An ontological website models-supported search agent for web services

Sheng-Yuan Yang *

Department of Computer and Communication Engineering, St. John’s University, Taipei, 499, Sec. 4, TamKing Road, Tamsui, Taipei County 25135, Taiwan, ROC

Abstract

In this paper, we advocate the use of ontology-supported website models to provide a semantic level solution for a search engine so that it can provide fast, precise and stable search results with a high degree of user satisfaction. A website model contains a website profile along with a set of webpage profiles. The former remembers the basic information of a website, while the latter contains the basic information, statistics information, and ontology information about each webpage stored in the website. Based on the concept, we have developed a Search Agent which manifests the following interesting features: (1) Ontology-supported construction of website models, by which we can attribute correct domain semantics into the Web resources collected in the website models. One important technique used here is ontology-supported classification (OntoClassifier). Our experiments show that the OntoClassifier performs very well in obtaining accurate and stable webpages classification to support correct annotation of domain semantics. (2) Website models-supported Website model expansion, by which we can collect Web resources based on both user interests and domain specificity. The core technique here is a Focused Crawler which employs progressive strategies to do user query-driven webpage expansion, autonomous website expansion, and query results exploitation to effectively expand the website models. (3) Website models-supported Webpage Retrieval, by which we can leverage the power of ontology features as a fast index structure to locate most-needed webpages for the user.

Keywords: Ontology; Website models; Search agents; Web services

1. Introduction

In this information-exploding era, the user expects to spend short time retrieving really useful information rather than spending plenty of time and ending up with lots of garbage information. Current general search engines, however, produce so many entries of data that often overwhelms the user. For example, Fig. 1 shows six of over 4000 entries returned from Google, a well-known robot-based search engine, for a query that asks for assembling a cheaper and practical computer. The user usually gets frustrated after a series of visits on these entries, however, when he discovers that dead entries are everywhere, irrelevant entries are equally abundant, what he gets is not exactly what he wants, etc. It is increasing clear that one has to properly narrow down the scope of search and to cleverly sort the sites or combine them in order to benefit much from the Internet. This process is hard to general users, however; so is it to the experts. Two major factors are behind this difficulty. First, the Web is huge; it’s reported that six major public search engines (AltaVista, Excite, HotBot, Infoseek, Lycos, and Northern Light) collectively only covered about 60% of the Web and the largest coverage of a single engine was about one-third of the estimated total size of the Web (Lawewnce & Giles, 2000). Empirical study also indicates that no single search engine could return more than 45% of the relevant results (Selberg & Etzioni, 1995). Second, most pre-defined index structures used by the search engines are complex and inflexible. Lawewnce and Giles (1999) reports that indexing of new or modified pages in one of the major search engines could take months. Even if we can follow the well-organized Web directory structure of one search engine, it is still very easy that we get lost in the complex category
indexing jungle (see Fig. 2 for an example). Current domain-specific search engines do help users to narrow down the search scope by the techniques of query expansion, automatic classification and focused crawling; their weakness, however, is almost completely ignoring the user interests (Wang, 2003).

In general, current search engines face two fundamental problems. First, the index structures are usually very different from what the user conjectures about his problems. Second, the classification/clustering mechanisms for data hardly reflect the physical meanings of the domain concepts. These problems stem from a more fundamental problem: lack of semantic understanding of Web documents. New standards for representing website documents, including XML (Henry, David, Murray, & Noah, 2001), RDF (Brickley & Guha, 2004), DOM (Arnaud et al., 2004), Dublin metatag (Weibel, 1999), and WOM (Manola, 1998), can help cross-reference of Web documents; they alone, however, cannot help the user in any semantic level during the searching of website information. OIL (2000), DAML (2003), DAML+OIL (2001), and the concept of ontology stand for a possible rescue to the attribution of information semantics. In this paper, we advocate the use of ontology-supported website models (Yang, 2004).
to provide a semantic level solution for a search engine so that it can provide fast, precise and stable search results with a high degree of user satisfaction.

Basically, a website model consists of a website profile for a website and a set of webpage profiles for the webpages contained in the website. Each webpage profile reflecting a webpage describes how the webpage is interpreted by the domain ontology, while a website profile describes how a website is interpreted by the semantics of the contained webpages. The website models are closely connected to the domain ontology, which supports the following functions used in website model construction and application: query expansion, webpage annotation, webpage/website classifications (Yang, 2006), and focused collection of domain-related and user-interested Web resources (Yang, 2006). We have developed a Search Agent using website models as the core technique, which helps the agent successfully tackle the problems of search scope and user interests. Our experiments show that the Search Agent can locate, integrate and update both domain-related and user-interested Web resources in the website models for ready retrieval.

The personal computer (PC) domain is chosen as the target application of our Search Agent and will be used for explanation in the remaining sections. The rest of the paper is organized as follows. Section 2 develops the domain ontology. Section 3 describes Website models and how they are constructed. Section 4 illustrates how Website models can be used to do better Web search. Section 5 describes the design of our search agent and reports how it performs. Section 6 discusses related works, while Section 7 concludes the work.

2. Domain ontology as the first principles

Ontology is a method of conceptualization on a specific domain (Noy & Hafner, 1997). It plays diverse roles in developing intelligent systems, for example, knowledge sharing and reusing (Decker et al., 2000, 1998), semantic analysis of languages (Moldovan & Mihalcea, 2000), etc. Development of an ontology for a specific domain is not yet an engineering process, but it is clear that an ontology must include descriptions of explicit concepts and their relationships of a specific domain (Ashish & Knoblock, 1997). We have outlined a principle construction procedure in Yang and Ho (1999); following the procedure we have developed an ontology for the PC domain. Fig. 3 shows part of the PC ontology taxonomy. The taxonomy represents relevant PC concepts as classes and their parent–child relationships as isa links, which allow inheritance of features from parent classes to child classes. We then carefully selected those properties of each concept that are most related to our application and used them as attributes to define the corresponding class. Fig. 4 exemplifies the definition of ontology class “CPU”. In the figure, the uppermost node uses various fields to define the semantics of the CPU class, each field representing an attribute of “CPU”, e.g., interface, provider, synonym, etc. The nodes at the bottom level represent various CPU instances that capture real world data. The arrow line with term “io” means the instance of relationship. Our ontology construction tool is Protégé 2000 (Noy & McGuinness, 2001) and the complete PC ontology can be referenced from the Protégé Ontology Library at Stanford Website (http://protege.stanford.edu/ontologies.html). Fig. 5 demonstrates how the ontology looks like on Protégé 2000, where the left column represents the taxonomy hierarchically and the right column contains respective attributes for a specific class node selected. The example shows that the CPU ontology contains synonyms, along with a bunch of attributes and constraints on their values. Although the domain ontology was developed in Chinese (but was changed to English here for easy explanation) corresponding English names are treated as Synonyms and can be processed by our system too.

In order to facilitate Web search, the domain ontology was carefully pre-analyzed with respect to how attributes are shared among different classes and then re-organized into Fig. 6. Each square node in the figure contains a set of representative ontology features for a specific class, while each oval node contains related ontology features between two classes. The latter represents a new node type called “related concept”. We select representative ontology features for a specific class by first deriving a set of
candidate terms from pre-selected training webpages belonging to the class. We then compare them with the attributes of the corresponding ontology class; those candidate terms that also appear in the ontology are singled out and dubbed as the representative ontology features of the class. The representative features are then removed from the set of candidate terms and the rest of candidate terms are compared with the attributes of other ontology classes. For any other ontology class that contains some of these candidate terms, we add a related concept node to relate it to the class. Fig. 7 takes CPU and motherboard as two example classes and show how their related concept node looks like. The figure shows a related concept node between two classes; in fact, we may have related concept nodes among three or more classes too. For instance in Fig. 6, we have a related concept node that relates CPU, motherboard and SCSI Card together. Table 1 illustrates related concept nodes of different levels, where level n means the related concept node relates n classes together. Under this definition, level 1 reduces to the representative features of a specific class. Thus, term “graphi” in level 3 means it appears in three classes: graphic card, monitor, and motherboard. This design clearly structures semantics between ontology classes and their relationships and can facilitate the building of semantics-directed website models to support Web search.
3. Website model and construction

A website model contains a website profile and a set of webpage profiles. A website profile contains statistics about a website. A webpage profile contains basic information, statistics information, and ontology information about a webpage. To produce the above information for website modeling, we use DocExtractor to extract primitive webpage information as well as to perform statistics. It also transforms the original webpage into a tag-free document and turns it to OntoAnnotator to annotate ontology information. This section gives detailed description of what a website model looks like and how it is constructed.

3.1. Website model

Fig. 8(a) illustrates the format of a website model. The webpage profile contains three sections, namely, basic information, statistics information, and ontology information. The first two sections profile a webpage and the last annotates domain semantics to the webpage.

Each field in the basic information section is explained below. DocNo is automatically generated by the system for identifying a webpage in the structure index. Location remembers the path of the stored version of the Web page in the website model; we can use it to answer user queries. URL is the path of the webpage on the Internet, same as the returned URL index in the user query result; it helps hyperlinks analysis. WebType identifies one of the following six Web types: com (1), net (2), edu (3), gov (4), org (5), and other (0), each encoded as an integer in the parentheses. WebNo identifies the website that contains this webpage. It is set to zero if we cannot decide what website the webpage comes from. Update_Time/Date remembers the webpage when the webpage was modified last time. The statistics information section stores statistics about HTML tag properties, e.g., #Frame for the number of frames, #Tag for the number of different tags, and various texts enclosed in tags. Specifically, we remember the texts associated with Titles, Anchors, and Headings for webpage analysis; we also record Outbound_URLs for user-oriented webpage expansion.

Finally, the ontology information section remembers how the webpage is interpreted by the domain ontology. It shows that a webpage can be classified into several classes with different scores of belief according to the ontology. It also remembers the ontology features of each class that appear in the webpage along with their term frequencies (i.e., number of appearance in the webpage). Domain_Mark is used to remember whether the webpage belongs to a specific domain; it is set to “true” if the webpage belongs to the domain, and “false” otherwise. This section annotates how a webpage is related to the domain and can serve as its semantics, which helps a lot in correct retrieval of webpages.

Let us turn to the website profile. WebNo identifies a website, the same as that used in the webpage profile. Through this number, we can access those webpage profiles describing the webpages that belong to this website. Website_Title remembers the text between tags (TITLE) of the homepage of the website. Start_URL stores the starting address of the website. It may be a domain name or a directory URL under the domain address. WebType identifies one of the six Web types as those used in the webpage profile. Tree_Level Limit remembers how the website is structured, which can keep the search agent from exploring too deeply, e.g., 5 means it explores at most 5 levels of the website structure. Update_Time/Date remembers when the website was modified last time. Fig. 8(b) illustrates an example website model. This model structure helps interpret the semantics of a website through the gathered information; it also helps fast retrieval of webpage information and autonomous Web resources search. The last point will become clearer later. Fig. 8(c) illustrates how website profiles and webpage profiles are structured.

3.2. Website modeling

Website modeling involves three modules. We use DocExtractor to extract basic webpage information and perform statistics. We then use OntoAnnotator to annotate ontology information. Since the ontology information contains webpage classes, OntoAnnotator needs to call OntoClassifier to perform webpage classification. Fig. 9 illustrates the architecture of DocExtractor. DocExtractor receives a webpage from DocPool and produces informa-
tion for both basic information and statistics information sections of a webpage profile. It also transforms the webpage into a list of words (pure text) for further processing by OntoAnnotator. Specifically, DocPool contains webpages retrieved from the Web. HTML Analyzer analyzes the HTML structure to extract URL, Title texts, anchor texts and heading texts, and to calculate tag-related statistics for website models. HTML TAG Filter removes HTML tags from the webpage, deletes stop words using 500 stop words we developed from McCallum (1996), and performs word stemming and standardization. Document Parser transforms the stemmed, tag-free webpage into a list of words for further processing by OntoAnnotator.

Fig. 10 illustrates the architecture of OntoAnnotator. Inside the architecture, OntoClassifier uses the ontology to classify a webpage, and Annotator uses the ontology to annotate ontology features with their term frequencies for each class according to how often they appear in the webpage. Domain Marker uses Table 2 to determine whether the webpage is relevant to the domain. The condition column in the table means the number of classes appearing in the webpage and the Limit column specifies a minimal threshold on the average number of features of
the class which must appear in the webpage. For example, row 2 means if a webpage contains only one class of a domain, then the features of the class appearing in the webpage must be greater than or equal to three in order for it to be considered to belong to the domain. In addition to classification of webpages, OntoClassifier is used to annotate each ontology class by generating a classification score for each ontology class. For instance, the annotation of \((\text{CPU}: 0.8, \text{Motherboard}: 0.5)\) means the webpage belongs to class CPU with score 0.8, while it belongs to class Motherboard with score 0.5.

OntoClassifier is a two-step classifier based on the re-organized ontology structure as shown in Figs. 6 and 7, where, for ease of explanation, we have deliberately skipped the information of term frequencies. Fig. 11 brings it back showing that each ontology feature is associated with a term frequency. The term frequency comes from the number of each feature appearing in the classified webpages.

OntoClassifier classifies a webpage in two stages. The first stage uses representative ontology features. We employ the level threshold, \(\text{THW}\) to limit the number of ontology features to be involved in this stage. For example, Fig. 11 shows \(\text{THW} = 1\), which means only the set of representative features at level 1 will be used. The basic idea of classification of the first stage is defined by Eq. (1). In the equation, \(\text{OntoMatch}(d, C)\) 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 (Joachims, 1997).

\[
\text{HOnToMatch}(d) = \arg \max_{C \in \text{OnToClass}} \frac{\text{OntoMatch}(d, C)}{\#w_c},
\]

\[
\text{OntoMatch}(d, C) > 3
\]

\[
\text{OntoMatch}(d, C) = \sum_{w \in d} M(w, C)
\]

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 feature 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 classes to form our ontology-based classifier.

---

**Table 2**

<table>
<thead>
<tr>
<th>Condition (class count)</th>
<th>Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>None</td>
</tr>
<tr>
<td>1</td>
<td>Average ≥ 3</td>
</tr>
<tr>
<td>2 ~ 6</td>
<td>Average ≥ 2</td>
</tr>
<tr>
<td>7 ~ 10</td>
<td>Average ≥ 1</td>
</tr>
</tbody>
</table>

---

Please cite this article in press as: Yang, S.-Y., An ontological website models-supported search agent for web services, *Expert Systems with Applications* (2007), doi:10.1016/j.eswa.2007.09.024
This level-related weighting 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 calculates a TFIDF score for each class \(C\) with respect to \(d\) and \(C\), where \(\text{TF}(x|y)\) means the number of appearance of word \(x\) in \(y\). Eq. (3) is used to create a list of class: score pairs for \(d\) and finally selects the one with the highest score of TFIDF as the class for webpage \(d\).

\[
\text{HOntoTFIDF}(d) = \arg \max_{C \in c} \text{OntoTFIDF}(d, C')
\]

\[
\text{OntoTFIDF}(d, C) = \sum_{w \in d} \frac{1}{L_w} \sum_{w' \in FC} \frac{\text{TF}(w'|C)}{\text{TF}(w'|C)} \times \frac{\text{TF}(w|d)}{\text{TF}(w|d)}
\]

(4)

4. Website models application

The basic goal of the website models is to help Web search taking account both user-interest and domain-dependence. Section 4.1 explains how this can be achieved. The second goal is to help fast retrieval of webpages stored in the website models for the user. Section 4.2 explains how this is done.

