A Study on Developing an Ontology-Supported Information Agent Shell

Sheng-Yuan Yang[1] **Chun-Liang Hsu**[2] **Yu-Chun Chu**[3] **Kuan-Wei Wu**[4]
[1]Dept. of Computer and Communication Engineering, St. John's University
[2]Dept. of Electrical Engineering, St. John's University
[3]Dept. of Information Management, National Taiwan University of Science and Technology
[4]Dept. of Electronic Engineering, St. John's University
Email:[1]ysy@mail.sju.edu.tw

Abstract- *This paper developed an Ontology-supported Information Agent Shell (OntoIAS) based on the ontology and information agent technologies. It contains the four main modules, including information crawling, information extracting, information classifying, and information presenting/ranking and orderly corresponding to OntoCrawler, OntoExtractor, OntoClassifier, and OntoRecommander, respectively. The preliminary experiment outcomes proved the technology proposed in this paper to be able to really reach the goal of an information agent, and accordingly proved the feasibility of this shell architecture.*

Keywords: Ontology, Information Agent Shell.

1. Introduction

"*Information as a concept has a diversity of meanings, from everyday usage to technical settings. Generally speaking, the concept of information is closely related to notions of constraint, communication, control, data, form, instruction, knowledge, meaning, mental stimulus, pattern, perception, and representation.*" (http://en.wikipedia.org/wiki/Information) With increasing popularity of the Internet, people resort more to the Web to obtain their information. A variety of query systems have thus appeared and become ever important since they can help people to effectively use this voluminous repository, for example, search engines, information portals, and even information agents. "*An agent is either an entity who is capable of action or someone (or something) who acts on behalf of another person.*" (http://en.wikipedia.org/wiki/Agents) Information agent is a software product for assisting and guiding users to reach the goal of information retrieval. The four main modules of an information agent are information crawling/searching, information extracting, information classifying, and information presentation/ranking. Until to now, most of Web information agent systems are closely related to the domain knowledge that resulting in the core part of information agent, i.e. the four main modules described before, cannot directly apply to Web systems in ubiquitous environments. In the 1980s, expert system "shells" were introduced (including one based on MYCIN - a medicine expert system developed in the early 1970s at Stanford University, known as E-MYCIN) and supported the development of expert systems in a wide variety of application areas. Thus, how to conduct an information agent shell that can effectively search, extract, classify and ranking data and information from Internet with intelligent techniques in ubiquitous environments has become important research issues.

We notice that ontology is mostly used in the systems that work on information gathering or integration to improve their gathering processes or the search results from disparate resources [3]. For instance, Intelligent Web search agent [7] distills and aggregates information found in HTML documents. By using web page ontology and web search agent ontology, it reduces the need for a human being to look at each hit to determine its relevance. Chen and Soo [1] describe an ontology-based information gathering agent which utilizes the domain ontology and corresponding support (e.g., procedure attachments, parsers, wrappers and integration rules) to gather the information related to users' queries from disparate information resources in order to provide much more coherent results for the users. Swoogle [2] is a crawler-based system that discovers, retrieves, analyzes and indexes knowledge encoded in semantic web documents on the Web, which can use either character N-Gram or URIrefs as keywords to find relevant documents and to compute the similarity among a set of documents. Finally, in [4] the authors design a retrieval system for searching OWL/RDF(s) domain ontology to maximize the reusability of knowledge information, which consists of Crawler, Classifying module, Ranking module, and Retrieval module.

In this paper, we developed an Ontology-supported Information Agent Shell (OntoIAS) based on the technologies described before, shown in Fig. 1. It contains the four main modules of information agents, including information crawling, information extracting, information classifying, and information presenting/ranking and orderly corresponding to OntoCrawler, OntoExtractor, OntoClassifier, and OntoRecommander, respectively. The reason of the beginning word “Onto-“ is all of module functions supported by ontology, which means the information agent shell is the core part of information agent separated from the domain ontology. 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 fusion processing from other related domain ontologies. Ontology Database is the key component, which stores both domain ontology and query ontology. 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. The approach not only can provide the basic operation semantic to the shell, but also can make the shell fast and precisely access information based on those semantic understanding. User Interface is responsible for singling out significant keywords from the user queries. Specifically, we use ontology, which defines the physical meanings of the important concepts involved in a given task, to successfully help the user interface develop a user profile to provide the personality functionality and store in User Profile Database through OntoRecommander. The Scholar webpages domain is chosen as the target application of the proposed system and will be used for explanation in the remaining sections.

