An Ontology-Supported Information Agent Shell for Ubiquitous Services

Sheng-Yuan Yang
Dept. of Computer and Communication Engineering
St. John’s University
Taipei, Taiwan
e-mail: ysy@mail.sju.edu.tw

Abstract—The paper proposed and developed a core component an “information agent shell,” IAS, to apply in the general Web systems and investigate the related problems with the extension in a wide variety of application areas. The shell contains the four main components: OntoCrawler, OntoExtractor, OntoClassifier, and OntoRecommender, which are responsible for information searching, information extracting, information classifying, and information representing/ranking, respectively. Inside the shell, it would introduce the ubiquitous interface agent more to explore the related problems of the ontology-supported information agent shell in ubiquitous environments. The preliminary experimental outcomes can also show forth on basic efficiencies of related sub-systems.

Keywords—Information Agent Shell, Ontology, Ubiquitous Services.

I. INTRODUCTION

Information agent is a software product for assisting and guiding users to reach the goal of information retrieval. There are lots of agent applications in literatures. For example, Wu [7] focused on ebXML framework, which offered a way of executing both ebXML Collaboration Protocol Agreement and ebXML Business Process Specification Schema automatically, and through the technique of agent, manage numerous business processes; Pakonen et al [4] not only presented a multi-agent-based architecture for process automation, which aims to support process operators in their monitoring activities, but also presented an agent based concept for process automation that provides operators with easily configured information retrieval and monitoring services, releasing them from tedious data harvesting tasks [5]. In the discussion on the above literatures, not only can information agents possess the four main functions: information searching, information extracting, information classifying, and information representing/ranking, but also it can really and effectively up-rise the performance of information query to the user and collocate the factors in user interfaces, network speed, amount of the backend databases, and usage scenarios. Up to now, however, most of Web information agent systems are closely related to the domain knowledge that can not directly apply to Web systems resulting from the core part of information agent in ubiquitous environments. As in the beginning of 1980s, after the development of the medical expert system MYCIN, the expert system shells: EMYCIN and TMYCIN (Tiny EMYCIN) have been developed by removing its knowledge bases. The main motivation of this paper on development of an information agent shell that was the core component suited to lots of web systems is to explore the feasibilities on construction of an information agent shell and expansion of its application areas. The preliminary experimental outcomes can also show forth on basic efficiencies of related sub-systems.

II. SYSTEM ARCHITECTURE

Figure 1 illustrates the OntoIAS architecture diagram. It contains the four main modules of information agents, including information searching, information extracting, information classifying, and information presenting/ranking and orderly corresponding to OntoCrawler, OntoExtractor, OntoClassifier, and OntoRecommender, respectively. The advantages of the approach are practicably the shell application areas based on the domain ontology and extensible its application areas according to assimilation and integration processing from other related domain ontologies. Ontological Database (OD) is a stored structure designed according to the ontology structure, serving as an ontology-directed canonical format for storing webpage information processed by OntoIAS, which is separated from the shell. Finally, the system relied on the stream technology to access backend databases. Users can employ Ubi-Interface Agent of OntoIAS to use the shell via related mobile equipments. User profile databases are responsible for recording relevant user models. The system can trigger OntoRecommender to provide relatively personal information services, and related techniques of every part were described as below.