4.1. Focused web crawling supported by website models

In order to effectively use the website models to narrow down the search scope, we propose a new focused crawler as shown in Fig. 12, which features a progressive crawling strategy in obtaining domain relevant Web information.

Inside the architecture, Web Crawler gathers data from the Web. DocPool was mentioned before; it stores all returned Web pages from Web Crawler for DocExtractor during the construction of webpage profiles. It also stores query results from search engines, which usually contains a list of URLs. URLExtractor is responsible for extracting URLs from the query results and dispatching those URLs that are domain-dependent but not yet in the website models to Distiller. User-Oriented Webpage Expander pinpoints interesting URLs in the website models for further webpage expansion according to the user query. Autonomous Website Evolver autonomously discovers URLs in the website models that are domain-dependent for further webpage expansion. Since these two types of URLs are both derived from website models, we call them website model URLs in the figure. User Priority Queue stores the user search strings and the website model URLs from User-Oriented Webpage Expander. Website Priority Queue stores the website model URLs from Autonomous Website Evolver and the URLs extracted by URLExtractor.

Distiller controls the Web search by associating a priority score with each URL (or search string) using Eq. (5) and placing it in a proper Priority Queue. Eq. (5) defines ULScore\((U,F)\) as the priority score for each URL (or search string).

\[
\text{ULScore}(U,F) = W_F \times S_F(U)
\]

(5)

where \(U\) represents a URL or search string; and \(F\) identifies the way \(U\) is obtained as shown in Table 3, which also assigns to each \(F\) a weight \(W_F\) and a score \(S_F(U)\). Thus, if \(F = 1\), i.e., \(U\) is a search string, then \(W_1 = 3\), and \(S_1(U) = 100\), which implies all search strings are treated as the top-priority requests. As for \(F = 3\), if \(U\) is new to the website models, \(S_3(U)\) is set to 1 by URLExtractor; otherwise it is set to 0.5. Finally, for \(F = 2\), the URLs may come from User-Oriented Webpage Expander or Autonomous Website Evolver. In the former case, we follow the algorithm in Fig. 13 (to be explained in Section 4.1.1) to calculate \(S_2(U)\) for each \(U\). The assignment of
Table 3
Basic weighting for URLs to be explored

<table>
<thead>
<tr>
<th>Input type</th>
<th>F</th>
<th>W_F</th>
<th>S_d(U)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search strings</td>
<td>1</td>
<td>3</td>
<td>S_1(U) = 100</td>
</tr>
<tr>
<td>Website model URLs</td>
<td>2</td>
<td>2</td>
<td>S_2(U)</td>
</tr>
<tr>
<td>URLs extracted by URLExtractor</td>
<td>3</td>
<td>1</td>
<td>S_3(U)</td>
</tr>
</tbody>
</table>

S_d(U) in the latter case is more complicated; we will describe it in Section 4.1.2.

Note that Distiller schedules all the URLs (and search strings) in the User Priority Queue according to their priority scores for web crawling before it starts to schedule the URLs in the Website Priority Queue. In addition, whenever there are new URLs coming into the User Priority Queue, Distiller will stop the scheduling of the URLs in the Website Priority Queue and turn to schedule the new URLs in the User Priority Queue. This design prefers user-oriented Web resource crawling to website maintenance, since user-oriented query or webpage expansion takes into account both user interest and domain constraint, which can better meet our design goal than website maintenance.

4.1.1. User-oriented web search supported by website models

Conventional information retrieval research has mainly been based on (computer-readable) text (Salton & Buckley, 1988, 1983) to locate desired text documents using a query consisting of a number of keywords, very similar to the keyword-based search engines. Retrieved documents are ranked by relevance and presented to the user for further exploration. The main issue of this query model lies in the difficulty of query formulation and the inherent word ambiguity in natural language. To overcome this problem, we propose a direct query expansion mechanism, which helps users implicitly formulate their queries. This mechanism uses domain ontology to expand user query. One straightforward expansion is to add synonyms of terms contained in the user query into the same query. Synonyms can be easily retrieved from the ontology. Table 4 illustrates a simple example. More complicated expansion adds ontology classes according to their relationships with the query terms. The most used relationships follow the inheritance structure. For instance, if more than half of the subconcepts of a concept appear in a user query, we add the concept to the query too.

We also propose an implicit webpage expansion mechanism oriented to the user interest to better capture the user intention. This user-oriented webpage expansion mechanism adds webpages related to the user interest for further retrieval into the website models. Here we exploit the outbound hyperlinks of the stored webpages in the website models. To be more precise, we are using Anchor Texts specified by the webpage designer for the hyperlinks, which contain terms that the designer believes are most suitable to describe the hyperlinked webpages. We can compare these anchor texts against a given user query to determine whether the hyperlinked webpages contain the terms in the query. If yes, it implies the hyperlinked webpages might be interested by the user and should be collected in the website models for further query processing. Fig. 13 formalizes this idea into a strategy to select those hyperlinks, or URLs, that the users are strongly interested in. The algorithm returns a URL-list which contains hyperlinks along with their scores. The URL-list will be sent to the Focused Crawler (to be discussed later), which uses the scores to rank the URLs and, accordingly fetches hyperlinked webpages. Note that the algorithm uses R_s,d to modulate the scores, which decrease those hyperlinks that are less related to the target domain. R_s,d specifies the degree of domain correlation of website S with respect to domain D, as defined by Eq. (6). In the equation, N_s,d refers to the number of webpages on website S belonging to domain D; and N_s stands for the number of webpages on website S. Here we need the parameter Domain_Mark in the webpage profile to determine N_s,d. In short, R_s,d measures how strong a website is related to a domain. Thus, the autonomous webpage expansion mechanism is also domain-oriented in nature.

Table 4
Example of direct query expansion

<table>
<thead>
<tr>
<th>Original user query:</th>
<th>Expanded user query:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mainboard CPU Socket KV133 ABIT</td>
<td>Mainboard Motherboard CPU Central process unit, processor Socket Slot, connector KV133 ABIT</td>
</tr>
</tbody>
</table>

Fig. 13. User-oriented webpage expansion strategy supported by the website models.

Please cite this article in press as: Yang, S.-Y., An ontological website models-supported search agent for web services, Expert Systems with Applications (2007), doi:10.1016/j.eswa.2007.09.024
4.1.2. Domain-oriented web search supported by website models

Now, we can discuss how Autonomous Website Evolver does domain-dependent website expansion. Autonomous Website Evolver employs a four-phase progressive strategy to autonomously expand the website models. The first phase uses Eq. (7) to calculate \( S_2(U) \) for each hyperlink \( U \) which is referred to by website \( S \) and recognized to be in \( S \) from its URL address but whose hyperlinked webpage is not yet collected in \( S \).

\[
S_2(U) = \sum_{C \in D} R_{S,D} \times (1 - P_{S,D}(C)), \\
U \in S \text{ and } P_{S,D}(C) \neq 0
\]  

(7)

where \( C \) is a concept of domain \( D \), \( R_{S,D} \) was defined in Eq. (6), and \( P_{S,D}(C) \) measures the proportion of concept correlation of website \( S \) with respect to concept \( C \) of domain \( D \). \( N_{S,C} \) refers to the number of webpages talking about domain concept \( C \) on website \( S \). Fig. 14 shows the algorithm for calculating \( N_{S,C} \). In short, \( P_{S,D}(C) \) measures how strong a website is related to a specific domain concept.

\[
P_{S,D}(C) = \frac{N_{S,C}}{N_{S,D}}
\]  

(8)

Literally, Eq. (7) assigns a higher score to \( U \) if \( U \) belongs to a website \( S \) which has a higher degree of domain correlation \( R_{S,D} \), but contains less domain concepts, i.e., less \( P_{S,D}(C) \). Fig. 15 illustrates how this strategy works. It shows webpages I and J will become the first choices for calculating their priority scores by the first phase. In the figure, the upper nodes represent the webpages stored in the website models and the lower nodes represent the webpages whose URLs are hyperlinked in the upper nodes. Nodes in dotted circles, e.g., nodes I and J represent webpages not yet stored in the website models and needed to be collected. The figure shows webpage J referred to by webpage A in website 1 belongs to website 1, but is not yet collected into its website model. Similarity, webpage I referred to by webpage E in website 2 has not been collected into its website model. Note that webpage I is also referred to by webpage C in website 1. In summary, the first phase prefers to expand the websites that are well profiled in the website models but have less coverage of domain concepts.

The first phase is good at collecting more webpages for well profiled websites; it cannot help with unknown websites, however. Our second phase goes a step further by searching for webpages that can help define a new website profile. In this phase, we exploit URLs that are in the website models, but belong to some unknown website profile. We use Eq. (9) to calculate \( S_2(U) \) for each domain concept 1 hyperlink \( U \) of some webpages that is stored in an indefinite website profile.

\[
S_2(U) = \text{Anchor}(U, D)
\]  

(9)

where function \( \text{Anchor}(U, D) \) gives outbound link \( U \) a weight according to how many terms in the anchor text of \( U \) belong to domain \( D \).

Fig. 16 illustrates how this strategy works. Phase 2 will choose hyperlink \( \text{H} \) for priority score calculation. The unknown website \( X \) represents a website profile which contains insufficient webpages to make clear its relationships to some domains. Thus, the second phase prefers to expand those webpages that can help bring in more information to complete the specification of indefinite website profiles.

In the third phase, we relax one more constraint; we relax the condition of unknown website profiles. We exploit any URLs as long as they are referred to by some webpages in the website models. We use Eq. (10) to calculate \( S_2(U) \) for each page hyperlink \( U \) which are referred to by any webpage in the website models. This equation heavily relies on the anchor texts to determine which URLs should receive higher priority scores.

\[
\text{Let } C \text{ be a Concept, } S \text{ be a Website, and } D \text{ be a Domain.} \\
\text{Let } \text{THW} \text{ be some threshold numbers for the website models.} \\
\text{OntoPage}(S) = \text{The webpages on website } S \text{ belong to } D. \\
\text{OntoCon(THW, P)} = \text{The top THW domain concepts of webpage } P.
\]

Calculating \( N_{S,C}(S, C) \)

\[
\begin{align*}
\text{Calculating } N_{S,C}(S, C) & \text{ \{} \\
& \text{For each webpage } P \text{ in OntoPage}(S) \text{ \{} \\
& \quad \text{If OntoCon(THW, P)} \text{ contains } C \\
& \quad \quad N_{S,C} = N_{S,C} + 1 \\
& \quad \text{\} } \\
& \text{Return } N_{S,C} \\
\end{align*}
\]  

Fig. 14. Algorithm for calculating \( N_{S,C} \).

Let C be a concept, S be a website, and D be a domain.
Let THW be some threshold numbers for the website models.
OntoPages(S) = The webpages on website S belong to D.
OntoCon(THW, P) = The top THW domain concepts of webpage P.
Calculating \( N_{S,C}(S, C) \)

\[
\begin{align*}
\text{Calculating } N_{S,C}(S, C) & \text{ \{} \\
& \text{For each webpage } P \text{ in OntoPage}(S) \text{ \{} \\
& \quad \text{If OntoCon(THW, P)} \text{ contains } C \\
& \quad \quad N_{S,C} = N_{S,C} + 1 \\
& \quad \text{\} } \\
& \text{Return } N_{S,C} \\
\end{align*}
\]  

Fig. 14. Algorithm for calculating \( N_{S,C} \).

Please cite this article in press as: Yang, S.-Y., An ontological website models-supported search agent for web services, Expert Systems with Applications (2007), doi:10.1016/j.eswa.2007.09.024
\[ S_2(U) = \sum_{C \in B} \text{Anchor}(U,C) \times R_{S,D} \times P_{S,D}(C) \]  

(10)

Fig. 17 illustrates how the strategy works. It will select all nodes in dotted circles for priority score calculation. In short, the third phase tends to collect every webpage that is referred to by the webpages in the website models.

In the last phase, we resort to general website information to refresh and expand website profiles. This phase is periodically invoked according to the Update_Time/Date stored in the website profiles and webpage profiles. Specifically, we refer to the analysis of refresh cycles of different types of websites conducted in Cho and Garcia-Molina (2000) and define a weight for each web type as shown in Table 5. This phase then uses Eq. (11) to assign an \( S_2(U) \) to each \( U \) which belongs to a specific website type \( T \).

\[ S_2(U) = R_{S,D} \times W_T, \quad U \in T \]  

(11)

Fig. 18 summarizes how this four-phase progressive website expansion strategy works.

4.2. Webpage retrieval from website models

Webpage retrieval concerns the way of providing most-needed documents for users. Traditional ranking methods employ an inverted full-text index database along with a ranking algorithm to calculate the ranking sequence of relevant documents. The problems with this method are clear: too many entries in returned results and too slow response time. A simplified approach emerged, which employs various ad hoc mechanisms to reduce query space (Salton & McGill, 1983, 1988). Two major problems are behind these mechanisms: (1) they need a specific, labor-intensive and time-consuming pre-process and; (2) they cannot respond to the changes of the real environment in time due to the off-line pre-process. Another new method called Page, Brin, Motwani, and Winograd (1999) was employed in Google to rank webpages by their link information. Google spends lots of offline time pre-analyzing the link relationships among a huge number of webpages and calculating proper ranking scores for them before storing them in a special database for answering user query. Google’s high speed of response stems from a huge local webpage database along with a time-consuming, offline detailed link structure analysis.

Instead, our solution ranking method takes advantage of the semantics in the website models. The major index structure uses ontology features to index webpages in the website models. The ontology index contains terms that are stored in the webpage profiles. The second index structure is a partial full-text inverted index since it contains no ontology features. Fig. 19 shows this two-layered index structure. Since we require each query contain at least one ontology feature, we always can use the ontology index to locate a set of webpages. The partial full-text index is then used to further reduce them into a subset of webpages for users.

This design of separating ontology indices from a traditional full-text is interesting. Since we then know what
ontology features are contained in a user query. Based on this information, we can apply OntoClassifier to analyze what ontology classes (equivalently domain concepts) the user are really interested in and use the information to fast locate user interested webpages. Let us explain how this is done in our system. First, we use the second stage of OntoClassifier along with a threshold, say THU, to limit the best classes a query is associated with. For example, if we set THU to three, we select the best three ontology classes from a query and use them as indices to fast locate user interested webpages.