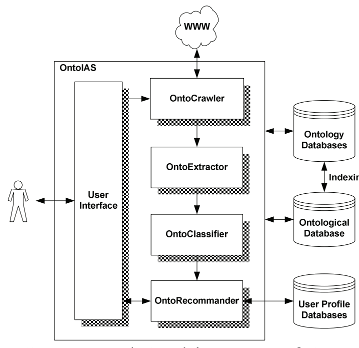
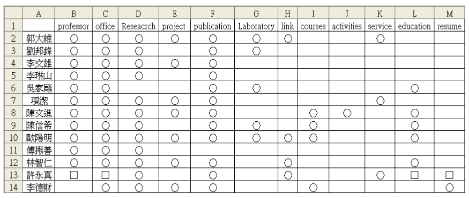


Fig. 1 Conceptual Architecture of OntoIAS

2. Construction of Ontology Database

Our ontology is a knowledge sharing database which was constructed for specific domain. That is to say in which took advantage of built ontology database of some scholars to support whole system operation, for example, serve OntoCrawler for querying webpage of related scholars, assist OntoClassifier in processing of webpage classification, etc. In the mentioned ontology database, it included two constructed stages; one is statistics and analysis of related concepts of scholars, the other is construction of ontology database. We detailed the procedures as below.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1		professor	office	Reseacrch	project	publication	Laboratory	link	courses	activities	service	education	resume
2	郭大維	○	○	○	○	○	○	○			○		
3	劉邦鋒	○	○	○		○	○						
4	李文雄	○	○	○	○	○							
5	李琳山	○	○	○		○							
6	吳家麟	○	○			○	○					○	
7	項潔	○	○	○	○	○					○		
8	陳文進	○	○	○	○	○			○	○		○	
9	陳信希	○	○	○		○	○		○			○	
10	歐陽明	○	○	○	○	○	○	○	○			○	
11	傅楸善	○	○	○									
12	林智仁	○	○	○	○	○		○				○	
13	許永真	□	□	○		○		○			○	□	□
14	李德財		○	○	○	○			○				○

Fig. 2 Survey table of scholars’ webpage of National Taiwan University

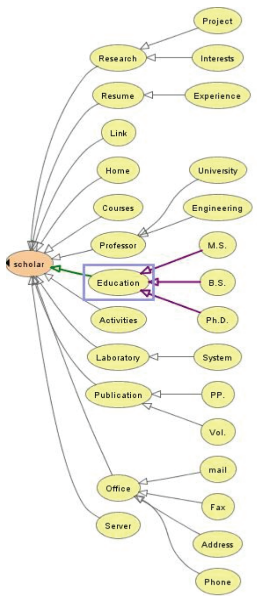


Fig. 3 The ontology structure of Scholars

First of all, we conducted statistics and survey of homepage of related scholars to fetch out the related concepts and their synonym appearing in the homepage. In Fig. 2, illustrated statistics and analysis of 50 datum about related concepts from webpage of scholars in National Taiwan University. The symbol “○” stood for this concept definitely appearing in the homepage; the symbol “□” stood for the concept appearing the next hyperlink-page. Fig. 3 indicated the structure of domain ontology of scholars in Protégé, taking the middle frame of the screen for instance, the related concept “Education” was linking behind with “M.S”, “PH.D”, and “B.S”. In application, we defined those as related concepts and that means “education” is nothing but an combination of these related concepts that would be conveniently interpreted by OntoCrawler to compare with content of the queried webpage, and if the compared outcomes were corresponding to any item among the four, we would infer the related concept "Education" as matched condition for web page querying.

Fig. 4 shows the second stage of ontology constructing of Scholars, in which the main part work is to transfer the ontology built with Protégé 2000 (http://protege.stanford.edu/) into MS SQL database. The procedures are as following:

(1) Exporting an XML file constructed with Protégé knowledge base and then importing into MS Excel for correcting. That was the strong evidence of knowledge reusing and fast embedding within Protégé.
(2) Finally importing MS Excel into MS SQL Sever to finish the ontology construction of this system.