A. Construction and Support of Ontology and Related Services

The system employed Protégé to construct the infrastructure of domain knowledge, detailed construction steps in [10]; furthermore, utilized Protégé’s APIs to form those primitive functions for supporting ontology applications. First of all, we conducted statistics and survey of related typical homepages to fetch out the related concepts and their synonym appearing in the homepage; and then employed Protégé to conduct the infrastructure of the ontology database; finally transferred the ontology built with Protégé into the XML format for doing the proper fixing and stored in MS SQL database for completing the ontology database construction. The author had employed the above ontology construction procedure to develop a scholar’s ontology [12], as shown in Figure 2, to support searching and classifying scholar webpages, and accordingly obtained correct efficiency.
Figure 3 showed the system operational structure of OntoCrawler [12]. The Action module can transfer internal query into URI code, and then embed into Google’s/Yahoo’s query URL. The LinkToGoogle module is responsible for declaring an URL object and adding above query URL on well transferred URI code; and then using a Java’s object: BufferedReader to read and used while loop to add webpage content line by line; finally, outputting the reading content as text file as final analysis reference. The RetrieveLinks module can use the default regular expressions [9,12] to search for whether there are matched URL, at the same time, output the txt file to provide the system for further processing. The RetrieveContent module is responsible for checking one URL link once a time and really linked the URL. After judging what kind coding of the webpage was, the system read in the html source file of webpage with correct coding and output it as text file so as to let system conduct further processing. After completing all procedures mentioned above, the system could used SearchMatches method to judge whether the webpage was located in the range we queried; supposed the answer was “yes”, the system would execute RemoveHTMLLabel to delete the html label from source file and remained only the text content so as to let system conduct further processing and analyzing. The SearchMatches module can support RetrieveContent internal calling service for judging whether the webpage was the range we queried. It linked the ontology database and fetched out the content to compare content of the string when using this linking method. The RemoveHTMLLabel module just like SearchMatches, it supported RetrieveContent internal calling service and deleted html label in the html source file. This technique has practically been installed in Google and Yahoo search engines and furthermore searched some related scholars’ personal webpages and accordingly stored the results into a database to let the backend systems to do advanced processes [12].

C. Structure of OntoExtractor

Basically, OntoExtractor bases on the wrapper technique [10] to help parse, retrieve, and transform webpage information and stores it in an ontology-directed database, i.e., OD. It is a stored structure designed according to the ontology structure, shown in Table I, which in turn contains a field of “Query keywords” to represent the query intent. Other important fields in the structure include “segmented words of webpage” to record the word segmentation results from the webpage produced by the segmentation software MMSEG (http://technology.chtsai.org/mmseg/); and “number of feedbacks”, “date of feedbacks” and “aging count” to support the aging and anti-aging mechanism. Still other fields are related to statistics information to help speed up the system performance, including “number of webpage keywords”, “appearance frequency of webpage keywords”, and “total satisfaction degree”. Finally, the structure has some fields to store auxiliary information to help tracing back to the original webpage, including “original webpage” and “webpage URL”. The design philosophy is to conveniently employ the ontology primitive functions for deleting conflicts among webpage keywords, gathering result statistics, and getting the best retrieval beads on the statistic results.

![Figure 4. System structure of OntoExtractor](image)

Figure 4 shows the system operational structure of OntoExtractor. Webpage Parser deletes unnecessary spaces and segments the words in the webpage using MMSEG. The results of MMSEG segmentation were bad, for the predefined MMSEG word corpus contains insufficient terms of the specific domain. The system can easily fix this by using Ontology Base as a second word corpus to bring those mis-segmented words back. Keyword Extractor is responsible for building canonical keyword indices. It first extracts keywords from the segmented words, applies the ontology services to check whether they are ontology terms, and then eliminates ambiguous or conflicting terms accordingly. It then treats the remained, consistent keywords as canonical keywords and makes them the indices for OD. Finally, Keyword Statistics calculates statistic information associated with the canonical ontological keywords and stores them in proper database tables. This technique has practically been supported by PCDIV ontology [10], which can not only precisely eliminate ambiguous, missing or conflicting terms but also accurately and effectively achieve 5 to 20% improvement in support to operation of backend system.