As a matter of fact, we can leverage the identified ontology features in a user query to properly rank the webpages for the user using the ranking method defined by Eq. (12). In the first term of the equation, $M_{Q}(P)$ is the number of user terms appearing in webpages $P$, which can be obtained from Fig. 19, and $W_{QU}$ is its weighting value. $P_{S,D}(T)$ is defined by Eq. (13), which measures, for each term $T$ in the user term part of query $Q$ (i.e., $Q_{U}(Q)$), the ratio of the number of webpages that contain $T$ (i.e., $N_{ST}$), and the total number of webpages related to $D$ (i.e., $N_{SD}$), on website $S$. Multiplying these factors together represents how strong the to-be-retrieved webpages are user terms-oriented. The second term of Eq. (12) does a similar analysis on ontology features appearing in the user query. Basically, $W_{QO}$ is a weighing value for the ontology term part of query $Q$, and $M_{Q}(P)$ is the number of ontology features appearing in webpage $P$, which can be obtained from Fig. 19 too. As to the factor of $P_{S,D}(T)$, we have a slightly different treatment here. It is used to calculate the ratio of the number of webpages containing ontology feature $T$, but we restrict $T$ to appear only in the top THU concepts, as we have set a threshold number of domain concepts for each user query. We thus need to add a factor $P_{TH}(Q,P)$ to reflect the fact that we also apply a threshold number of domain concepts, THW, for each webpage (see Section 3.2). $P_{TH}(Q,P)$ is defined by Eq. (14) measuring the ratio of the number of domain concepts that appear both in the top THU concepts of query $Q$ and the top THW concepts of domain $D$ (i.e., $M_{TH}(Q,P)$) to the number of domain concepts that appear only in the top THU concepts of $Q$ (i.e., $M_{TH}(Q)$). This second term thus represents how strong the to-be-retrieved webpages are related to user interested domain concepts.

$$RA(Q,P) = W_{QU} \cdot M_{Q}(P) \cdot \sum_{T \in Q_{U}(Q)} P_{S,D}(T) + W_{QO} \cdot M_{QO}(P) \cdot P_{TH}(Q,P) \cdot \sum_{T \in Onto(THU,Q)} P_{S,D}(T)$$

(12)

$$P_{S,D}(T) = \frac{N_{ST}}{N_{SD}}$$

(13)

$$P_{TH}(Q,P) = \frac{M_{TH}(Q,P)}{M_{TH}(Q)}$$

(14)

$$W_{QU} + W_{QO} = 1$$

(15)

Note that the two weighting factors are correlated as defined by Eq. (15). The user is allowed to change the ratio between them as illustrated in Fig. 20 to reflect his emphasis on either user terms or ontology features in retrieving webpages.

5. System evaluation

5.1. Architecture of the search agent

Fig. 21 shows the architecture of our Search Agent, which integrates the design concepts and techniques discussed in the previous sections. To recap, Focused Crawler is responsible for gathering webpages into DocPool according to user interests and website model weakness. Model Constructor extracts important information from a webpage stored in DocPool and annotates proper ontology information to make a webpage profile for it. It also constructs a website profile for each website in due time according to what webpages it contain. Webpage Retrieval uses ontology features in a given user query to fast locate and rank a set of most needed webpages in the website models and displays it to the user. Finally, the User Interface receives a user query, expands the query using ontology, and sends it to Webpage Retrieval, which in turn returns a list of ranked webpages. Fig. 22 illustrates the various ways in which the user can enter query. Fig. 22(a) shows the traditional keyword-based model along with the help of ontology features shown in the left column. The user can directly choose ontology terms into
the input field. Fig. 22(b) shows the user can use natural language to input his query too. The user interface employs a template matching technique to select best-matched query templates (Chiu, 2003; Yang, 2007), as shown in Fig. 22(c) for the users to conform. Note that either of the query is finally transformed into a list of keywords internally by User Interface for subsequent query expansion.

Fig. 23 illustrates a retrieval result for user query returned from Webpage Retrieval which contains a ranked list of webpages. This retrieval method emphasizes the “precision” criteria. As a matter of fact, Webpage Retrieval can structure another way of retrieval result to emphasize the “recall” factor by retrieving and ranking the websites that are most suited to answer the user query. Fig. 24 exemplifies such a returned result composed of a ranked list of websites along with their contained webpages. Either type of the ranked lists will be turned over to the User Interface for personalization before being displayed to the user.

5.2. Experiment environment

Our Search Agent is developed using Borland JBuilder 7.0 on Windows XP. We collected in total ten classes with 100 webpages in each class from hardware-related Websites, as shown in Table 6.

5.3. Performance evaluation of OntoClassifier

The first experiment is to learn how well OntoClassifier works. We applied the feature selection program as described in ontology-reorganization to all collected webpages to select ontology features for each class. Table 7 shows the number of features for each class.

To avoid unexpected delay we limit the level of related concepts to 7 during the second stage classification of OntoClassifier. Fig. 25 shows its performance for each class. Several interesting points deserve notice here. First, with a very small number of ontology features, OntoClassifier can perform very accurate classification results in virtually all classes. Even with 10 features, over 80% accuracy of classification can be obtained in all classes. Second, the accuracy of classification of OntoClassifier is very stable with respect to the number of ontology features. In Wang

Table 6

<table>
<thead>
<tr>
<th>Class</th>
<th>Webpage count</th>
<th>Website count</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>Motherboard</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>Graphic card</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>Sound card</td>
<td>100</td>
<td>13</td>
</tr>
<tr>
<td>Network card</td>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td>SCSI card</td>
<td>100</td>
<td>7</td>
</tr>
<tr>
<td>Optical drive</td>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td>Monitor</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>Hard drive</td>
<td>100</td>
<td>5</td>
</tr>
<tr>
<td>Modem</td>
<td>100</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 7

<table>
<thead>
<tr>
<th>Class</th>
<th># of features</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>93</td>
</tr>
<tr>
<td>Motherboard</td>
<td>96</td>
</tr>
<tr>
<td>Graphic card</td>
<td>90</td>
</tr>
<tr>
<td>Sound card</td>
<td>92</td>
</tr>
<tr>
<td>Network card</td>
<td>67</td>
</tr>
<tr>
<td>SCSI card</td>
<td>55</td>
</tr>
<tr>
<td>Optical drive</td>
<td>88</td>
</tr>
<tr>
<td>Monitor</td>
<td>79</td>
</tr>
<tr>
<td>Hard drive</td>
<td>57</td>
</tr>
<tr>
<td>Modem</td>
<td>66</td>
</tr>
</tbody>
</table>
(2003), we have reported a performance comparison between OntoClassifier and three other similar classifiers, namely, O-PrTFIDF (Joachims, 1997), T-PrTFIDF (Ting, 2000), and D-PrTFIDF (Wang, 2003). All the three classifiers and their respective feature selection methods were re-implemented. We found that none of these three classifiers can match the performance of OntoClassifier with respect to either classification accuracy or classification stability.

To verify that the superior performance of OntoClassifier is not due to overfitting, we used 1/3, 1/2, and 2/3 of the collected webpages, respectively, for training in each class to select the ontology features and used all webpages of the class for testing. Table 8 shows how OntoClassifier behaves with respect to different ratios of training samples. The column of number of features gives the number of ontology features used in each class. It does show that the superior accuracy performance can be obtained even with 1/3 of training webpages.

5.4. Do ontology features work for any classifiers?

The second experiment is to learn whether the superior performance of OntoClassifier is purely due to ontology features. In other words, can the ontology features work for other classifiers too? From this purpose, this experiment will use the same set of ontology features derived for OntoClassifier to test the performance of O-PrTFIDF, D-PrTFIDF and T-PrTFIDF, the three classifiers mentioned before. This time we only used 1/3 of the collected Web pages for training the ontology features and used all Web pages for testing. To make each classifier work best, we allow each classifier to arbitrarily choose the first 40 features that make it work best. We limit the number to be 40, because the class SCSI only has 38 features. Fig. 26 illustrates how the four classifiers work for class as CPU, Motherboard, Graphic Card, and Sound Card.

We note that O-PrTFIDF and D-PrTFIDF classifiers are the most unstable among the four with respect to different numbers of features. The T-PrTFIDF classifier works rather well except for larger features, which is because T-PrTFIDF was designed to work based on ontology (Ting, 2000). Its computation complexity is greater than OntoClassifier though. From this experiment, we learn that ontology features alone do not work for any classifier; the ontology features work best for those classifiers that are crafted by taking into account how to leverage the power of ontology. OntoClassifier is such a classification mechanism.

5.5. User-satisfaction evaluation of system prototype

Table 9 shows the comparison of user satisfaction of our system prototype against other search engines. In the table, $S_T$, for satisfaction of testers, represents the average of satisfaction responses from 10 ordinary users, while $S_E$, for satisfaction of experts, represents that of satisfaction responses from 10 experts. Basically, each search engine receives 100 queries and returns the first 100 webpages for evaluation of satisfaction by both experts and non-experts. The table shows that our system prototype with the techniques described above, the last row, enjoys the highest satisfaction in all classes. From the evaluation, we conclude that, unless the comparing search engines are specifically tailored to this specific domain, such as HotBot and Excite, our system prototype, in general, retrieves more correct webpages in almost all classes.

6. Related works

We notice that ontology is mostly used in the systems that work on information gathering or classification to improve their gathering processes or the search results.

Table 8

<table>
<thead>
<tr>
<th>Class</th>
<th>1/3 Training data</th>
<th>1/2 Training data</th>
<th>2/3 Training data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of Features</td>
<td>Accuracy (%)</td>
<td># of Features</td>
</tr>
<tr>
<td>CPU</td>
<td>69</td>
<td>97</td>
<td>78</td>
</tr>
<tr>
<td>Motherboard</td>
<td>81</td>
<td>100</td>
<td>89</td>
</tr>
<tr>
<td>Graphic card</td>
<td>61</td>
<td>100</td>
<td>73</td>
</tr>
<tr>
<td>Sound card</td>
<td>73</td>
<td>98</td>
<td>73</td>
</tr>
<tr>
<td>Network card</td>
<td>53</td>
<td>94</td>
<td>60</td>
</tr>
<tr>
<td>SCSI card</td>
<td>38</td>
<td>93</td>
<td>48</td>
</tr>
<tr>
<td>Optical drive</td>
<td>73</td>
<td>90</td>
<td>82</td>
</tr>
<tr>
<td>Monitor</td>
<td>69</td>
<td>100</td>
<td>74</td>
</tr>
<tr>
<td>Hard drive</td>
<td>39</td>
<td>99</td>
<td>44</td>
</tr>
<tr>
<td>Modem</td>
<td>64</td>
<td>100</td>
<td>66</td>
</tr>
</tbody>
</table>
from disparate resources (Eichmann, 1998). For instance, MELISA (Abasolo & Gómez, 2000) is an ontology-based information retrieval agent with three levels of abstraction, separated ontologies and query models, and definitions of some aggregation operators for combining results from different queries. WebSifter II is a semantic taxonomy-based, personalizable meta-search agent (Kerschberg, Kim, & Scime, 2001) that tries to capture the semantics of a user’s decision-oriented search intent, to transform the semantic query into target queries for existing search engines, and to rank the resulting page hits according to a user-specified weighted-rating scheme. Chen and Soo (2001) describes 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. Park and Zhang (2003) describe a novel method for webpage classification that is based on a sequential learning of the classifiers which are trained on a small number of labeled data and then augmented by a large number of unlabeled data. Wang, Yu, and Nishino (2004) propose a new website information detection system based on Webpage type classification for searching information in a particular domain. SALEM (Semantic Annotation for LEgal Management) (Bartolini, Lenci, Montemagni, Pirrelli, & Soria, 2004) is an incremental system developed for automated semantic annotation of (Italian) law texts to effective indexing and retrieval of legal documents. Chan and Lam (2005) propose an approach for facilitating the functional annotation to the Gene ontology by focusing on a subtask of annotation, that is, to determine which of the Gene ontology a literature is associated with. Swoogle (Ding et al., 2004) 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, Song, Lim, Park, Kang, and Lee (2005) suggest an automated method for document classification using an ontology, which expresses terminology information and vocabulary contained in Web documents by way of a hierarchical structure. In this paper, we not only proposed ontology-directed classification mechanism, namely, Onto-

![Graph](image-url)

**Table 9**

<table>
<thead>
<tr>
<th>K_Word method</th>
<th>CPU ($S_D/S_T$)</th>
<th>Motherboard ($S_D/S_T$)</th>
<th>Memory ($S_D/S_T$)</th>
<th>Average ($S_D/S_T$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yahoo</td>
<td>67% / 61%</td>
<td>77% / 78%</td>
<td>38% / 17%</td>
<td>61% / 52%</td>
</tr>
<tr>
<td>Lycos</td>
<td>64% / 67%</td>
<td>77% / 76%</td>
<td>36% / 20%</td>
<td>59% / 34%</td>
</tr>
<tr>
<td>InfoSeek</td>
<td>69% / 70%</td>
<td>71% / 70%</td>
<td>49% / 28%</td>
<td>63% / 36%</td>
</tr>
<tr>
<td>HotBot</td>
<td>69% / 63%</td>
<td>78% / 76%</td>
<td>62% / 31%</td>
<td>70% / 37%</td>
</tr>
<tr>
<td>Google</td>
<td>66% / 64%</td>
<td>81% / 80%</td>
<td>38% / 21%</td>
<td>62% / 35%</td>
</tr>
<tr>
<td>Excite</td>
<td>66% / 62%</td>
<td>81% / 81%</td>
<td>50% / 24%</td>
<td>66% / 56%</td>
</tr>
<tr>
<td>Alta Vista</td>
<td>63% / 61%</td>
<td>77% / 78%</td>
<td>30% / 21%</td>
<td>57% / 53%</td>
</tr>
<tr>
<td><strong>Our prototype</strong></td>
<td><strong>78% / 69%</strong></td>
<td><strong>84% / 78%</strong></td>
<td><strong>45% / 32%</strong></td>
<td><strong>69% / 60%</strong></td>
</tr>
</tbody>
</table>

**Fig. 26.** Do ontology features work for any classifiers?
Classifier can make a decision of the class for a webpage or a website in the semantic decision process for Web services, but advocated the use of ontology-supported website models to provide a semantic level solution for a search agent so that it can provide fast, precise and stable search results.