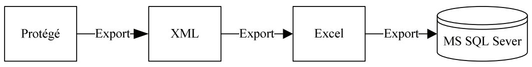


Fig. 4 Ontology database transferring procedures of scholars

3. Structure of OntoCrawler

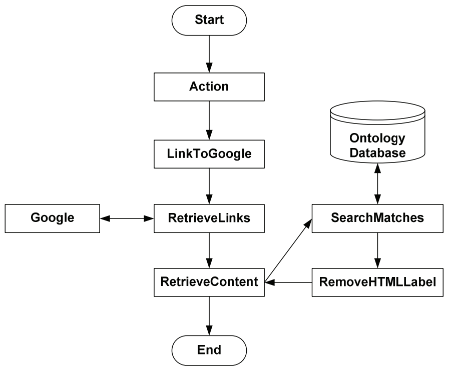


Fig. 5 System structure of OntoCrawler

Fig. 5 showed the operation system structure of OntoCrawler [11], and related techniques and functions of every part were described as below.

(1) *Action*: to transfer internal query into URI code, and then embed into Google's query URL: an example as follow http://www.google.com.tw/search?hl=zh-TW&q=%E5%AD%B8%E8%80%85&meta=.
(2) *LinkToGoogle*: to declare an URL object and add Google query URL on well transferred URI code, and then used a BufferedReader to read and used while loop to add String variable "line" line by line. Finally, output "line" as text file as final analysis reference. The file content was the html source file of the webpage.
(3) *RetrieveLinks*: to use regular expression to search for whether there are matched URL. But it couldn't retrieve all the linkages in one time out because of the Google webpage editing with indenting. So we used a "for" loop and ran for twice. The semantic of the two in regular expression were slightly different so as to completely fetch out related hyperlinks corresponding to the conditions. Finally, added all hyperlinks on String variable "RetrieveLink" and output the txt file to provide the system for further processing.
(4) *RetrieveContent*: to use BufferedReader (Java Class) to read in "RetrieveLink" with "while" loop line by line, that meant we checked one URL link once a time and really linked the URL. After judging what kind coding of the webpage was, we read in the html source file of webpage with correct coding added on String variable s3 and output it as text file so as to let system conduct further processing. After completing all procedures mentioned above, we could used *SearchMatches* method (described later) to judge whether the webpage was located in the range we hoped to query; supposed the answer was "yes", we would execute *RemoveHTMLLabel* (described later) to delete the html label from source file and remained only the text content so as to let system conduct further processing and analyzing. Finally, we collected the number of queried webpage and divided with total of the webpage and the mean we got was the percentage of query processing. Remember to clear RetrieveLink lest the next query should cause errors.
(5) *SearchMatches*: supporting *RetrieveContent* internal calling service to judge whether the webpage was the range we queried. It linked the ontology database and fetched out the content to compare content of s3 when using this linking method. If there were any return value corresponding to the value we set, and then system would return one "true" Boolean variable "matches." That meant the webpage matched our query condition, on the other hand, if returned "false" meant the webpage didn't match our query condition.
(6) *RemoveHTMLLabel*: just like *SearchMatches*, it supported *RetrieveContent* internal calling service and deleted html label in the html source file.

4. Structure of OntoExtractor

Fig. 9 shows the structure of OntoExtracter. Webpage Parser deletes unnecessary spaces and segments the words in the webpages using MMSEG (http://technology.chtsai.org/mmseg/). The results of MMSEG segmentation were bad, for the predefined MMSEG word corpus contains insufficient terms of the Scholar domain. We easily fixed this by using Ontology Database as a second word corpus to bring those mis-segmented words back. Keyword Extractor is responsible for building canonical keyword indices for Webpages. 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. Ontology techniques used here include employing ontology synonyms to delete redundant data, utilizing the features of ontology concepts to restore missing data, and exploiting the value constraints of ontology concepts to resolve inconsistency. 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.

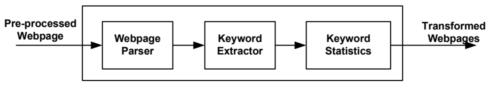


Fig. 9 Structure of OntoExtractor

5. Structure of OntoClassifier

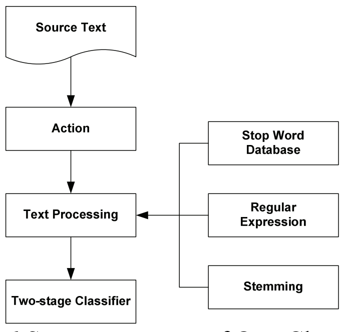


Fig. 6 System structure of OntoClassifier

Fig. 6 illustrated the system operational structure and flowchart of OntoClassifier. The Source Text originated from OntoExtractor. The rest functions and related techniques were described as below.