D. Structure of OntoClassifier

Figure 5 illustrated the system operational structure of OntoClassifier [9]. The Action module can input the pure text of webpage after excluding HTML tags and fixing segmentation errors. Its initial task included as following: loading Stop Word database, ontology database, and normalization database. The Text Processing module contains three parts of works. Firstly, the Stop words Filtering sub-module used “stop list” to store stop words and ought to be excluded when indexing words and phrases so as to decrease the noise in document and increase classification precision for improving system performance. The Normalization sub-module loaded the source text and normalized database array and then searched the source text with the regular expressions to replace every non-expression word with space character till
the end of the source text. Finally, the Stemming sub-module did the stem transformation means transfer different word type into stem to arise the extracting precision. Finally, OntoClassifier employed the Two-stage Classifier module to classify a webpage in two stages [9]. The first stage uses representative ontology features. The classifier employed the level threshold to limit the number of ontology features to be involved in this stage. If for any reason the first stage cannot return a class for a webpage, OntoClassifier moved to the second stage of classification. The second stage no longer uses level thresholds but gives an ontology feature a proper weight according to which level it is associated with. The Two-stage Classifier technique has practically been supported by PCDIY ontology [9], which can not only precisely and stably classify webpages but also accurately support semantic annotation.

![Figure 5. System structure of OntoClassifier](image)

E. Approach of OntoRecommender

Using the full keywords match method, OntoRecommender retrieves only those webpages contain all the query keywords from OD as candidate outputs. If none of webpages can be located, it turns to the partial keywords match method to find solutions. This method allows us to select the best half number of query keywords according to their TFIDF values and use them to retrieve webpages from OD. Finally, the system can apply different ranking methods to rank the retrieval results [10]. During retrieval of recommendation webpages, OntoRecommender trimmed irrelevant query keywords supported by ontology services, employed either full keywords match or partial keywords match to retrieve suitable and exact webpages, and removes conflicting webpages before turning the final results to the user. Ontology plays the key role in all the above activities. To produce a more effective presentation of the recommendation results, the system employs an enhanced ranking technique, which includes Appearance Probability, Satisfaction Value, Compatibility Value, and Statistic Similarity Value as four measures with proper weights to rank the recommendation webpages [8]. The system is also equipped with an aging and anti-aging mechanism based on the user feedback to properly track the hot topics. This technique has practically been supported by PCDIY ontology and related experiments show it does improve precision rate and produces better ranking results [10].

F. Ontology Assimilation and Integration

In this paper, we employed WordNet (http://wordnet.princeton.edu/) to be the base of the match mode, and then utilized the similarity concept of Jaccard [6] to estimate the consistencies of ontology concepts each other. The basic concept is to use the consistencies and relevant locations between the domain concepts needed to assimilate and integrate and corresponding concepts of WordNet to index those concepts, and accordingly employ the identification code Synset_ID of WordNet to access those concepts for supporting system operation. Briefly, we utilized OntoIAS to carry the assimilation and integration of related domain ontologies out. Firstly, OntoCrawler is responsible for searching related webpages of domain ontologies; OntoExtractor is responsible for per-processing those webpages; and then OntoClassifier is responsible for classifying domain ontologies extracted from them; finally, OntoRecommender is responsible for executing the necessary of ranking and indexing (or storing). Before classification processing, the system has to extract Main Concept Candidates in advance, for example on OWL/RDF(s), those relationships can be defined in “owl:Functional Property,” “owl:InverseFunctional Property,” and “owl:ObjectProperty.” Therefore, we can define related parsing rules [2] in OntoExtractor for extracting those main concepts, in turn, the system employed equation (1) [1] to calculate and get the best representative domain concepts from them, which are the maximum of information content value:

$$\text{DomainConcept} = \max_{c \in c_c} \left[ 1 - \log{P(c)} \right]$$

Furthermore, the system based on the property “hasURI” of the best representative domain concepts for indexing the correct location of the domain concept in WordNet and used Jaccard similarity to calculate the consistency of the domain concept, and accordingly stored in the corresponding property “hasConsistency” to finish the assimilation and integration of the domain ontology. Finally, the system could access the concept to support system operations according to its Synset_ID in WordNet.