As to Web search, current general search engines use the concept of crawlers (spider or soft-robut) to help users automatically retrieve useful Web information in terms of ad-hoc mechanisms. For example, Dominos (Hafri & Djeraba, 2004) can crawl several thousands of pages every second, include a high-performance fault manager, be platform independent and be able to adapt transparently to a wide range of configurations without incurring additional hardware expenditure. Ganesh, Jayaraj, Kalyan, and Aghila (2004) proposes the association-metric to estimate the semantic content of the URL based on the domain dependent ontology, which in turn strengthens the metric that is used for prioritizing the URL queue. Ubicrawler (Boldi, Codenotti, Samtini, & Vigna, 2004), a scalable distributed web crawler, is platform independent, linear scalability, graceful degradation in the presence of faults, a very effective assignment function for partitioning the domain to crawl, and more in general the complete decentralization of every task. Chan (2008) proposes an intelligent spider that consists of a URL searching agent and an auction data agent to automatically correct related information by crawling over 1000 deals from Taiwan’s eBay whenever users input the searched product. Finally, Google adopts a PageRank approach to rank a large number of webpage link information and pre-record it for solving the problem (Brin & Page, 1998). A general Web crawler is, in general, a greedy tool that may make the URL list too large to handle. Focused Crawler, instead, aims at locating domain knowledge and necessary meta information for assisting the system to find related Web targets. The concept of Distiller is employed to rank URLs for the Web search (Barfourosh, Nezhad, Anderson, & Perlis, 2002; Diligenti, Coetzee, Lawrence, Giles, & Gori, 2000; Rennie & McCallum, 1999). IBM is an example, which adopts the HITs algorithm, similar to PageRank, for controlling web search Kleinberg, 1999. These methods are ad-hoc and need an off-line, time-consuming pre-processing. In our system, we not only develop a focused crawler using website models as the core technique, which helps search agents successfully tackle the problems of search scope and user interests, but introduce the four-phase progressive website expansion strategy for the focused crawler to control the Web search, which takes into account both user interests and domain specificity.

7. Conclusions and discussion

We have described how ontology-supported website models can effectively support Web search. A website model contains webpage profiles, each recording basic information, statistics information, and ontology information of a webpage. The ontology information is an annotation of how the webpage is interpreted by the domain ontology. The website model also contains a website profile that remembers how a website is related to the webpages and how it is interpreted by the domain ontology.

We have developed a Search Agent, which employs domain ontology-supported website models as the core technology to search for Web resources that are both user-interested and domain-oriented. Our preliminary experimentation demonstrates that the system prototype can retrieve more correct webpages with higher user satisfaction. The Agent features the following interesting characteristics. (1) Ontology-supported construction of website models. By this, we can attribute domain semantics into the Web resources collected and stored in the local database. One important technique used here is the Ontology-supported OntoClassifier which can do very accurate and stable classification on webpages to support more correct annotation of domain semantics. Our experiments show that OntoClassifier performs very well in obtaining accurate and stable webpages classification. (2) Website models-supported Website model expansion. By this, we can take into account both user interests and domain specificity. The core technique here is the Focused Crawler which employs progressive strategies to do user query-driven webpage expansion, autonomous website expansion, and query results exploitation to effectively expand the website models. (3) Website models-supported Webpage Retrieval. We leverage the power of ontology features as a fast index structure to locate most-wanted webpages for the user. (4) We mentioned that the User Interface works as a query expansion and answer personalization mechanism for Search Agent. As a matter of fact, the module has been expanded into a User Interface Agent in our information integration system (Yang, 2006; Yang, 2006). The User Interface Agent can interact with the user in a more semantics-oriented way according to his proficiency degree about the domain (Yang, 2007; Yang, 2007).

Most of our current experiments are on the performance test of OntoClassifier. We are unable to do experiments on or comparisons of how good the Search Agent is at expanding useful Web resources. Our difficulties are summarized below. (1) To our knowledge, none of current Web search systems adopt a similar approach as ours in the sense that none of them are relying on ontology as heavily as our system to support Web search. It is thus rather hard for us to do a fair and convincing comparison. (2) Our ontology construction is based on a set of pre-collected webpages on a specific domain; it is hard to evaluate how critical this pre-collection process is to the nature of different domains. We are planning to employ the technique of automatic ontology evolution, for example, cooperated with data mining technology for discovering useful information and generating desired knowledge that support ontology construction (Wang, Lu, & Zhang, 2007), to help study the robustness of our ontology. (3) In general, a domain ontology-based technique cannot work as a gen-

Please cite this article in press as: Yang, S.-Y., An ontological website models-supported search agent for web services, Expert Systems with Applications (2007), doi:10.1016/j.eswa.2007.09.024
eral-purpose search engine. We are planning to create a general-purpose search engine by employing a multiple number of our Search Agent, supported by a set of domain ontologies, through a multi-agent architecture.

Acknowledgements

The author would like to thank J.-Chiang Liou, Yung Ting, Jr-Well Wu, Yu-Ming Chung, Zing-Tung Chou, Ying-Hao Chiu, Ben-Chin Liao, Yi-Ching Chu, Shu-Ting Chang, Yai-Hui Chang, Chung-Min Wang, and Fang-Chen Chuang for their assistance in system implementation. This work was supported by the National Science Council, ROC, under Grants NSC-89-2213-E-011-059, NSC-89-2218-E-011-014, and NSC-95-2221-E-129-019.

References


Developing of an ontological interface agent with template-based linguistic processing technique for FAQ services

Sheng-Yuan Yang *

Department of Computer and Communication Engineering, St. John's University, 499, Sec. 4, TamKing Road, Tamsui, Taipei County 251, Taiwan, ROC

ARTICLE INFO

Available online xxxx

Keywords:
Ontological interface agents
Template-based query processing
FAQ services

ABSTRACT

This paper proposes an ontological Interface agent which works as an assistant between the users and FAQ systems. We integrated several interesting techniques including domain ontology, user modeling, and template-based linguistic processing to effectively tackle the problems associated with traditional FAQ retrieval systems. Specifically, we address the following issues. Firstly, how can an interface agent learn a user's specialty in order to build a proper user model for him/her? Secondly, how can domain ontology help in establishing user models, analyzing user query, and assisting and guiding interface usage? Finally, how can the intention and focus of a user be correctly extracted? Our work features a template-based linguistic processing technique for developing ontological interface agents; a nature language query mode, along with an improved keyword-based query mode; and an assistance and guidance for human–machine interaction. Our preliminary experimentation demonstrates that user intention and focus of up to eighty percent of the user queries can be correctly understood by the system, and accordingly provides the query solutions with higher user satisfaction.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

With increasing popularity of the Internet, people depend more on the Web to obtain their information just like a huge knowledge treasury waiting for exploration. People used to be puzzled by such problems as “how to search for information on the Web treasury?” As the techniques of Information Retrieval (Salton, Wong, & Yang, 1975; Salton & McGill, 1983) matured, a variety of information retrieval systems have been developed, e.g., Search engines, Web portals, etc., to help search on the Web. How to search is no longer a problem. The problem now comes from the results from these information retrieval systems which contain so much information that overwhelms the users. Now, people wants the information retrieval systems to do more help, for instance, by only retrieving the results which better meet the user's requirement. Other wanted capabilities include a better interface for the user to express his/her true intention, better-personalized services and so on. In short, how to improve traditional information retrieval systems to provide search results which can better meet the user requirements so as to reduce his/her cognitive loading is an important issue in current research (Chiu, 2003).

The websites which provide Frequently Asked Questions (FAQ) organize user questions and expert answers about a specific product or discipline in terms of question–answer pairs on the Web. Each FAQ is represented by one question along with one answer and is characterized, to be domain-dependent, short and explicit, and frequently asked (Lee, 2000; OuYang, 2000). People usually go through the list of FAQs and read those FAQs that are to his/her questions. This way of answering the user’s questions can save the labor power for experts from answering similar questions repeatedly. The problem is after the fast accumulation of FAQs, it becomes harder for people to single out related FAQs. Traditional FAQ retrieval systems, however, provide only little help, became they fail to provide assistance and guidance for human–machine interaction, personalized information services, flexible interaction interface, etc. (Chiu, 2003).

In order to capture true user’s intention and accordingly provides high-quality FAQ answers to meet the user requests, we have proposed an Interface Agent acquires user intention through an adaptive human–machine interaction interface with the help of ontology-directed and template-based user models (Yang et al., 1999; Yang, Chiu, & Ho, 2004; Yang, 2006). It also handles user feedback on the suitability of proposed responses. The agent features ontology-based representation of domain knowledge, flexible interaction interface, and personalized information filtering and display. Specifically, according to the user's behavior and mental state, we employed the technique of user modeling to construct a user model to describe his/her characteristics, preference and knowledge proficiency level, etc. We also used the technique of user stereotype (Rich, 1979) to construct and initialize a new user model, which helps provide fast-personalized services for new users.

* Corresponding author. Tel.: +886 2 28013131x6394; fax: +886 28013131x6391. E-mail address: yys@mail.sju.edu.tw.
We built domain ontology (Noy & McGuinness, 2001) to help define domain vocabulary and knowledge and based on that to construct user models and the Interface Agent. We extended the concept of pattern match (Hovy, Hermjakob, & Ravichandran, 2002) to query template to construct natural language based query models. This idea leads to the extraction of true user intention and focus from his/her query posted as nature language understanding, which help the agent to fast find out precise information for the user, just as (Coyle & Smyth, 2005) in which using interaction histories for enhancing search results. Our preliminary experiment demonstrates that the intention and focus of up to eighty percent of the users’ queries can be correctly understood, and accordingly provides the query solutions with higher user satisfaction.

The rest of the paper is organized as follows: Section 2 describes the fundamental techniques. Section 3 explains the Interface Agent architecture. Section 4 reports the system demonstrations and evaluations. Section 5 compares the work with related works, while Section 6 conclude the work. The Personal Computer (PC) domain is chosen as the target application of our Interface Agent and will be used for explanation in the remaining sections.

2. Fundamental techniques

2.1. Domain ontology

The concept of ontology in artificial intelligence refers to knowledge representation for domain-specific contents (Chandrasekaran, Josephson, & Benjamins, 1999). It has been advocated as an important tool to support knowledge sharing and reusing in developing intelligent systems. Although development of an ontology for a specific domain is not yet an engineering process, we have outlined a procedure for this in Yang et al. (1999) from how the process was conducted in existent systems. By following the procedure we developed an ontology for the PC domain using Protégé 2000 (Noy & McGuinness, 2001) as the fundamental background knowledge for the system, which was originally developed in Chinese (Yang, Chuang, & Ho, 2007) but was changed to English here for easy explanation. Fig. 1 shows part of the ontology taxonomy. The taxonomy represents relevant PC concepts as classes and their parent–child relationships as isa links, which allow inheritance of features from parent classes to child classes. We carefully selected those properties that are most related to our application from each concept, and defined them as the detailed ontology for the corresponding class. Fig. 2 exemplifies the detailed ontology of the concept of CPU. In the figure, the root node uses various fields to define the semantics of the CPU class, each field representing an attribute of “CPU”, e.g., interface, provider, synonym, etc. The nodes at the lower level represent various CPU instances, which capture real world data. The arrow line with term “io” means the instance of relationship. The complete PC ontology can be referenced from the Protégé Ontology Library at Stanford Website (<http://protege.stanford.edu/download/download.html>). We also developed a problem ontology to deal with query questions. Fig. 3 illustrates part of the Problem ontology, which contains query type and operation type. Together they imply the semantics of a question. Finally, we use Protégé’s APIs to develop a set of ontology services, which provide primitive functions to support the application of the ontologies. The ontology services currently available include transforming query terms into canonical ontology terms, finding definitions of specific terms in ontology, finding relationships among terms, finding compatible and/or conflicting terms against a specific term, etc.
2.2. Query templates

To build the query templates, we have collected in total 1215 FAQs from the FAQ website of six famous motherboard factories in Taiwan and used them as the reference materials for query template construction (Hovy et al., 2002; Soubbotin et al., 2001). Currently, we only take care of the user query with one intention word and at most three sentences. These FAQs were analyzed and categorized into six types of questions as shown in Table 1. For each type of question, we further identified several intention types according to its operations. Table 2 illustrates some examples of intention types. Finally, we define a query pattern for each intention type. Table 3 illustrates the defined query patterns for the intention types of Table 2. Table 4 explains the syntactical constructs of the query patterns.

Now all information for constructing a query template is ready, and we can formally define a query template. Table 5 defines what a query template is. It contains a template number, number of sentences, intention words, intention type, question type, operation type, query patterns, and focus. Table 6 illustrates an example query template for the ANA_CAN_SUPPORT intention type. Note here that we collect similar query patterns in the field of “Query patterns,” which are used in detailed analysis of a given query.

Table 1
Question types

<table>
<thead>
<tr>
<th>Question type</th>
<th>Intention</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A-NOT-A)</td>
<td>Asks about can or cannot, should or should not, have or have not</td>
</tr>
<tr>
<td>HOW</td>
<td>Asks about solving methods</td>
</tr>
<tr>
<td>WHAT</td>
<td>Enumerates related information</td>
</tr>
<tr>
<td>WHEN</td>
<td>Asks about time, year, or date</td>
</tr>
<tr>
<td>WHERE</td>
<td>Asks place or position</td>
</tr>
<tr>
<td>WHY</td>
<td>Asks reasons</td>
</tr>
</tbody>
</table>

Table 2
Examples of intention types

<table>
<thead>
<tr>
<th>Intention Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANA_CAN_SUPPORT</td>
<td>Asks if support some specifications or products</td>
</tr>
<tr>
<td>HOW_SET</td>
<td>Asks the method of assignment</td>
</tr>
<tr>
<td>WHAT_IS</td>
<td>Asks the meaning of terminology</td>
</tr>
<tr>
<td>WHEN_SUPPORT</td>
<td>Asks when can support</td>
</tr>
<tr>
<td>WHERE_DOWNLOAD</td>
<td>Asks where can download</td>
</tr>
<tr>
<td>WHY_SETUP</td>
<td>Asks reasons about setup</td>
</tr>
</tbody>
</table>

Table 3
Examples of query patterns

<table>
<thead>
<tr>
<th>Query Pattern</th>
<th>Intention Type</th>
<th>Operation Type</th>
<th>Question Type</th>
</tr>
</thead>
</table>
| ANA_CAN_SUPPORT | <s1> 是否 支援 <s2> | GA-7VXK 樣板卡係可以支援 KINOMAX DDR-640 ? (Could the GA-7VXK motherboard support the KINOMAX DDR-640 memory type?)
| HOW_SET       | <s1> 如何 安装 <s2> | 如何在 Windows 98SE 下，安裝 8RDA 的音效驅動程式？ (How to setup the 8RDA sound driver on a Windows 98SE platform?)
| WHAT          | <s1> 是 <s2> | WHAT_IS <s1> 是 <s2> | AUX power connector 是什麼？ (What is an AUX power connector?)
| WHEN_SUPPORT  | <s1> 該時 選用 <s2> | P4T 何時才推出支援 32-bit 512 MB RDRAM 記憶體技術？ (When can the P4T support the 32-bit 512 MB RDRAM memory specification?)
| WHERE_DOWNLOAD| <s1> 下載 <s2> | WHERE_DOWNLOAD <s1> 下載 <s2> | CUA 的 Driver CD 損失，請問檔案可以在哪裡下載驅動程式？(Where can I download the sound driver of CUA whose Driver CD was lost?)
| WHY_PRINT     | <s1> 原因 <s2> | WHY_PRINT <s1> 原因 <s2> | 為什麼在 Win ME 之下，從休眠狀態中回復後，印表機無法列印。 (Why can I not print after coming back from dormancy on a Win ME platform?)