(1) Action: input file format was *.txt that was pure text of webpage after excluding labels and fixing mis-segmentation and inconsistency. The initial task included as following: loading Stop Word database, ontology database, and SourceText document for further processing.

(2) Text Processing:

  (a) Regular Expression: to load SourceText and formalize database array, and then search SourceText to replace every non-expression word with space character till the end of SourceText.

  (b) Filter Stop words: stop words or noise words were words that often appeared in document but have no contribution for reading whole document such as “is”, “of”, and “the” etc. we used “stop list” to store those words and ought to be excluded when indexing words and phrases so as to decrease the noise in document and increase classification precision.

  (c) Stemming: stem transformation means transfer different word type into stem for example, “connection”, “connections”, “connective”, “connected”, “connecting” would be all transferred as “connect” to arise the extracting precision [9]. In practical processing, we could use Java API JWNL provided by WordNet (http://jwordnet.sourceforge.net/) to accomplish it.

(3) Two-stage classifier: classifies a webpage in two stages. The first stage uses representative ontology features. We employ the level threshold to limit the number of ontology features to be involved in this stage. The basic idea of classification of the first stage is defined by Eq. (1). In the equation, *OntoMatch(d,C)* is defined by Eq. (2), which calculates the number of ontological features of class *C* that appears in webpage *d*, where *M(w,c)* returns 1 if word *w* of *d* is contained in class *C*. Thus, Eq. (1) returns class *C* for webpage *d* if *C* has the largest number of ontology features appearing in *d*. Note that not all classes have the same number of ontology features; we have added $\#w_{C'}$, the number of words in each class *C'*, for normalization. Also note that Eq. (1) only compares those classes with more than three ontology features appearing in *d*, i.e., it filters less possible classes. As to why classes with less than three features appearing in *d* are filtered, we refer to Joachims' concept that the classification process only has to consider the term with appearance frequency larger than three [5].

$$HOntoMatch(d) = \arg\max_{C' \in c} \frac{OntoMatch(d,C')}{\# w_{C'}}, \quad \underset{C' \in c}{OntoMatch}(d,C') > 3 \qquad (1)$$

$$OntoMatch\ (d,C) = \sum_{w \in d} M(w,C) \qquad (2)$$

If for any reason the first stage cannot return a class for a webpage, we move to the second stage of classification. The second stage no longer uses level thresholds but gives an ontology term a proper weight according to which level it is associated with. That is, we modify the traditional classifiers by including a level-related weighting mechanism for the ontology concepts to form our ontology-based classifier. This level-related weighting (*L*) mechanism will give a higher weight to the representative features than to the related features. The second stage of classification is defined by Eq. (3). Inside the equation, *OntoTFIDF(d,C)* is defined by Eq. (4), which is basically the calculation of a *TFIDF* score on the ontology features of class *C* with respect to webpage *d*, where *TF(x|y)* means the number of appearance of word *x* in *y*. Thus, Eq. (3) returns class *C* for webpage *d* if *C* has the highest score of *TFIDF* with respect to.

$$HOntoTFIDF(d) = \arg\max_{C' \in c} OntoTFIDF(d, C') \tag{3}$$

$$OntoTFIDF(d, C) = \sum_{w \in d} \frac{1}{L_w} \frac{TF(w|C)}{\sum_{w' \in F_C} TF(w'|C)} \times \frac{TF(w|d)}{\sum_{w' \in F_C} TF(w'|d)} \tag{4}$$

6. Approach of OntoRecommander

We finally apply different ranking methods to rank and present the retrieval results according to whether full keywords match or partial keywords match is applied.

