (OTA) [18n] and, obviously it needs to be re-thought and improved on the basis of our experiments.

This leads us to the obvious fact that there exist a large number of research and/or practical issues that need to be addressed in the near future. Let us list some of them (and, obviously, this list is only a partial one): (1) re-evaluation of the index storage technology, with a strong possibility of replacing the ebXML Registry/Repository by a more robust solution; (2) completion of the content management subsystem as described in this paper (including the auxiliary agents) – this would allow us to launch the system to automatically collect index tokens and populate the registry for further experiments; (3) addressing the question of intelligence of search agents – we would like them to be effective in filtering web content and supplying our system with complete index tokens while being relatively lightweight – and, definitely we need high reliability results when we will start to automatically populate the repository (here it will be better to reject a correctly categorized resource than to accept an incorrectly categorized one); (4) investigating how many agents of various types (indexing, token completion, search, GIS etc.) are required to prevent processing bottlenecks in the content management subsystem; (5) evaluating if the proposed indexing schema is robust enough to support the content delivery functions; (6) study how does the proposed indexing schema match with the personalization oriented functions that the system is to support (in particular user behavior data storing and mining [5]). Our experimental findings (like the fact that the ebXML Registry/Repository may not be capable of supporting our needs) indicate that the above listed research questions will have to be investigated both theoretically and practically. As suggested in [11] it will be the experiments that will play a crucial role in guiding the development of our system. We will report on our progress in subsequent publications.

References

[1] Abramowicz, W., Kalczyński, P., Węcel, K. (2002) “Filtering the Web to Feed Data Warehouses.” Springer Verlag Publishing, New York.

[2] Angryk, R., Galant, G, Gordon, M., Paprzycki M. (2002) “Travel Support System – an Agent-Based Framework,” Proceedings of the International Conference on Internet Computing (IC’02), CSREA Press, Las Vegas, pp. 719-725

[2a] Apache Lucene from http://jakarta.apache.org

[3] DAML Crawler (n.d.). Retrieved Feb 11, 2003 from http://www.daml.org/crawler/

[4] V. Galant, J. Jakubczyc and M. Paprzycki. “Infrastructure for E-Commerce.” In Nycz M., Owoc M. L. (eds.), Proceedings of the 10th Conference on Knowledge Extraction from Databases, Wrocław University of Economics Press, 2002, 32-47.

[5] Galant V. and Paprzycki M. (2002) “Information Personalization in an Internet Based Travel Support System.” Proceedings of the BIS’2002 Conference, Poznań, Poland, April, 2002, pp. 191-202

[6] Hendler, J. (2001) “Agents and semantic web,” IEEE Intelligent Systems Journal, 16(2), pp. 30-37

[7] HTMLParser (n.d.). Retrieved Feb 11, 2003 from http://htmlparser.sourceforge.net

[8] JADE (n.d.). Retrieved Feb 11, 2003 from http://jade.cselt.it

[9] Menczer, F., Pant, G., and Srinivasan, P. “Topical Web Crawlers: Evaluating Adaptive Algorithms,” ACM Transaction on Internet Technology, 5(N), pp. 1-38

[9a]Martin Porter. Porter stemming algorithm. http://www.tartarus.org/ ~martin/index.html.

[10] Ndumu, D., Collins, J., Nwana, H. (1998) “Towards Desktop Personal Travel Agents,” BT Technological Journal, 16 (3), pp. 69-78

[11] H. Nwana, D. Ndumu, A Perspective on Software Agents Research, The Knowledge Engineering Review, 14 (2), 1999, 1-18

[12] Paprzycki M., Angryk R., Kołodziej K., Fiedorowicz I., Cobb M., Ali D. and Rahimi S. (2001) “Development of a Travel Support System Based on Intelligent Agent Technology,” in: S. Niwiński (ed.), Proceedings of the PIONIER 2001 Conference, Technical University of Poznań Press, Poznań, Poland, pp. 243-255

[13] Paprzycki M., Kalczyński P. J., Fiedorowicz I., Abramowicz W. and Cobb M. (2001) “Personalized Traveler Information System,” in: Kubiak B. F. and Korowicki A. (eds.), Proceedings of the 5th International Conference Human-Computer Interaction, Akwila Press, Gdańsk, Poland, pp. 445-456

[14] Semantic Web (n.d.). Retrieved Feb 11, 2003 from http://www.semanticweb.org

Intelligent Systems for Tourism", in IEEE Intelligent Systems, November/December 2002, pp. 53-55

[16] Suarez J. N., O’Sullivan D., Brouchoud H., Cros P. (1999) “Personal Travel Market: Real-Life Application of the FIPA Standards.” Technical Report, BT, Project AC317

[17] Wright, J., Williams, S., Paprzycki, M., Harrington, P., Using ebXML Registry/Repository to Manage Information in an Internet Travel Support System, in: W. Abramowicz and G. Klein (eds.), Proceedings of the BIS'2003 Conference, Poznań University of Economics Press, Poznań, Poland, 2003, 81-89

[18] M. Paprzycki, A. Gilbert, M. Gordon, J. Wright, “The World of Travel: a Comparative Analysis of Classification Methods,” *Annales UMCS Informatica*, A1, 2003, 259-270

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

[20] A. Nauli, Using software agents to index data for an e-travel system, Masters Thesis, Oklahoma State University, 2003

[1] Author (year) *Title of the book*, Publisher.

[2] Author (year) Title of the paper, *Title of the journal*, Publisher, pp. nn--mm.

[3] Author (year) Title of the paper, *Title of the proceedings*, Publisher, Location, pp. nn--mm.