Please cite this article in press as: Yang, S.-Y., Developing of an ontological interface agent with template-based linguistic ..., Expert Systems with Applications (2008), doi:10.1016/j.eswa.2008.03.011
According to the generalization relationships among intention types, we can form a hierarchy of intention types to organize all FAQs. Currently, the hierarchy contains two levels as shown in Fig. 4. Now, the system can employ the intention type hierarchy to reduce the search scope during the retrieval of FAQs after the intention of a user query is identified. Table 7 shows the statistics of query templates in our system. Currently, we have in total 154 intention words which form an intention word base.

### 3. Interface agent architecture

#### 3.1. User modeling

In order to provide customized services, we observe, record, and learn the user behavior and mental state in a user model. A user model contains interaction preference, solution presentation, domain proficiency, terminology table, query history, selection history, and user feedback, as shown in Fig. 5. The interaction preference is responsible for recording user’s preferred interface, e.g., favorite query mode, favorite recommendation mode, etc. When the user logs on the system, the system can select a proper user interface according to this preference. We provide two modes, either through keywords or natural language input. We provide three recommendation modes according to hit rates, hot topics, or collaborative learning. We record recent user’s preferences in a time window, and accordingly determine the next interaction style. The solution presentation is responsible for recording solution ranking preferences of the user. We provide two types of ranking, either according to the degree of similarity between the proposed solutions and the user query, or according to user’s proficiency about the solutions. In addition, we use a Show_Rate parameter (described later) to control how many items of solutions for display each time, in order to reduce information-overloading problem. The domain proficiency factor describes how familiar the user is with the domain. By associating a proficiency degree with each ontology concept, we can construct a table, which contains a set of (concept proficiency-degree) pairs, as his/her domain proficiency. Thus, during the decision of solution representation, we can calculate the user’s proficiency degree on solutions using the table, and accordingly only show his/her most familiar part of solutions and hide the rest for advanced requests. To solve the problem of different terminologies to be used by different users, we include a terminology table to record this terminology difference. We can use the table to replace the terms used in the proposed solutions with the user favorite terms during solutions representation to help him better comprehend the solutions. Finally, we record the user’s query history as well as FAQ selection history and corresponding user feedback in each query session in the Interaction history, in order to support collaborative recommendation. The user feedback is a complicated factor. We remember both explicit user feedback in the selection history and implicit user feedback, which includes query time, time of FAQ click, sequence of FAQ clicks, sequence of clicked hyperlinks, etc.

In order to quickly build an initial user model for a new user, we pre-defined five stereotypes, namely, expert, senior, junior, novice, and amateur (Yang et al., 1999, 2004), to represent different user group’s characteristics. This approach is based on the idea that the same group of user tends to exhibit the same behavior and requires the same information. Fig. 6 illustrates an example user stereotype. When a new user enters the system, he/she is asked to complete a

---

**Table 5**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template_Number</td>
<td>Template ID</td>
</tr>
<tr>
<td>#Sentence</td>
<td>The number of sentences: 1 for one sentence, 2 for two sentences, and 3 for three or more sentences</td>
</tr>
<tr>
<td>Intention_Word</td>
<td>Intention words must appear</td>
</tr>
<tr>
<td>Intention_Type</td>
<td>Intention type</td>
</tr>
<tr>
<td>Question_Type</td>
<td>Question type</td>
</tr>
<tr>
<td>Operation_Type</td>
<td>Operation type</td>
</tr>
<tr>
<td>Query_Patterns</td>
<td>The semantic pattern consists of intention words and keywords</td>
</tr>
<tr>
<td>Focus</td>
<td>The focus of the user</td>
</tr>
</tbody>
</table>

**Table 6**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template_Number</td>
<td>Intention type: ANA CAN SUPPORT</td>
</tr>
<tr>
<td>Intention_Word</td>
<td>是否 (if), 支持 (Support)</td>
</tr>
<tr>
<td>Intention_Type</td>
<td>ANA CAN SUPPORT</td>
</tr>
<tr>
<td>Question_Type</td>
<td>是否 (if)</td>
</tr>
<tr>
<td>Operation_Type</td>
<td>支持 (Support)</td>
</tr>
<tr>
<td>Query_Patterns</td>
<td>[S1]=S1, 是否 支持 S2&gt; [S2]=是否 支持 S1&gt;</td>
</tr>
<tr>
<td>Focus</td>
<td>S1</td>
</tr>
</tbody>
</table>

**Table 7**

<table>
<thead>
<tr>
<th>Question type</th>
<th>#Intention type</th>
<th>#Template</th>
<th>#Pattern</th>
<th>#FAQ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>是否 (if)</td>
<td>19</td>
<td>53</td>
<td>113</td>
<td>385 (31.7)</td>
</tr>
<tr>
<td>如何 (how)</td>
<td>18</td>
<td>44</td>
<td>121</td>
<td>265 (21.8)</td>
</tr>
<tr>
<td>什麼 (what)</td>
<td>6</td>
<td>18</td>
<td>19</td>
<td>91 (7)</td>
</tr>
<tr>
<td>何時 (when)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3 (0.2)</td>
</tr>
<tr>
<td>何处 (where)</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4 (0.3)</td>
</tr>
<tr>
<td>为什麼 (why)</td>
<td>25</td>
<td>199</td>
<td>771</td>
<td>467 (38.4)</td>
</tr>
<tr>
<td>Total</td>
<td>69</td>
<td>319</td>
<td>1031</td>
<td>1215</td>
</tr>
</tbody>
</table>

---

**Fig. 4.** Intention type hierarchy.

**Fig. 5.** Our user model.
questionnaire, which is used by the system to determine his/her domain proficiency, and accordingly select a user stereotype to generate an initial user model to him/her. However, the initial user model constructed from the stereotype may be too generic or imprecise. It will be refined to reflect the specific user’s real intent after the system has experiences with his/her query history, FAQ-selection history and feedback, and implicit feedback (Chiu, 2003).

### 3.2. Architecture overview

Fig. 7 illustrates the architecture of the Interface Agent and shows how it interacts with other agents of the FAQ-master (Yang, 2006, 2007), which possesses intelligent retrieval, filtering, and integration capabilities and can provide high-quality FAQ answers from the Web. The Interaction Agent provides a personalization interaction, assistance, and recommendation interface for the user according to his/her user model, records interaction information and related feedback in the user model, and helps the User Model Manager and Proxy Agent (Yang, 2007) to update the user model. The Query Parser processes the user queries by first segmenting word, removing conflicting words, and standardizing terms, followed by the recording of the user’s terminologies in the terminology table of the user model. It finally applies the template matching technique to select best-matched query templates, and accordingly transforms the query into an internal query for the Proxy Agent to search for solutions and collect them into a list of FAQs, which each containing a corresponding URL. The Web Page Processor pre-downloads FAQ-relevant webpages and performs some pre-processing tasks, including labeling keywords for subsequent processing. The Scorer calculates the user’s proficiency degree for each FAQ in the FAQ list according to the terminology table in his/her user model. The Personalizer then produces personalized query solutions according to the terminology table. The User Model Manager is responsible for quickly building an initial user model for a new user using the technique of user stereotyping as well as updating the user models and stereotypes to dynamically reflect the changes of user behavior. The Recommender is responsible for recommending information for the user based on hit count, hot topics, or group’s interests when a similar interaction history is detected.
3.2.1. Interaction agent

The Interaction Agent consists of the following three components: Adapter, Observer, and Assistant. First, the Adapter constructs best interaction interfaces according to user's favorite query and recommendation modes. It is also responsible for presenting to the user the list of FAQ solutions (from the Personalizer) or recommendation information (from the Recommender). During solution representation, it arranges the solutions in terms of the user's preferred style (query similarity or solution proficiency) and displays the solutions according to the "Show_Rate." Second, the Observer passes the user query to the Query Parser, and simultaneously collects the interaction information and related feedback from the user. The interaction information contains user preferred query mode, recommendation mode, solution presentation mode, and FAQ clicking behavior, while the related feedback contains user satisfaction degree and comprehension degree about each FAQ solution. The User Model Manager needs both interaction information and related feedback to properly update user models and stereotypes. The satisfaction degree in related feedback can also be passed to the Proxy Agent for tuning the solution search mechanism (Yang, 2006).

Finally, the Assistant provides proper assistance and guidance to help the user query process. First, the ontology concepts are structured and presented as a tree so that the users who are not familiar with the domain can check on the tree and learn proper terms to enter their queries. We also rank all ontology concepts by their probabilities and display them in a keyword list. When the user enters a query at the input area, the Assistant will automatically "scroll" the content of the keyword list to those terms related to the input keywords. Fig. 8 illustrates an example of this automatic keyword scrolling mechanism. If the displayed terms of the list contain a concept that the user wants to enter, he can double-click the terms into the input area, e.g., "華碩" (ASUS) at step 2 of Fig. 8. In addition to the keyword-oriented query mode, the Assistant also provides lists of question types and operation types to help question type-oriented or operation type-oriented search. The user can use one, two, or all of these three mechanisms to help form his/her query in order to convey his/her intention to the system.

3.2.2. Query Parser

The Query Parser pre-processes the user query by performing Chinese word segmentation, correction on word segmentation, fault correction on homophonous or multiple words, and term standardization. It then applies the template-based pattern matching technique to analyze the user query, extract the user intention and focus, select best-matched query templates as shown in Fig. 9c (Yang et al., 2004), and trims any irrelevant keywords in accord with the templates. Finally, it transforms the user query into the internal query format and then passes the query to the Proxy Agent for retrieving proper solutions. Fig. 10 shows the flow chart of user query processing. We decided to use template matching for query processing because it is an easy and efficient way to handle pseudo-natural language processing and the result is usually acceptable on a focused domain.

Given a user query in Chinese, we segment the query using MMSEG. The results of segmentation were not good, for the predefined MMSEG word corpus contains insufficient terms of the PC domain. For example, it does not contain the keywords "華碩" or "AGP4X", and returns wrong word segmentation like "華,” “碩”, "AGP", and “4X". The step of query pruning can easily fix this by using the ontology as a second word corpus to bring those mis-segmented words back. It also performs fault correction on homophonous or multiple words using the ontology and homophone debug base (Chiu, 2003).

The step of query standardization is responsible for replacing the terms used in the user query with the canonical terms in the ontology and intention word base. The original terms and the corresponding canonical terms will then be stored in the terminology table for solution presentation personalization. Finally, we label those recognized keywords by symbol "K" and intention words by symbol "I". The rest are regarded as stop words and removed from the query. Now, if the user is using the keyword mode, we directly jump to the step of query formulation. Otherwise, we use template-based pattern matching to analyze the natural language input.

The step of pattern match is responsible for identifying the semantic pattern associated with the user query. Using the pre-constructed query templates in the template base, we can compare the user query with the query templates and select the best-matched one to identify user intention and focus. Fig. 11 shows the algorithm of fast selecting possibly matched templates, Fig. 12 describes the algorithm which finds out all patterns matched with the user query, and Fig. 13 removes those matched patterns that are generalization of some other matched patterns.

Now we explain how template matching is working by the following user query: “Could Asus K7V motherboard support a CPU
over 1 GHz?" Interface Agent first applies MMSEG to obtain the following list of keywords from the user query: could, support, 1 GHz, K7V, motherboard, Asus, CPU. The "could" query type in the intention type hierarchy is then followed to retrieve corresponding query templates. Table 6 illustrates the only one corresponding query template, which contains two patterns, namely, (could support S1) and (could support S2). We find the second pattern matches the user query and can be selected to transform the query into an internal form by query formulation (step 6) as shown in Table 8a. Note that there may be more than two patterns to become candidates for a given query. In this case, the Query Parser will prompt the user to confirm his/her intent (step 5), as illustrated in Fig. 9c. If the user says "No", which means the pattern matching results is not the true intention of the user, he/she is allowed to modify the matched result or change to the keyword mode for placing query.

The purpose of keyword trimming is to remove irrelevant keywords from the user query; irrelevant keywords sometimes cause adversarial effects on FAQ retrieval. Query Parser uses trimming rules, as shown in Table 9, to prune these keywords. For examples, in Table 8a, "motherboard" is trimmed and replaced with "K7V", since the latter is an instance of the former and can subsume the

---

**Table 8** Internal user query and keyword trimming

<table>
<thead>
<tr>
<th>(a) Internal query form before keyword trimming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query type</td>
</tr>
<tr>
<td>Operation type</td>
</tr>
<tr>
<td>Keywords</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(b) Internal query form after keyword trimming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query type</td>
</tr>
<tr>
<td>Operation Type</td>
</tr>
<tr>
<td>Keywords</td>
</tr>
</tbody>
</table>

---

**Table 9** Examples of trimming rules

<table>
<thead>
<tr>
<th>Rule no.</th>
<th>Rule description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A super-class can be replaced with its sub-class</td>
<td>&quot;interface card&quot; =&gt; &quot;Sound card&quot;</td>
</tr>
<tr>
<td>2</td>
<td>A class can be replaced with its instance</td>
<td>&quot;CPU&quot; =&gt; &quot;P4&quot;</td>
</tr>
<tr>
<td>3</td>
<td>A slot value referring to some instance of a class can be replaced with the instance</td>
<td>&quot;Microsoft&quot; =&gt; &quot;Windows 2000&quot;</td>
</tr>
</tbody>
</table>
former according to Trimming Rule 2. Table 8b shows the result of the user query after keyword trimming, which now contains only three keywords, namely, 1 GHz, K7V, and CPU.

3.2.3. Web Page Processor
The Web Page Processor receives a list of retrieved solutions, which contains one or more FAQs matched with the user query from the Proxy Agent, each represented as Table 10, and retrieves and caches the solution webpages according to the FAQ_URLs. It follows to pre-process those webpages for subsequent customization process, including URL transformation, keyword standardization, and keyword marking. The URL transformation changes all hyperlinks to point toward the cached server. The keyword standardization transforms all terms in the webpage content into ontology vocabularies. The keyword labeling marks the keywords appearing in the webpages by boldfaced (B) Keyword

3.2.4. Scorer
Each FAQ is a short document; concepts involved in FAQs are in general more focused. In other words, the topic (or concept) is much clearer and professional. The question part of an FAQ is even more pointed about what concepts are involved. Knowing this property, we can use the keywords appearing in the question part of an FAQ to represent its topic. Basically, we use the table of domain proficiency to calculate a proficiency degree for each FAQ by calculating the proficient concepts appearing in the question part of the FAQ, detailed as shown in Fig. 14.

3.2.5. Personalizer
The Personalizer replaces the terms used in the solution FAQs with the terms in the user’s terminology table, collected by the Query Parser, for improving the solution readability.

3.2.6. User model manager
The first task of the User Model Manager is to create an initial user model for a new user. To do this, we pre-defined several questions for each concept in the domain ontology, for example, “Do you know a CPU contains a floating co-processor?”, “Do you know the concept of 1 GB = 1000 MB in specifying the capacity of a hard disk?”, etc. The difficulty degrees of the questions are proportional to the hierarchy depth of the concepts in the ontology. When a new user logs on the system, the Manager randomly selects questions from the ontology. The user either answers a YES or NO to each question. The answers are collected and weighted according to the respective degrees and passed to the Manager, which then calculates a proficiency score for the user according to the percentage of correctness of his/her responses to the questions and accordingly instantiates a proper user stereotype as the user model for the user.