6.1 Ranking method for Full Keywords Match

If only one webpage can be located in OD under full keywords match, OntoRecommander will directly output the webpage to the user. If more than one, say *N*, is retrieved, it employs Eq. (5) to calculate a match score (*MS*) for each webpage, i.e., WP.

$$MS(WP_i) = W_{AP} \times \frac{AP_i}{Max(AP_1 \ldots AP_N)} + W_{SV} \times \frac{SV_i}{Max(SV_1 \ldots SV_N)} \tag{5}$$

where AP_i is Appearance Probability and SV_i means Satisfaction Value of WP_i. Weight factors W_{AP} and W_{SV} are set to 0.6 and 0.4, respectively [12]. Eq. (6) and (7), in turn, define AP_i.

$$AP_i = \prod_{j=1}^{n} P(k_{i,j}) \tag{6}$$

$$P(k_{i,j}) = \begin{cases} 1, & if \quad k_{i,j} \in user's \quad query \\ \dfrac{\#k_i}{N}, & otherwise \end{cases} \tag{7}$$

where $k_{i,j}$ represents the *jth* keyword of WP_i; $\#k_i$ is the number of keywords in WP_i.

We use Eq. (8) to calculate SV_i.

$$SV_i = \frac{\sum_{m=1}^{n} USL_m \times UPL_m}{\sum_{m=1}^{n} Max(USL_1 \ldots USL_n) \times Max(UPL_1 \ldots UPL_n)} - (0.1 \times IA) \tag{8}$$

where USL_m represents the user satisfaction level of the *mth* feedback, which takes on one of the five predefined user feedback levels, namely, *highly satisfied, satisfied, normal, unsatisfied,* and *highly unsatisfied* with corresponding scores 5, 4, 3, 2, and 1; UPL_m stands for the user proficiency level of the *mth* user feedback, which takes on one of the five predefined user levels [10,13], namely, *expert, senior, junior, novice*, and *amateur* with corresponding weights 5, 4, 3, 2, and 1; *n* is the total number of feedbacks to WP_i; *IA* stands for Aging Index with an initial value of zero. Note that OntoRecommander employs *IA* to record how aged a webpage is in order to track the hot topics. It increases or decreases the *IA* of a webpage according to the user feedback. Thus, if a webpage receives no feedback over seven days, it will increase its *IA*, signifying the aging process. On the other hand, if a webpage's *USL* multiplied by the user's *UPL* is larger than 9, implying the user's *SV* is better than a junior user can give, OntoRecommander will decrease its *IA*, which in turn raises its *SV*, signifying the anti-aging process.

6.2 Ranking method for Partial Keywords Match

In the partial keywords match method, we calculate match scores for retained WPs according to Eq. (9).

$$MS(WP_i) = W_{CV} \times \frac{CV_i}{Max(CV_1 \ldots CV_N)} + W_{SSV} \times \frac{SSV_i}{Max(SSV_1 \ldots SSV_N)}) + W_{CR} \times \frac{CR_i}{Max(CR_1 \ldots CR_N)} + W_{SV} \times \frac{SV_i}{Max(SV_1 \ldots SV_N)}) \tag{9}$$

where SV_i is the same as in Eq. (4) and SSV_i stands for Statistic Similarity Value of WP_i, which calculates the inner product of the two-keyword vectors according to the Vector Space Model [8]. Eq. (10) defines CV_i as Compatibility Value and Eq. (11) defines CR_i as Coverage Ratio for WP_i.

$$CV_i = \frac{C(T_{i,q}, T_{i,f})}{|T_{i,q}| \times |T_{i,f}|} \text{ with } C(T_{i,q}, T_{i,f}) = \sum_{q_k \in T_{i,q}, f_j \in T_{i,f}} c(q_k, f_j) \text{ and}$$

$$c(q_k, f_j) = \begin{cases} 1, & q_k \quad compatible \quad with \quad f_j \\ 0, & else \end{cases} \tag{10}$$

where $T_{i,q}$ contains unmatched keywords in WP_i, while $T_{i,f}$ contains unmatched keywords in the user query. Function *c*(q_k, f_j) checks for compatibility and is supported by the ontology services, which check whether the two keywords are related with conflicting constraints. If yes, it returns 0; otherwise, it returns 1.

$$CR_i = \frac{\sum_{q_k \in K_{i,q}, f_j \in K_{i,f}} E(q_k, f_j)}{|K_{i,f}|} \text{ with } E(q_k, f_j) = \begin{cases} 1, & if \quad q_k = f_j \\ 0, & else \end{cases} \tag{11}$$

where $K_{i,f}$ contains the keywords in WP_i. Function *E*(q_k, f_j) checks for syntactical equality between keyword q_k and keyword f_j syntactically.