G. Architecture of Ubi-Interface Agent and Interaction

Diagram with OntoIAS

Users can employ Ubi-Interface Agent of OntoIAS to use the shell via related mobile equipments for coming to the goal of ubiquitous research of OntoIAS. Therefore, the interface agent has to provide the communication bridge between those mobile and wireless equipments and related web information systems, as shown in Figure 6, for satisfying basic requirements of seamless information services in ubiquitous computing, whose related interaction diagrams contain: users keyin specific information requirement to trigger OntoIAS for returning query information, users directly query OntoIAS to provide in common use of hot information, users enter proper URL of ontology website to directly enhance the robustness of backend ontology of OntoIAS, etc.

![Figure 6. System architecture of Ubi-Interface agent](image)
In short, we simplify the design of Ubi-Interface Agent into the data decoding control of related communication equipments; and then employed CURRL to transform them into internal canonical format; finally, triggered OntoIAS to provide information solutions. The interaction diagram can not only solve the congenital defects of above mobile equipments, but also adequately elaborate the powerful functions of backend system OntoIAS.

### TABLE II. Frame Structure of CURRL

<table>
<thead>
<tr>
<th>Slot</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theme</td>
<td>Topic of user's command</td>
</tr>
<tr>
<td>Content</td>
<td>Relevant content of command</td>
</tr>
<tr>
<td>Time</td>
<td>Related time of topic</td>
</tr>
<tr>
<td>Space</td>
<td>Related space of object</td>
</tr>
</tbody>
</table>

### TABLE III. Examples on User Commands with CURRL

<table>
<thead>
<tr>
<th>USER COMMAND</th>
<th>CURRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is Intel CPU?</td>
<td>Theme = Intel, Space = WWW, tSpace = At (WWW), Command</td>
</tr>
<tr>
<td>Anything else?</td>
<td>tSpace = At (WWW)</td>
</tr>
<tr>
<td>Retrieving CPU webpage with exception of Mac</td>
<td>Command</td>
</tr>
</tbody>
</table>

### III. CONCLUSION AND DISCUSSION

The system prototype of OntoIAS had been developed, shown in Figure 7, which created the interaction diagram with both solving the congenital defects problems of mobile equipments and adequately elaborating the powerful functions of backend information systems, and then can effectively support the basic information requirement to users in ubiquitous environments. This diagram not only regards as the foundation stone of construction of ubiquitous information systems, but also takes for the consultation example of related industries of intelligent life space. The preliminary experimental outcomes also can show forth on basic efficiencies of related sub-systems, detailed in [8,9,10,12].

![Figure 7. System prototype of OntoIAS](image)

In conclusion, OntoIAS system prototype manifests the following interesting features. First, OntoCrawler can offer more precision and recall rate than Google and Yahoo on webpage searching. Second, OntoExtractor employed ontology and related services to support keywords trimming and conflict resolution, and the webpages are stored in an ontology-directed internal format, i.e., OD, which supports semantics-constrained retrieval of webpages. Third, OntoClassifier not only has a higher degree of classification precision but also shows the two-stage classifier we proposed has its availability and better system performance indeed. Finally, not only can OntoIAS fast search, extract, classify, and integrate specific domain documents, but also it can precisely recommend important information from them to take excellent information integration and recommendation ranking. In the future, we will face how to improve the efficiency of OntoIAS, for example, combining with the data mining technique [11] to predict the user information requirement for reducing the system processing and users waiting time; furthermore, integrating with the case-based reasoning mechanism [11] to speed up the system processing time, which is under our investigation; finally, to evaluate how well Ubi-Interface Agent works would be the everlasting research in the future.

Acknowledgments

The author would like to thank Sau-Hsien Lu, Ting-An Chen, Chi-Feng Wu and Hung-Chun Chiang for their assistance in related sub-system study and development in advance. This partial work was supported by the National Science Council, R.O.C., under Grants NSC-98-2221-E-129-012.

### REFERENCES