The second task is to update user models. Here we use the interaction information and user feedback collected by the Interaction Agent in each interaction session or query session. An interaction session is defined as the time period from the time point the user logs in to when he logs out, while a query session is defined as the time period from when the user gives a query up to when he gets the answers and completes the feedback. An interaction session may contain several query sessions. After a query session is completed, we immediately update the interaction preference and solution presentation of the user model. Specifically, the user’s query mode and solution presentation mode in this query session are remembered in both time windows, and the statistics of the preference change for each mode is calculated accordingly, which will be used to adapt the Interaction Agent on the next query session. Fig. 15 illustrates the algorithm to update the Show_Rate of the similarity mode. The algorithm uses the ratio of the number of user selected FAQs and that of the displayed FAQs to update the show rate; the algorithm to update the Show_Rate of the proficiency mode is similar.

In addition, each user will be asked to evaluate each solution FAQ in terms of the following five levels of understanding, namely, very familiar, familiar, average, not familiar, very not familiar. This provides an explicit feedback and we can use it to update his/her domain proficiency table.

Table 10
Format of retrieved FAQ.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAQ_No.</td>
<td>FAQ's identification</td>
</tr>
<tr>
<td>FAQ_Question</td>
<td>Question part of FAQ</td>
</tr>
<tr>
<td>FAQ_Answer</td>
<td>Answer part of FAQ</td>
</tr>
<tr>
<td>FAQ_Similarity</td>
<td>Similarity degree of the FAQ met with the user query</td>
</tr>
<tr>
<td>FAQ_URL</td>
<td>Source or related URL of the FAQ</td>
</tr>
</tbody>
</table>

Fig. 14. Proficiency degree calculation algorithm.

\[
\text{Proficiency}_{\text{Concept} j} = \frac{T \text{Concept} j \times \text{Learning rate}}{\text{Understanding Level}}
\]

\[
T \text{Concept} j = \begin{cases} 
0, & \text{Concept} j \text{ doesn't appear in } F_i \\
1, & \text{Concept} j \text{ appears in } F_i 
\end{cases},
\]

\[
\text{Understanding Level} = \begin{cases} 
0, & \text{user rates "average" for } F_i \\
1, & \text{user rates "not familiar" for } F_i \\
2, & \text{user rates "very not familiar" for } F_i
\end{cases}
\]

Fig. 15. Algorithm to update show rate in similarity mode.

Please cite this article in press as: Yang, S.-Y., Developing of an ontological interface agent with template-based linguistic ..., Expert Systems with Applications (2008), doi:10.1016/j.eswa.2008.03.011
window. At the same time, we add the query and FAQ-selection records of the user into the query history and selection history of his/her user model.

The third task of the User Model Manager is to update user stereotypes. This happens when a sufficient number of user models in a stereotype has undergone changes. First, we need to reflect these changes to stereotypes by re-clustering all affected user models, as shown in Fig. 17, and then re-calculates all parameters in each stereotype, an example as shown in Fig. 18.

3.2.7. Recommender

The Recommender uses the following three policies to recommend information. (1) High hit FAQs. It recommends the first N solution FAQs according to their selection counts from all users in the same group within a time window. (2) Hot topic FAQs. It recommends the first N solution FAQs according to their popularity, calculated as statistics on keywords appearing in the query histories of the same group users within a time window. The algorithm does the hot degree calculation as shown in Fig. 19. (3) Collaborative recommendation. It refers to the user’s selection histories of the same group to provide solution recommendation. The basic idea is this. If user A and user B are in the same group and the first n interaction sessions of user A are the same as those of user B, then we can recommend the highest-rated FAQs in the (n + 1)th session of user A for user B, detailed algorithm as shown in Fig. 20.

4. Demonstrations and evaluations

Our Interface Agent was developed using the Web-Based client-server architecture. On the client site, we use JSP (Java Server Page) and Java Applet for easy interacting with users, as well as observing and recording user’s behavior. On the server site, we use Java and Java Servlet, under Apache Tomcat 4.0 Web Server and MS SQL2000 Server running Microsoft Windows XP. In this section, we first demonstrate the developed Interface Agent, and then report how better it performs.

4.1. System demonstrations

When a new user enters the system, the user is registered by the Agent as shown in Fig. 21. At the same time, a questionnaire is produced by the Agent for evaluating the user’s domain proficiency. His/Her answers are then collected and calculated in order to help build an initial user model for the new user.

Fig. 17. Algorithm to re-cluster all user groups.

For each user: in each user proficiency group

\[
\text{Proficiency}_{\text{proficiency}}(\text{user}) = \frac{\sum \text{Proficiency} \times \text{Concept}_{j}}{\text{Number of concepts in Domain Proficiency Table}}
\]

where Concept j : jth Concept in user’s Domain Proficiency Table, if \(0.8 \leq \text{Proficiency}_{\text{proficiency}}(\text{user}) \leq 1.0\), user is reassigned to Expert group, if \(0.6 \leq \text{Proficiency}_{\text{proficiency}}(\text{user}) < 0.8\), user is reassigned to Senior group, if \(0.4 \leq \text{Proficiency}_{\text{proficiency}}(\text{user}) < 0.6\), user is reassigned to Junior group, if \(0.2 \leq \text{Proficiency}_{\text{proficiency}}(\text{user}) < 0.4\), user is reassigned to Novice group, if \(0.0 \leq \text{Proficiency}_{\text{proficiency}}(\text{user}) < 0.2\), user is reassigned to Amateur group.

Fig. 18. Example to update the stereotype of expert.

\[
\text{Score}_{\text{Stereotype}} = \frac{\sum \text{Score}_{\text{Similarity}}(\text{Stereotype}) \times \text{Similarity} \times \text{Score}_{\text{Similarity}}}{\sum \text{Score}_{\text{Similarity}}}
\]

\[
\text{Score}_{\text{Similarity}} = \frac{\sum \text{Score}_{\text{Similarity}}(\text{Stereotype}) \times \text{Similarity}}{\sum \text{Score}_{\text{Similarity}}}
\]

For each Concept j in Domain Proficiency Table (DPT)

\[
\text{Score}_{\text{DPT}} \times \text{Proficiency}(\text{Concept}_{j}) = \frac{\sum \text{Score}_{\text{DPT}} \times \text{Proficiency}(\text{Concept}_{j})}{\sum \text{Score}_{\text{DPT}}}
\]

Fig. 19. Algorithm to calculate the hot degree.

Fig. 20. Algorithm to do the collaborative recommendation.

Our Interface Agent was developed using the Web-Based client-server architecture. On the client site, we use JSP (Java Server Page) and Java Applet for easy interacting with users, as well as observing and recording user’s behavior. On the server site, we use Java and Java Servlet, under Apache Tomcat 4.0 Web Server and MS SQL2000 Server running Microsoft Windows XP. In this section, we first demonstrate the developed Interface Agent, and then report how better it performs.

4.1. System demonstrations

When a new user enters the system, the user is registered by the Agent as shown in Fig. 21. At the same time, a questionnaire is produced by the Agent for evaluating the user’s domain proficiency. His/Her answers are then collected and calculated in order to help build an initial user model for the new user.

Please cite this article in press as: Yang, S.-Y., Developing of an ontological interface agent with template-based linguistic..., Expert Systems with Applications (2008), doi:10.1016/j.eswa.2008.03.011
Now the user can get into the main tableau of our system (Fig 22), which consists of the following three major tab-frames, namely, query interface, solution presentation, and logout. The query interface tab is comprised of the following four frames: user interaction interface, automatic keyword scrolling list, FAQ recommendation list, and PC ontology tree. The user interaction interface contains both keywords and NLP query modes as shown in Fig. 9. The keyword query mode provides the lists of question types and operation types, which allow the users to express their precise intentions. The automatic keyword-scrolling list provides ranked-keyword guidance for user query. A user can browse the PC ontology tree to learn domain knowledge. The FAQ recommendation list provides personalized information recommendations from the system, which contains three modes: hit, hot topic, and collaboration. When the user clicked a mode, the corresponding popup window is produced by the system.

The solution presentation tab is illustrated in Fig 23. It pre-selects the solutions ranking method according to the user’s preference and hides part of solutions according to his/her Show_Rate for reducing the cognitive loading of the user. The user can switch the solution ranking method between similarity ranking and proficiency ranking. The user can click the question part of an FAQ (Fig. 24) for displaying its content or giving it a feedback, which contains the satisfaction degree and comprehension degree. Fig. 25 illustrates the window before system logout, which asks the user to fill a questionnaire for statistics to help further system improvement.

4.2. System evaluations

The evaluation of the overall performance of our system involves lots of manpower and is time-consuming. Here, we focus on the performance evaluation of the most important module, i.e., the Query Parser. Our philosophy is that if it can precisely parse user queries and extract both true query intention and focus from them, then we can effectively improve the quality of the retrieved. First, we have done two experiments in evaluating how the user query processing performs under the support of query templates and domain ontology. Recall that this processing employs the technique of template-based pattern matching mechanism to understand user queries and the templates were manually constructed from 1215 FAQs. In the first experiment, we use these same FAQs for testing queries, in order to verify whether any conflicts exist within the query. Table 11 illustrates the experimental results, where only 33 queries match with more than one query patterns and result in confusion of query intention, called “error” in the table. These errors may be corrected by the user. The experiment shows the effectiveness rate of the constructed query templates.
reaches 97.28%, which implies the template base can be used as an effective knowledge base to do natural language query processing.

Our second experiment is to learn how well this process understands new queries. First, we collected in total 143 new FAQs, different from the FAQs collected for constructing the query templates, from four famous motherboard factories in Taiwan, including ASUS (<http://www.asus.com/>), SIS (<http://www.sis.com/>), Infineon (<http://www.siemens.com/techtalk/techtalk.html>), and GIGABYTE (<http://www.giga-byte.com/>). Then, we used the question parts of those FAQs for testing queries, which test how well this process performs. Our experiments show that we can precisely extract true query intentions and focuses from 112 FAQs. The rest of 31 FAQs contain up to three or more sentences in queries, which explain why we failed to understand them. In summary, 78.3% (112/143) of the new queries can be successfully understood.

Finally, Table 12 shows the comparison of user satisfaction of our systemic prototype against other search engines. In the table, $S_T$ for Satisfaction of testers, represents the average of satisfaction responses from 10 ordinary users, while $S_R$ for Satisfaction of experts, represents that of satisfaction responses from 10 experts.

<table>
<thead>
<tr>
<th>K_WORD METHOD</th>
<th>CPU ($S_S/S_T$) (%)</th>
<th>MOTHERBOARD ($S_S/S_R$) (%)</th>
<th>MEMORY ($S_S/S_T$) (%)</th>
<th>AVERAGE ($S_S/S_T$) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AltaVista</td>
<td>63/61</td>
<td>77/78</td>
<td>30/21</td>
<td>57/53</td>
</tr>
<tr>
<td>Excite</td>
<td>66/62</td>
<td>81/81</td>
<td>50/24</td>
<td>66/56</td>
</tr>
<tr>
<td>Google</td>
<td>66/64</td>
<td>81/80</td>
<td>38/21</td>
<td>62/55</td>
</tr>
<tr>
<td>HotBot</td>
<td>65/63</td>
<td>78/76</td>
<td>62/31</td>
<td>70/57</td>
</tr>
<tr>
<td>Infineon</td>
<td>69/70</td>
<td>71/70</td>
<td>40/28</td>
<td>63/56</td>
</tr>
<tr>
<td>Lycos</td>
<td>64/67</td>
<td>77/76</td>
<td>36/20</td>
<td>59/54</td>
</tr>
<tr>
<td>Yahoo</td>
<td>67/69</td>
<td>71/78</td>
<td>38/17</td>
<td>61/52</td>
</tr>
<tr>
<td>Our approach</td>
<td>78/69</td>
<td>84/78</td>
<td>45/32</td>
<td>69/60</td>
</tr>
</tbody>
</table>

5. Related works and comparisons

The work of Lee (2000) presents a user query system consisting of an intention part and a keywords part. With the help of syntax and parts-of-speech (POS) analysis, he constructs a syntax grammar from collected FAQs, and accordingly offers the capability of intention extraction from queries. He also extracts the keywords from queries through a sifting process on POS and stop words. Finally, he employs the semantic comparison technique of downward recursion on a parse tree to calculate the similarity degree of intention parts between the user query and FAQs, and then uses the concept of space mode (Salton et al., 1975) to calculate the vector similarity degree of the keyword parts between the user query and FAQs for finding out best-matched FAQs.

The work of OuYang (2000) classifies pre-collected FAQs according to the business types of ChungHwa Telecom (<http://www.cht.com.tw/CHTFinalE/Web/>). He employs the technique of TFIDF (Term Frequency and Inversed Document Frequency) to calculate the weights of individual keywords and intention words, and accordingly select representative keywords and intention words from them to work as the index to each individual class.

A user query, he first determines the class of the user query according to the keywords and intention words, and then calculates the similarity degree between the user query and FAQs in the related classes.

The natural language processing technique was not used in the work of Sneiders et al. (1999) for analyzing user queries. It, instead, was applied to analyze the FAQs stored in the database long before any user queries are submitted, where each FAQ is associated with a required, optional, irrelevant, or forbidden keyword to help subsequent prioritized keyword matching. By this way, the work of FAQ retrieval can be reduced to keyword matching without inference.

Razmerita, Angehrn, and Maedche (2003) presents a generic ontology-based user modeling architecture, (OntoBum), applied in the context of a Knowledge Management System (KMS). The proposed user modeling system relies on a user ontology, using Semantic Web technologies, based on the IMS LIP specifications, and it is integrated in an ontology-based KMS called Ontologing.

Degemmis, Lichelli, Lops, and Semeraro (2004) presents the Profile Extractor, a personalization component based on machine learning techniques, which allows for the discovery of preferences and interests of users that have access to a website. Galassi, Giordana, Sahtia, and Botta (2005) also presents a method for automatically constructing a sophisticated user/process profile from traces of user/process behavior, which is encoded by means of a Hierarchical Hidden Markov Model (HHMM). Finally, Hsu and Ho (1999) propose an intelligent interface agent to acquire patient data with medicine-related common sense reasoning.

In summary, the work of OuYang determines the user query intention according to keywords and intention words appearing in query: while the work of Sneiders uses both similarity degrees on intention words and keywords for solution searching and selection. Both approaches only consider the comparison between words and skip the problem of word ambiguity: e.g., two sentences with the same intention words may not have the same intention. The work of Lee uses the analysis of syntax and POS to extract query intention, which is a hard job with Chinese query, because solving the ambiguity either explicit or implicit meanings of Chinese words, especially in query analysis on long sentences or sentences with complex syntax, is not at all a trivial task. In this paper, we integrated several interesting techniques including user modeling, domain ontology, and template-based linguistic processing to effectively tackle the above annoying problems, just like Razmerita et al., 2003 in which differently associated with ontology and user modeling and especially, (Paraiso & Barthes, 2006) highlights the role of ontologies for semantic interpretation. In addition, (Degemmis et al., 2004 & Galassi et al., 2005) propose different learning techniques for processing usage patterns and user profiles. The automatic processing feature, supported by HHMM and unsupervised learning and common sense reasoning techniques, respectively, provides another level of automation in interaction mechanism and deserves more attention.