7. Conclusions

We had developed an Ontology-supported

Information Agent Shell (OntoIAS) based on the ontology and information agent technologies. It contains the four main modules, including information crawling, information extracting, information classifying, and information presenting/ranking and orderly corresponding to OntoCrawler, OntoExtractor, OntoClassifier, and OntoRecommander, respectively. We expect the techniques can properly tackle the issues of how to conduct an information agent shell that can effectively search, extract, classify and ranking data and information from Internet with intelligent techniques in ubiquitous environments. To substantiate this expectation, an overall system evaluation is necessary, which, however, is both difficult and time-consuming. To help us gather this confidence in a shorter period, we have carefully focused our experiments on the evaluation of performance of the key components in the system. Our first experiment is to learn how well OntoCrawler could really up-rise the precision rate of webpage searching up to around 90% [11]. The second experiment is to learn how well OntoExtractor with ontology supports keywords trimming and conflict resolution from 5 to 20% improvement in precision rate and accordingly produces better ranking results [12]. The third experiment is to learn how well OntoClassifier could really up-rise the precision rate of webpage classification up to around 92% [6]. Those preliminary experiment outcomes proved the technology proposed in this paper to be able to really reach the goal of an information agent, and accordingly proved the feasibility of this shell architecture.

Acknowledgement

The authors would like to thank T.A. Chen, H.C. Chiang, and C.F. Wu for their assistance in system implementation and experiments.

References

[1] Y.J. Chen and V.W. Soo, "Ontology-Based Information Gathering Agents," *The 2001 International Conference on Web Intelligence*, Maebashi TERRSA, Japan, 2001, pp. 423-427.

[2] L. Ding, T. Finin, A. Joshi, R. Pan, R.S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs, "Swoogle: A Search and Metadata Engine for the Semantic Web," *Proc. of the 13th ACM International Conference on Information and Knowledge Management*, Washington, USA, 2004, pp. 652-659.

[3] D. Eichmann, "Automated Categorization of Web Resources," Available at http://www.public.iastate.edu/~CYBERSTACKS/Aristotle.htm, 1998.

[4] M.G. Hwang, H.J. Kong, and P.K. Kim, "The Design of the Ontology Retrieval System on the Web," *Proc. of the 8th International Conference on Advanced Communication Technology*, Gwangju, South Korea, 2006, pp. 1815-1818.

[5] T. Joachims, "A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization," *Proc. of the 14th International Conference on Machine Learning*, Nashville, USA, 1997, pp. 143-151.

[6] D.L. Lee, S.Y. Yang, and C.L. Hsu, "Ontology-Supported Webpage Classifier for Scholar's Webpages in Ubiquitous Information Environment," *Proc. of The First IEEE International Conference on Ubi-media Computing*, Lanzhou, China, 2008, pp. 523-528.

[7] T.K. Plunkett and D. Thompson, "Intelligent Web Search Agents," *Encyclopedia of Library and Information Science*, 67(30), 2000, pp. 261-262.

[8] G. Salton, and M.J. McGill, *Introduction to Modern Information Retrieval*, McGraw-Hill Book Company, New York, USA, 1983.

[9] S.Y. Yang, Y.C. Chu, and C.S. Ho, "Ontology-Based Solution Integration for Internet FAQ Systems," *Proc. of the 6th Conference on Artificial Intelligence and Applications*, Kaohsiung, Taiwan, 2001, pp. 52-57.

[10] S.Y. Yang, "FAQ-master: A New Intelligent Web Information Aggregation System," *International Academic Conference 2006 Special Session on Artificial Intelligence Theory and Application*, Tao-Yuan, Taiwan, 2006, pp. 2-12.

[11] S.Y. Yang, T.A. Chen, and C.F. Wu, "Ontology-Supported Focused Crawler for Scholar's Webpage," *Proc. of 2008 International Conference on Advanced Information Technology*, TaiChung, Taiwan, 2008, pp. 55.

[12] S.Y. Yang, "Developing an Ontological FAQ System with FAQ Processing and Ranking Techniques for Ubiquitous Services," *Proc. of The First IEEE International Conference on Ubi-media Computing*, Lanzhou, China, 2008, pp. 541-546.

[13] S.Y. Yang and C.S. Ho, "Ontology-Supported User Models for Interface Agents," *Proc. of the 4th Conference on Artificial Intelligence and Applications*, Chang-Hua, Taiwan, 1999, pp. 248-253.