6. Discussions and future work

We have developed an Interface Agent to work as an assistant between the users and FAQ systems, which is different from system architecture and implementation over our previous work (Yang et al., 2004). It is also used to retrieve FAQs on the domain of PC. We integrated several interesting techniques including domain ontology, user modeling, and template-based linguistic processing to effectively tackle the problems associated with traditional FAQ retrieval systems. Specifically, we have solved the following issues. Firstly, our ontological interface agent can truly learn a user’s specialty in order to build a proper user model for him/her. Secondly, the domain ontology can efficiently and effectively help in
establishing user models, analyzing user query, and assisting and guiding interface usage. Finally, the intention and focus of a user can be correctly extracted by the agent. In short, our work features an template-based linguistic processing technique for developing ontological interface agents; a nature language query mode, along with an improved keyword-based query mode; and an assistance and guidance for human–machine interaction. Our preliminary experimentation demonstrates that user intention and focus of up to eighty percent of the user queries can be correctly understood by the system, and accordingly provides the query solutions with higher user satisfaction.

Most of our current experiments are on the performance test of the Query Parser. We are unable to do experiments on or comparisons of how good the Interface Agent is at capturing all interaction information/intention of a user. Our difficulties are summarized below: (1) To our knowledge, none of current interface systems adopt a similar approach as ours in the sense that none of them are relying on ontology as heavily as our system to support user interaction. It is thus rather hard for us to do a fair and convincing comparison. (2) Our ontology construction is based on a set of pre-collected webpages on a specific domain; it is hard to evaluate how critical this pre-collection process is to the nature of different domains. We are planning to employ the technique of automatic ontology evolution, for example, cooperated with data mining technology for discovering useful information and generating desired knowledge that support ontology construction (Wang, Lu, & Zhang, 2007), to help study the robustness of our ontology. Finally, in the future, not only will we employ the techniques of machine learning and data mining to automate the construction of the template base, as to the allow system evaluation, but also we are planning to engage the concept of usability evaluation on the domain of human factor engineering to evaluate the performance of the agent.

Acknowledgements

The author would like to thank Yai-Hui Chang and Ying-Hao Chiu for their assistance in system implementation. This work was supported by the National Science Council, ROC, under Grants NSC-89-2213-E-011-059, NSC-89-2218-E-011-014, and NSC-95-2221-E-129-019.

References


Chiu, Y. H. (2003). An interface agent with ontology-supported user models. Master Thesis, Department of Electronic Engineering, National Taiwan University of Science and Technology, Taiwan, ROC.


OuYang, Y. J. (2000). Study and implementation of a dialogued-based query system for telecommunication FAQ services. Master Thesis, Department of Computer and Information Science, National Chiao Tung University, Taiwan, ROC.


An Ontological Interface Agent for FAQ Query Processing

SHENG-YUAN YANG
Dept. of Computer and Communication Engineering
St. John’s University
499, Sec. 4, TamKing Rd., Tamsui, Taipei County 251
TAIWAN
ysy@mail.sju.edu.tw http://mail.sju.edu.tw/~ysy

Abstract: - In this paper, we describe an Interface Agent which works as an assistant between the users and FAQ systems to retrieve FAQs on the domain of Personal Computer. It integrates several interesting techniques including domain ontology, user modeling, and template-based linguistic processing to effectively tackle the problems associated with traditional FAQ retrieval systems. Specifically, we address how ontology helps interface agents to provide better FAQ services and describe related algorithms in details. Our work features an ontology-supported, template-based user modeling technique for developing interface agents. Our preliminary experimentation demonstrates that user intention and focus of up to eighty percent of the users' queries can be correctly understood by the system, and accordingly provides the query solutions with higher user satisfaction.

Key-words: - Ontology, Template-based Processing, User Modeling, Interface Agents.

1 Introduction

In this information-exploding era, the Internet affects people’s life style in terms of how people acquire, present, and exchange information. Especially the use of the World Wide Web has been leading to a large increase in the number of people who access FAQ knowledge bases to find answers to their questions [10]. As the techniques of Information Retrieval [6,7] matured, a variety of information retrieval systems have been developed, e.g., Search engines, Web portals, etc., to help search on the Web. How to search is no longer a problem. The problem now comes from the results from these information retrieval systems which contain some much information that overwhelms the users. Therefore, how to improve traditional information retrieval systems to provide search results which can better meet the user requirements so as to reduce his cognitive loading is an important issue in current research [2].

We have proposed an FAQ-master as an intelligent Web information aggregation system, which provides intelligent information retrieval, filtering, and aggregation services [12,17]. Fig. 1 illustrates the system architecture of FAQ-master. The Interface Agent captures user intention through an adaptive human-machine interaction interface with the help of ontology-directed and template-based user models [18]. The Search Agent performs in-time, user-oriented, and domain-related Web information retrieval with the help of ontology-supported website models [19]. The Answerer Agent works as a back end process to perform ontology-directed information aggregation from the webpages collected by the Search Agent [14,20]. Finally, the Proxy Agent works as an ontology-enhanced intelligent proxy mechanism to share most query loading with the Answerer Agent [15,16,20].

This paper discusses the Interface Agent focusing on how it captures true user’s intention and accordingly provides high-quality FAQ answers. The agent features ontology-based representation of domain knowledge, flexible interaction interface, and personalized information filtering and display. Our preliminary experimentation demonstrates that the intention and focus of up to eighty percent of the users’ queries can be correctly understood by the system, and accordingly provides the query solutions with higher user satisfaction. The Personal Computer (PC) domain is chosen as the target application of our Interface Agent and will be used for explanation in the remaining sections.

2 Fundamental Techniques

2.1 Domain Ontology and Services

The concept of ontology in artificial intelligence refers to knowledge representation for domain-specific contents [1]. It has been advocated as an important tool to support knowledge sharing and reusing in developing intelligent systems. Although development of an ontology for a specific domain is not yet an engineering process, we have outlined a procedure for this in [11] from how the process was conducted in existent systems. By following the procedure we developed an ontology for the PC domain in Chinese using Protége 2000 [4], but was changed to English here for easy explanation, as the fundamental background knowledge for the system.

Fig. 2 shows part of the ontology taxonomy. The taxonomy represents relevant PC concepts as classes and their parent-child relationships as isa links, which allow inheritance of features from parent classes to child classes. Fig. 3 exemplifies the detailed ontology of the concept of CPU. In the figure, the root node uses various fields to define the semantics of the CPU class, each field representing an attribute of “CPU”, e.g., interface, provider, synonym, etc. The nodes at the lower level represent various CPU instances,
which capture real world data. The arrow line with term “to” means the instance of relationship. The complete PC ontology can be referenced from the Protégé Ontology Library at Stanford Website (http://protege.stanford.edu/download/download.html).

Fig. 2 Part of PC ontology taxonomy

We also developed a Problem ontology to deal with query questions. Fig. 4 illustrates part of the Problem ontology, which contains query type and operation type. Together they imply the semantics of a question. Finally, we use Protégé’s APIs to develop a set of ontology services, which provide primitive functions to support the application of the ontologies. The ontology services currently available include transforming query terms into canonical ontology terms, finding definitions of specific terms in ontology, finding relationships among terms, finding compatible and/or conflicting terms against a specific term, etc.

Fig. 3 Ontology of the concept of CPU

2.2 Ontological Query Templates

To build the query templates, we have collected in total 1215 FAQs from the FAQ website of six famous motherboard factories in Taiwan and used them as the reference materials for query template construction. Currently, we only take care of the user query with one intention word and at most three sentences. These FAQs were analyzed and categorized into six types of questions. For each type of question, we further identified several intention types according to its operations. Finally, we define a query pattern for each intention type.

Table 1 Examples of query patterns

<table>
<thead>
<tr>
<th>Question Type</th>
<th>Operation Type</th>
<th>Intention Type</th>
<th>Query Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{What} )</td>
<td>( \text{File} )</td>
<td>( \text{ANA_CAN_SUPPORT} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
</tr>
<tr>
<td>( \text{What} )</td>
<td>( \text{Setup} )</td>
<td>( \text{ANA_CAN_SUPPLY} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
</tr>
<tr>
<td>( \text{When} )</td>
<td>( \text{File} )</td>
<td>( \text{ANA_CAN_SET} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
</tr>
<tr>
<td>( \text{Where} )</td>
<td>( \text{Setup} )</td>
<td>( \text{ANA_CAN_SUPP} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
</tr>
<tr>
<td>( \text{How} )</td>
<td>( \text{Setup} )</td>
<td>( \text{ANA_CAN_SET} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
</tr>
<tr>
<td>( \text{Why} )</td>
<td>( \text{Print} )</td>
<td>( \text{ANA_CAN_SUPPLY} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
</tr>
</tbody>
</table>

Table 2 Query template for the ANA_CAN_SUPPORT intention type

<table>
<thead>
<tr>
<th>Intention Type</th>
<th>Question Type</th>
<th>Operation Type</th>
<th>Query Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{ANA_CAN_SUPPORT} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
<td>( \text{ANA_CAN_SUPPLY} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
</tr>
<tr>
<td>( \text{ANA_CAN_SUPPLY} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
<td>( \text{ANA_CAN_SET} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
</tr>
<tr>
<td>( \text{ANA_CAN_SUPP} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
<td>( \text{ANA_CAN_SUPP} )</td>
<td>( \text{WIN_M} ) and ( \text{WIN_98SE} )</td>
</tr>
</tbody>
</table>

3 System Architecture

3.1 User Modeling

A user model contains interaction preference, solution presentation, domain proficiency, terminology table, query history, selection history, and user feedback, as shown in Fig. 6.

The interaction preference is responsible for recording user’s preferred interface, e.g., favorite query mode, favorite recommendation mode, etc. When the user logs on the system, the system can select a proper user interface according to this preference. We provide two modes, either through keywords or natural language input. We provide three recommendation modes according to hit rates, hot topics, or collaborative learning. We record recent user’s preferences in a time window, and accordingly determine the next interaction style.

The solution presentation is responsible for recording solution ranking preferences of the user. We provide two types of ranking, either according to the degree of similarity...
between the proposed solutions and the user query, or according to user’s proficiency about the solutions. In addition, we use a Show_Rate parameter to control how many items of solutions for display each time, in order to reduce information overloading problem.

The domain proficiency factor describes how familiar the user is with the domain. By associating a proficiency degree with each ontology concept, we can construct a table, which contains a set of <concept proficiency-degree> pairs, as his domain proficiency. Thus, during the decision of solution representation, we can calculate the user’s proficiency degree on solutions using the table, and accordingly only show his most familiar part of solutions and hide the rest for advanced requests. To solve the problem of different terminologies to be used by different users, we include a terminology table to record this terminology difference. We can use the table to replace the terms used in the proposed solutions with the user favorite terms during solutions representation to help him better comprehend the solutions.

Finally, we record the user’s query history as well as FAQ selection history and corresponding user feedback in each query session in the Interaction history, in order to support collaborative recommendation. The user feedback is a complicated factor. We remember both explicit user feedback in the selection history and implicit user feedback, which includes query time, time of FAQ click, sequence of FAQ clicks, sequence of clicked hyperlinks, etc.

In order to quickly build an initial user model for a new user, we pre-defined five stereotypes [5], namely, expert, senior, junior, novice, and amateur [11], to represent different user group’s characteristics. This approach is based on the idea that the same group of user tends to exhibit the same behavior and requires the same information. Fig. 7 illustrates an example user stereotype. When a new user enters the system, he is asked to complete a questionnaire, which is used by the system to determine his domain proficiency, and accordingly select a user stereotype to generate an initial user model to him. However, the initial user model constructed from the stereotype may be too generic or imprecise. It will be refined to reflect the specific user’s real intent after the system has experiences with his query history, FAQ-selection history and feedback, and implicit feedback [2].

3.2 System Overview

Fig. 8 shows the architecture of the Interface Agent. The Interaction Agent provides a personalization interaction, assistance, and recommendation interface for the user according to his user model, records interaction information and related feedback in the user model, and helps the User Model Manager and Proxy Agent to update the user model. The Query Parser processes the user queries by first segmenting word, removing conflicting words, and standardizing terms, followed by the recording of the user’s terminologies in the terminology table of the user model. It finally applies the template matching technique to select best-matched query templates, and accordingly transforms the query into an internal query for the Proxy Agent to search for solutions and collect them into a list of FAQs, which each containing a corresponding URL. The Web Page Processor pre-downloads FAQ-relevant webpages and performs some pre-processing tasks, including labeling keywords for subsequent processing. The Scorer calculates the user’s proficiency degree for each FAQ in the FAQ list according to the terminology table in his user model. The Personalizer then produces personalized query solutions according to the terminology table. The User Model Manager is responsible for quickly building an initial user model for a new user using the technique of user stereotyping as well as updating the user models and stereotypes to dynamically reflect the changes of user behavior. The Recommender is responsible for recommending information for the user based on hit count, hot topics, or group’s interests when a similar interaction history is detected.

3.2.1 Interaction Agent

The Interaction Agent consists of the following three components: Adapter, Observor and Assistant. First, the Adapter constructs best interaction interfaces according to user’s favorite query and recommendation modes. It is also responsible for presenting to the user the list of FAQ solutions (from the Personalizer) or recommendation information (from the Recommender). During solution representation, it arranges the solutions in terms of the user’s preferred style (query similarity or solution proficiency) and displays the solutions according to the “Show_Rate.” Second, the Observer passes the user query to the Query Parser, and simultaneously collects the interaction information and related feedback from the user. The interaction information contains user preferred query mode, recommendation mode, solution presentation mode, and FAQ clicking behavior, while the related feedback contains user satisfaction degree and comprehension degree about each FAQ solution. The User Model Manager needs both interaction information and related feedback to properly update user models and stereotypes. The satisfaction degree in related feedback can also be passed to the Proxy Agent for tuning the solution search mechanism [20].

Finally, the Assistant provides proper assistance and guidance to help the user query process. First, the ontology concepts are structured and presented as a tree so that the users who are not familiar with the domain can check on the
3.2.2 Query Parser

tree and learn proper terms to enter their queries. We also rank all ontology concepts by their probabilities and display them in a keyword list. When the user enters a query at the input area, the Assistant will automatically “scroll” the content of the keyword list to those terms related to the input keywords. If the displayed terms of the list contain a concept that the user wants to enter, he can double-click the terms into the input area, e.g., “華碩” (ASUS) at step 2 of Fig. 9. In addition to the keyword-oriented query mode, the Assistant also provides lists of question types and operation types to help question type-oriented or operation type-oriented search. The user can use one, two, or all of these three mechanisms to help form his query in order to convey his intention to the system.

Fig. 9 Examples of automatic keyword scrolling mechanism

3.2.2 Query Parser

The Query Parser pre-processes the user query by performing Chinese word segmentation, correction on word segmentation, fault correction on homophonous or multiple words, and term standardization. It then employs template-based pattern matching to analyze the user query and extract the user intention and focus. Finally, it transforms the user query into the internal query format and then passes the query to the Proxy Agent for retrieving proper solutions [20]. Fig. 10 shows the flow chart of the Query Parser. Detailed explanation follows.

Given a user query in Chinese, we segment the query using MMSEG [9]. The results of segmentation were not good, for the predefined MMSEG word corpus contains insufficient terms of the PC domain. For example, it does not contain the keywords “華碩” or “AGP4X”, and returns wrong word segmentation like “華”, “碩”, “AGP”, and “4X”. The step of query pruning can easily fix this by using the ontology as a second word corpus to bring those mis-segmented words back. It also performs fault correction on homophonous or multiple words using the ontology and homophone debug base [2].

The step of query standardization is responsible for replacing the terms used in the user query with the canonical terms in the ontology and intention word base. The original terms and the corresponding canonical terms will then be stored in the terminology table for solution presentation personalization. Finally, we label those recognized keywords by symbol “K” and intention words by symbol “I.” The rest are regarded as stop words and removed from the query. Now, if the user is using the keyword mode, we directly jump to the step of query formulation. Otherwise, we use template-based pattern matching to analyze the natural language input.

The step of pattern match is responsible for identifying the semantic pattern associated with the user query. Using the pre-constructed query templates in the template base, we can compare the user query with the query templates and select the best-matched one to identify user intention and focus. Fig. 11 shows the algorithm of fast selecting possibly matched templates. Fig. 12 describes the algorithm which finds out all patterns matched with the user query, and Fig. 13 removes those matched patterns that are generalization of some other matched patterns.

Fig. 10 Flow chart of the Query Parser

Fig. 11 Query template selection algorithm

Fig. 12 Pattern match algorithm

Fig. 13 Query pattern removal algorithm

Take the following query as an example: “華碩 K7V 主機板是否支援 1GHz 以上的中央處理器呢？”, which means “Could Asus K7V motherboard support a CPU over 1GHz?” Table 2 illustrates a query template example, which contains two patterns, namely, <S1 是否支援 S2> and <S1 是否支援 S1>. We find the second pattern matches the user query and can be selected to transform the query into an internal form by query formulation (step 6) as shown in Table 3. Note that there may be more than two patterns to become candidates for
a given query. In this case, the Query Parser will prompt the user to confirm his intent (step 5). If the user says “No”, which means the pattern matching results is not the true intention of the user, he is allowed to modify the matched result or change to the keyword mode for placing query.

Table 3 Internal form

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAQ No.</td>
<td>FAQ’s identification</td>
</tr>
<tr>
<td>FAQ Question</td>
<td>Question part of FAQ</td>
</tr>
<tr>
<td>FAQ Answer</td>
<td>Answer part of FAQ</td>
</tr>
<tr>
<td>FAQ Simplicity</td>
<td>Simplicity degree of the FAQ and the web page</td>
</tr>
<tr>
<td>FAQ URL</td>
<td>Source or related URL of the FAQ</td>
</tr>
</tbody>
</table>

3.2.3 Web Page Processor

The Web Page Processor receives a list of retrieved solutions, which contains one or more FAQs matched with the user query from the Proxy Agent, each represented as Table 4, and retrieves and caches the solution webpages according to the FAQ URLs. It follows to pre-process those webpages for subsequent customization process, including URL transformation, keyword standardization, and keyword marking. The URL transformation changes all hyperlinks to point toward the cached server. The keyword standardization transforms all terms in the webpage content into ontology vocabularies. The keyword labeling marks the keywords appearing in the webpages by boldfaced <B>Keyword</B> to facilitate subsequent keywords processing webpage readability.

Table 4 Format of retrieved FAQ

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAQ No.</td>
<td>FAQ’s identification</td>
</tr>
<tr>
<td>FAQ Question</td>
<td>Question part of FAQ</td>
</tr>
<tr>
<td>FAQ Answer</td>
<td>Answer part of FAQ</td>
</tr>
<tr>
<td>FAQ Simplicity</td>
<td>Simplicity degree of the FAQ and the web page</td>
</tr>
<tr>
<td>FAQ URL</td>
<td>Source or related URL of the FAQ</td>
</tr>
</tbody>
</table>

3.2.4 Scorer

Each FAQ is a short document; concepts involved in FAQs are in general more focused. In other words, the topic (or concept) is much clearer and professional. The question part of an FAQ is even more pointed about what concepts are involved. Knowing this property, we can use the keywords appearing in the question part of an FAQ to represent its topic. Basically, we use the table of domain proficiency to calculate a proficiency degree for each FAQ by calculating the proficient concepts appearing in the question part of the FAQ, detailed as shown in Fig. 14.

![Fig. 14 Proficiency degree calculation algorithm](image)

3.2.5 Personalizer

The Personalizer replaces the terms used in the solution FAQs with the terms in the user’s terminology table, collected by the Query Parser, for improving the solution readability.

3.2.6 User Model Manager

The first task of the User Model Manager is to create an initial user model for a new user. To do this, we pre-defined several questions for each concept in the domain ontology, for example, “Do you know a CPU contains a floating co-processor?”, “Do you know the concept of 1GB=1000MB in specifying the capacity of a hard disk?”, etc. The difficulty degrees of the questions are proportional to the hierarchy depth of the concepts in the ontology. When a new user logs on the system, the Manager randomly selects questions from the ontology. The user either answers an YES or NO to each question. The answers are collected and weighted according to the respective degrees and passed to the Manager, which then calculates a proficiency score for the user according to the percentage of correctness of his responses to the questions and accordingly instantiates a proper user stereotype as the user model for the user.

The second task is to update user models. Here we use the interaction information and user feedback collected by the Interaction Agent in each interaction session or query session. An interaction session is defined as the time period from the time point the user logs in up to when he logs out, while a query session is defined as the time period from when the user gives a query up to when he gets the answers and completes the feedback. An interaction session may contain several query sessions. After a query session is completed, we immediately update the interaction preference and solution presentation of the user model. Specifically, the user’s query mode and solution presentation mode in this query session are remembered in both time windows, and the statistics of the preference change for each mode is calculated accordingly, which will be used to adapt the Interaction Agent on the next query session. Fig. 15 illustrates the algorithm to update the Show Rate of the similarity mode. The algorithm uses the ratio of the number of user selected FAQs and that of the displayed FAQs to update the show rate; the algorithm to update the Show Rate of the proficiency mode is similar.

![Fig. 15 Algorithm to update show rate in similarity mode](image)

In addition, each user will be asked to evaluate each solution FAQ in terms of the following five levels of understanding, namely, very familiar, familiar, average, not familiar, very not familiar. This provides an explicit feedback and we can use it to update his domain proficiency table. Fig. 16 shows the updating algorithm. Finally, after each interaction session, we can update the user’s recommendation mode in this session in the respective time window. At the
same time, we add the query and FAQ-selection records of the user into the query history and selection history of his user model.

For each user in each user proficiency group

\[ \sum \text{Proficiency} / \text{Concept} \] 

where Concept is the number of concepts in the domain proficiency table

For Example to update the stereotype of Expert

\[ \text{Show Rate} / \text{User Model} = \frac{\sum \text{U.S.Show Rate Standard Model}}{N} \]

For each concept in the domain proficiency table (DPT)

\[ \text{Show Rate} / \text{Standard Model} = \frac{\sum \text{U.S.Show Rate Standard Model}}{N} \]

4.1 System Demonstration and Experiments

The third task of the User Model Manager is to update user stereotypes. This happens when a sufficient number of user models in a stereotype has undergone changes. First, we need to reflect these changes to stereotypes by re-clustering all affected user models, as shown in Fig. 17, and then re-calculates all parameters in each stereotype, an example as shown in Fig. 18.

3.2.7 Recommender

For each FAQ \( \in \text{FAQ collection} \)

\[ \text{DPT Score} / \text{FAQ} = \frac{\text{number of concepts appearing in query}}{\text{Appearance} / \text{Concept}} \times \text{Weight} / \text{Concept} \] 

where Concept is Ontology,

\[ \text{Appearance} / \text{Concept} = \begin{cases} 1, & \text{Concept appears in query} \\ 0, & \text{Concept doesn't appear in query} \end{cases} \]

\[ \text{Weight} / \text{Concept} = \text{Within a time window, the times that the concept appears in user queries of the same user group} \]

Fig. 18 Example to update the stereotype of Expert

When a new user enters the system, the user is registered by the Agent as shown in Fig. 21. At the same time, a questionnaire is produced by the Agent for evaluating the user’s domain proficiency. His answers are then collected and calculated in order to help build an initial user model for the new user. Now the user can get into the main tableau of our system (Fig. 22), which consists of the following four major tab-frames: namely, query interface, solution presentation, and logout. The query interface tab is comprised of the following four frames: user interaction interface, automatic keyword scrolling list, FAQ recommendation list, and PC ontology tree. The user interaction interface contains both keywords and NLP (Natural Language Processing) query modes. The keyword query mode provides the lists of question types and operation types, which allow the users to express their precise intentions. The automatic keyword scrolling list provides ranked-keyword guidance for user query. A user can browse the PC ontology tree to learn domain knowledge. The FAQ recommendation list provides personalized information recommendations from the system, which contains three modes: hit, hot topic, and collaboration.
When the user clicked a mode, the corresponding popup window is produced by the system.

The solution presentation tab is illustrated in Fig. 23. It pre-selects the solutions ranking method according to the user’s preference and hides part of solutions according to his Show Rate for reducing the cognitive loading of the user. The user can switch the solution ranking method between similarity ranking and proficiency ranking. The user can click the question part of an FAQ (Fig. 24) for displaying its content or giving it a feedback, which contains the satisfaction degree and comprehension degree. Fig. 25 illustrates the window before system logout, which asks the user to fill a questionnaire for statistics to help further system improvement.

4.2 System Evaluation

The evaluation of the overall performance of our system involves lots of manpower and is time-consuming. Here, we focus on the performance evaluation of the most important module, i.e., the Query Parser. Our philosophy is that if it can precisely parse user queries and extract both true query intention and focus from them, then we can effectively improve the quality of the retrieved. Recall that the Query Parser employs the technique of template-based pattern matching mechanism to understand user queries and the templates were manually constructed from 1215 FAQs. In the first experiment, we use the same FAQs for testing queries, in order to verify whether any conflicts exist within the query. Table 5 illustrates the experimental results, where only 33 queries match with more than one query patterns and result in confusion of query intention, called “error” in the table. These errors may be corrected by the user. The experiment shows the effectiveness rate of the constructed query templates reaches 97.28%, which implies the template base can be used as an effective knowledge base to do natural language query processing.

Table 5 Effectiveness of constructed query patterns

<table>
<thead>
<tr>
<th>Template Base</th>
<th>CPU</th>
<th>MOTHERBOARD</th>
<th>MEMORY</th>
<th>AVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SE</td>
<td>ST</td>
<td>SE</td>
<td>ST</td>
<td>Precision Rate</td>
</tr>
<tr>
<td>1215</td>
<td>1182</td>
<td>33</td>
<td></td>
<td>97.28%</td>
</tr>
</tbody>
</table>

Our second experiment is to learn how well the Parser understands new queries. First, we collected in total 143 new FAQs, different from the FAQs collected for constructing the query templates, from four famous motherboard factories in Taiwan, including ASUS, GIGABYTE, MSI, and SIS. We then used the question parts of those FAQs for testing queries, which test how well the Parser performs. Our experiments show that we can precisely extract true query intentions and focuses from 112 FAQs. The rest of 31 FAQs contain up to three or more sentences in queries, which explain why we failed to understand them. In summary, 78.3% (112/143) of the new queries can be successfully understood.

Table 6 User satisfaction evaluation

<table>
<thead>
<tr>
<th>Method</th>
<th>Satisfaction testers</th>
<th>Satisfaction of experts</th>
<th>Satisfaction of ordinary users</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfoSeek</td>
<td>69% / 63%</td>
<td>78% / 76%</td>
<td>71% / 68%</td>
</tr>
<tr>
<td>Lycos</td>
<td>64% / 62%</td>
<td>78% / 76%</td>
<td>70% / 68%</td>
</tr>
<tr>
<td>HotBot</td>
<td>69% / 63%</td>
<td>78% / 76%</td>
<td>70% / 68%</td>
</tr>
<tr>
<td>Excite</td>
<td>66% / 61%</td>
<td>78% / 76%</td>
<td>71% / 68%</td>
</tr>
<tr>
<td>Yahoo</td>
<td>67% / 61%</td>
<td>78% / 76%</td>
<td>72% / 69%</td>
</tr>
<tr>
<td>Lycos</td>
<td>64% / 65%</td>
<td>77% / 76%</td>
<td>72% / 69%</td>
</tr>
<tr>
<td>Google</td>
<td>66% / 62%</td>
<td>78% / 76%</td>
<td>71% / 68%</td>
</tr>
<tr>
<td>Microsoft</td>
<td>64% / 65%</td>
<td>77% / 76%</td>
<td>71% / 68%</td>
</tr>
<tr>
<td>Alta Vista</td>
<td>63% / 61%</td>
<td>77% / 76%</td>
<td>70% / 68%</td>
</tr>
</tbody>
</table>

Finally, Table 6 shows the comparison of user satisfaction of our systemic prototype against other search engines. In the table, $S_T$ for Satisfaction of testers, represents the average of satisfaction responses from 10 ordinary users, while $S_E$ for Satisfaction of experts, represents that of satisfaction responses from 10 experts. Basically, each search engine receives 100 queries and returns the first 100 webpages for evaluation of satisfaction by both experts and non-experts. The table shows that our approach, the last row, enjoys the highest satisfaction in all classes. From the evaluation, we conclude that, unless the comparing search engines are specifically tailored to this specific domain, such as HotBot and Excite, our techniques, in general, retrieves more correct webpages in almost all classes.

5 Discussions and Future work

We have developed an Interface Agent to work as an assistant between the users and systems, which is different from system architecture and implementation over our previous work [12]. It is used to retrieve FAQs on the domain of PC. We integrated several interesting techniques including user modeling, domain ontology, and template-based linguistic processing to effectively tackle the problems associated with traditional FAQ retrieval systems. In short, our work features an ontology-supported, template-based user modeling technique for developing interface agents; a nature language query mode, along with an improved keyword-based query mode; and an assistance and guidance for human-machine
interaction. Our preliminary experimentation demonstrates that user intention and focus of up to eighty percent of the user queries can be correctly understood by the system, and accordingly provides the query solutions with higher user satisfaction. In the future, we are planning to employ the techniques of machine learning and data mining to automate the construction of the template base. As to the allover system evaluation, we are planning to employ the concept of usability evaluation on the domain of human factor engineering to evaluate the performance of the user interface.

Acknowledgements

The author would like to thank Yai-Hui Chang and Ying-Hao Chiu for their assistance in system implementation. This work was supported by the National Science Council, R.O.C., under Grant NSC-95-2221-E-129-019.

References:


