An ontological Proxy Agent with prediction, CBR, and RBR techniques for fast query processing

Sheng-Yuan Yang\textsuperscript{a,}\ast, Chun-Liang Hsu\textsuperscript{b}

\textsuperscript{a} Department of Computer and Communication Engineering, Saint John's University, 499, Sec. 4, Tamkung Road, Tamshui, Taipei, Taiwan, ROC
\textsuperscript{b} Department of Electrical Engineering, Saint John's University, 499, Sec. 4, Tamkung Road, Tamshui, Taipei, Taiwan, ROC

\textbf{A R T I C L E  I N F O}

\textbf{Keywords:}
Proxy Agent
Data mining
Query prediction
Case-based reasoning
Rule-based reasoning
Ontology

\textbf{A B S T R A C T}

This paper proposed a three-tier Proxy Agent working as a mediator between the users and the backend process of a FAQ system. The first tier makes use of an improved sequential patterns mining technique to propose effective query prediction and cache services. The second tier employs an ontology-supported case-based reasoning technique to propose adapted query solutions and tune itself by retaining and updating highly-satisfied cases identified by the user. Finally, the third tier utilizes an ontology-supported rule-based reasoning to generate possible solutions for the user. Our experiments show around 79.1\% of the user queries can be answered by Proxy Agent, leaving about 20.9\% of the queries for the backend Answerer Agent to take care, which can effectively alleviate the overloading problem usually associated with a backend server.

© 2009 Elsevier Ltd. All rights reserved.

1. Introduction

In this information-exploding era, the term ‘global village’ implies the shortening of distances between people as a result of the popularity and changing of the network technology with each passing day. We can virtually get any information (e.g., daily or new information about life) we are interested in by surfing the internet, no longer restricted in textbooks. The Internet is like a huge knowledge treasury waiting for exploration. A variety of query systems have thus appeared. The basic operational pattern of all the general query systems is to pass the user's query to a backend process, which is responsible for producing proper query result for the user. One major drawback of this approach is, when the number of queries increases, the backend process is overloaded, causing a dramatic degradation of system performance. The user then has to spend more time waiting for query responses. What is worse is that most of the long-awaited responses are usually dissatisfactory. Therefore, how to quickly get the information the user really wants over the limited bandwidth of the Internet and how to improve the retrieval performance of the query systems is becoming an important research topic. In addition, techniques that involve data gathering and integrating through database techniques are common in the literature. The following problems are usually associated with the techniques, however: (1) database relationships so constructed usually lack of physical meanings; (2) responses to user query are usually independent of the user level or the degree of user satisfaction; (3) automatic maintenance of the database through the user feedback is usually not available. Consequently, how to help users to find out user-oriented solutions; furthermore, to obtain, learn, and predict the best solution through user feedback, or how to support incremental maintenance of the solution database becomes an important research topic.

In order to provide high-quality FAQ answers on the Web to meet the user request, we have proposed an FAQ-master (Yang, 2007c; Yang, Hsu, Lee, & Deng, 2008) as an intelligent Web information aggregation system to provide intelligent information retrieval, filtering, and aggregation services. It contains four agents, namely, Interface Agent (Yang, 2007d, in press), Proxy Agent (Yang, 2007b; Yang et al., 2005a, 2005b), Answerer Agent (Chuang, 2003; Yang, 2008a; Yang, Chuang, & Ho, 2007), and Search Agent (Yang, 2006, 2007a, 2008b), supported by a content base, which in turn contains a user model base, a domain ontology base, a website model base, an ontological database, a solution library, and a rule base. In the context of FAQ service, this architecture can collect the most useful FAQ files, preprocess the files into consistent data, aggregate FAQ answers according to the user query and his/her user model, and provide quick FAQ answers through a proxy mechanism.

In this paper, we discussed an ontological Proxy Agent with solution integration and proxy techniques for Web information processing, which not only helps the user find out proper, integrated query results in accordance with his/her proficiency level or satisfaction degree, i.e., user-oriented solution, but supports
proxy access of query solutions through an intelligent solution finding process, as shown in Fig. 1. The architecture involves two agents, namely, Answerer Agent and Proxy Agent, and shows how it interacts with Interface Agent and Search Agent. Answerer Agent uses the wrapper approach to do web information preparation, including parsing, cleaning, and transforming Q–A pairs, obtained from heterogeneous websites, into an ontology-directed canonical format, then store them in ontological database (OD) via ontological database manager (ODM). In order to speed query processing, we introduced an ontological Proxy Agent with three proxy-relevant mechanisms, namely, CBR (case-based reasoning), RBR (rule-based reasoning), and solution prediction. Solution Finder is designed to serve as the central control in finding solutions. To ensure that all the knowledge used in CBR, RBR, and solution prediction can be automatically generated, we have introduced a rule miner and a Solution Predictor into the system. They will be described in the following sections.

Our experiments show around 79.1% of the user queries can be answered by Proxy Agent, leaving about 20.9% of the queries for Answerer Agent to take care, which can effectively alleviate the overloading problem usually associated with a backend server, a noticeable improvement in overall query performance. The personal computer (PC) domain is chosen as the target application of our Proxy Agent and will be used for explanation in the remaining sections. The rest of the paper is organized as followings. Section 2 develops the domain ontology as the fundamental semantics. Section 3 describes the design of the three-tier Proxy Agent and how it works. Section 4 illustrates how the first-tier performs query cache and query prediction services. Sections 5 and 6 illustrate how CBR works as the second-tier solution provider and how RBR works as the three-tier solution provider, respectively. Section 7 describes the implementation of our Proxy Agent and reports how much better it performs. Section 8 discusses related works, while Section 9 concludes the work.

2. Domain ontology as fundamental semantics

Ontology is a method of conceptualization on a specific domain (Noy & Hafner, 1997). It plays various roles in developing intelligent systems, including knowledge sharing, knowledge reuse, semantic analysis of languages, etc. Development of ontology for a specific domain is not yet an engineering process, but it is clear that a domain ontology must include descriptions of explicit concepts and their relationships about a specific domain (Asnicar & Tasso, 1997). We have outlined a principle construction procedure in Yang (2007d); following the procedure, we have developed an ontology for the PC domain. Fig. 2 shows part of the ontology taxonomy. The taxonomy represents relevant PC concepts as classes and their relationships as a variety of links. The figure illustrates an isa link, which allows inheritance of features from parent classes to child classes. We have carefully selected, from each concept, the properties that are most related to our application and defined them as the detailed ontology for the corresponding class. Fig. 3 exemplifies the detailed ontology of the concept of 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 lower level represent various CPU instances, which capture real world data. The complete PC ontology can be referenced from the Protégé Ontology Library at Stanford Website (http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library) or our website (http://pontology.et.ntust.edu.tw).

We have also developed a problem ontology to help process user queries. Fig. 4 illustrates part of the problem ontology, which contains query type and operation type. These two concepts constitute the basic semantics of a user query and are therefore used as indices to structure the cases in ODAC (ontological database access cases) and OD, which in turn can provide fast case retrieval. Finally, we have used Protégé’s APIs (application program interface) to develop a set of ontology services, which work as the
primitive functions to support information retrieval from the ontology. As illustrated in Fig. 5, 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.
3. Three-tier proxy design

The Proxy Agent performs solution finding by a three-tier architecture. Fig. 1 illustrates the architecture of Proxy Agent and how it interacts with Interface Agent and the backend process, i.e., Answerer Agent. First, Interface Agent collects queries from the user according to his/her user model stored in user model base, which models his/her important behavior as well as his/her query history. Answerer Agent manages OD, which stores preprocessed Q–A pairs collected from the Web, and retrieves proper Q–A pairs to respond to user queries. Inside Proxy Agent, Ontology Base contains the PC ontology as shown in Fig. 2. ODAC is the case base for CBR. It stores past user query cases, which come from the Q–A pairs produced by Answerer Agent. Solution Predictor is responsible for tracking user query history, storing most occurring queries in Cache Pool, and predicting next possible queries for storage in Prediction Pool. Cache Pool and Prediction Pool are combined into a Query Pool, which provides query cache and query prediction to reduce query response time. CBR employs the case-based reasoning technique to reason about solutions for a given query from ODAC. If there is an existing case in ODAC that is exactly the same as the user query, CBR directly outputs it as the solution to the user. If only similar cases to the user query exist in ODAC, CBR modifies the solutions for the user through a case adaptation process. CBR is also responsible for the maintenance of cases in ODAC by evaluating and determining whether a new query-answer pair deserves storage in ODAC to serve as a new case according to the user's feedback to the solution. The job functions of RBR can be divided into two parts: off-line rule mining and on-line reasoning. Firstly, the former employed the mining, validating, and generalizing techniques for mining out the rules to suitably do rule-based reasoning and then store those rules into Rule Base for doing on-line reasoning. When Solution Finder turns to RBR for asking for solutions, the latter can employ the reasoning technique to pick out the best suitable rules to answer to the user query from Rule Base, under the support of PCDIY Ontology to carry rule-based reasoning out, and accordingly provide the reasoned solutions to Solution Finder.

Solution Finder is the central control in finding solutions. After receiving a user query from Interface Agent, it tries to produce an answer following an intelligent solution finding process, which includes predicted solution retrieval, CBR, RBR, and solution integration. Firstly, Solution Finder checks whether any predicted queries exist in User Model Base. If yes, it directly produces the answer retrieved from the question–answer pair. If none, Solution Finder starts CBR to find out query solution. If the given query is already existent in ODAC, Solution Finder directly outputs its answer part. If none, Solution Finder performs case adaptation to solve the query. If CBR provides no solutions, Solution Finder exerts RBR, i.e., using the rules in Rule Base to find out query solution. Note that, in addition to the exact query answer, Solution Finder also gathers the solutions that are super set of the user question for the user. The last mechanism of finding solutions is to trigger Solution Integrator to integrate solutions from OD. If this integrated solution is credited with a high degree of satisfaction by the user, it will be stored in the ODAC serving as a new case.

In short, Proxy Agent employs a three-tier proxy mechanism, including Solution Predictor, CBR, and RBR modules, to speed up query processing, at the same time reducing the loading of Answerer Agent. If Solution Finder gets a solution from one of three tiers, it immediately passes the solution to Interface Agent, which then produces a proper query result for the user according to his proficiency level and idiosyncratics (Yang, 2007d; Yang et al., 2007). If none of three tiers can provide solutions, Solution Finder asks Answerer Agent to derive a solution (Yang et al., 2007c, 2008).

![Fig. 6. First-tier proxy.](image-url)
4. First-tier proxy: query prediction and query cache

The basic idea behind the first-tier proxy is the observation of common sequential patterns in the query behavior of the users. Solution Predictor exploits this observation and uses the sequential pattern mining technique to analyze implicit correlations from the user query behavior. These correlations then serve as potential rules for prediction of future user query.

Fig. 6 shows the architecture of the Predictor working as the first-tier proxy. Query pattern miner looks for frequent sequential query patterns using the Full-Scan-with-PHP algorithm, from the query histories of the users of the same group as recorded in User Models Base (Yang et al., 2005a). Note that we pre-partitioned the users into five user groups according to their proficiency on the domain (Yang, 2007d). Query Miner then turns the frequent sequential query patterns to Case Retriever, which is responsible for retrieving corresponding solutions from ODAC and constructing ‘frequent queries’ for storage in Cache Pool. The frequent sequential query patterns are also sent to Prediction Module to construct a prediction model for each user group. Pattern Matching Monitor is responsible for monitoring recent query records and using the prediction model to produce next possible queries for storage in Prediction Pool.

In short, the predictor produces ‘frequent queries’ for Cache Pool and ‘predicted queries’ for Prediction Pool off-line. Given a new query online, Solution Finder passes the query to the predictor, which employs both query prediction and query cache mechanisms to produce possible solutions for the query.

4.1. Full-Scan-with-PHP algorithm

The Full-Scan algorithm was developed, based on the DHP algorithm, to perform sequential pattern mining (Chen, Park, & Yu, 1998). The PHP algorithm introduced the perfect hash mechanism into the DHP algorithm and proved that it performs better than the DHP algorithm (Özel & Güvenir, 2001). Fig. 7 integrates PHP with Full-Scan to form our sequential pattern mining algorithm, which returns \( L \) (step 10) as a set of frequent sequential patterns. The pruning method (step 7) is the same as PHP. The major feature of the algorithm is that it can obtain \( L_2 \) by directly scanning the perfect hash table \( PH_k \) (step 8) without spending extra time scanning the database to calculate supports for the frequent sequence patterns. Fig. 8 illustrates how the algorithm works on example database \( D_1 \) with minimum support \( min\_supp \) set to 2.

4.2. Construction of Query Pool

As noted before, Cache Pool contains frequent queries while Prediction Pool contains predicted queries. Both are constructed based on the frequent sequential query patterns discovered by the Full-Scan-with-PHP algorithm. Note that we treat each query case stored in ODAC as an item during the operation of the algorithm. The algorithm first constructs \( L_1 \) as a set of 1-item frequent sequential patterns. Strictly speaking, this set has nothing to do with query sequences yet, but it represents the pool of all frequent queries. Thus, each element in the set can be treated as a frequently happening query, which can be cached somewhere for ready use in the future. We hence put each element of the set to Cache Retriever, which then retrieves its corresponding answer part from the ODAC to form a complete frequent query for storage in Cache Pool. Fig. 9 illustrates some examples of frequent queries stored in Cache Pool.

The final output of the mining algorithm returns \( L \), a set of frequent sequential patterns, and submits it to Prediction Module for producing sequential rules. The basic idea follows (Zhang, 2001) by defining the sequential rule confidence as follows.

**Definition 1.** Given two frequent sequential patterns \( S_1 = \{t_1, t_2, \ldots, t_k\} \) and \( S_2 = \{t_{k+1}, \ldots, t_{k+l}\} \), where \( k \geq 2 \) and \( S_2 \) is a sub-sequential pattern of \( S_1 \), we produce a sequential rule “\( t_1, t_2, \ldots, t_{k-1} \rightarrow t_k \)” with the following rule confidence:

![Fig. 7. Full-Scan-with-PHP algorithm.](image)

![Fig. 8. Illustration on sequential pattern mining of Full-Scan-with-PHP algorithm.](image)

![Fig. 9. Examples of frequent queries in Cache Pool.](image)
Confidence \(t_1, t_2, \ldots, t_k \rightarrow t_k = \text{Support}(S_1)/\text{Support}(S_2)\)

We define that a sequential rule is legal if its confidence satisfies some minimal confidence. Table 1 shows part of the legal sequential rules generated from the database \(D_1\) of Fig. 9 with minimal confidence 0.6. Fig. 10 illustrates some examples of prediction rules about the PC domain stored in the prediction model.

With the prediction model constructed, Pattern Matching Monitor can do query prediction and produce predicted queries into Prediction Pool. Basically, it monitors and traces the query record of the user in a session, and employs the longest-path-first method (Zhang, 2001) to check whether any matched sequential rules exist in the prediction model. A query record is treated as sequence of queries. If the antecedent of a sequential rule matches the pattern of the traced query record, Pattern Matching Monitor produces the consequence of the rule as the next possible query and retrieves its corresponding solution part from ODAC to form a predicted query.

### 4.3. First-tier solution service

Given a user query, Solution Finder first checks whether there are any predicted queries in Prediction Pool. If yes, it directly generates a solution by retrieving the answer part of the predicted query for the user. If none exists, it turns to check whether any cached queries exist in Cache Pool. If one exists, it generates a solution from its corresponding answer part. If none exists, it turns to the second tier for generating solutions.

### 5. Second-tier proxy: case-based reasoning

The second-tier proxy employs CBR to provide a broader range of search for solutions than the first-tier. Fig. 11 illustrates the architecture of the second-tier proxy mechanism. Again, the ODAC is the case library, which contains query cases produced by the backend Answerer Agent. Case Retriever is responsible for retrieving a case from ODAC, which is the same as or similar to the user query. Case Reuser then uses the case to check for any discrepancy against the user query. If the case is completely the same as the user query, Reuser directly outputs it to the user. If the case is only similar to the user query, Reuser passes it to Case Reviser for case adaptation. Case Reviser employs the PC ontology along with Adaptation Rule Base to adapt the retrieved case for the user. Adaptation Rule Base contains adaptation rules, constructed by the domain expert. Case Retainer is responsible for the maintenance of ODAC, processing case addition, deletion, and aging. In short, the second-tier proxy involves four operations, namely, case retrieval, case reuse, case adaptation, and case maintenance, to be detailed below.

#### 5.1. Case retrieval

Fig. 12 shows how Case Retriever performs case retrieval. At step 1, if Case Retriever retrieves a case, which contains the same question features with, or more question features than, the user query, it directly outputs it as the solution for the user query. Otherwise, at step 2, it checks for similar cases using the VRelation-
ship in the ontology. Table 2 illustrates some examples of VRelationship, to be explained below.

(1) Mutually exclusive VRelationship: This constraint enforces that an attribute can only take on one and only one value from a legal set of values. For example, row 1 of the table illustrates that a motherboard cannot belong to two different producers at the same time. A retrieved case containing attributes having this VRelationship with the user query cannot become a similar case.

(2) Downward-compatible VRelationship: This constraint defines some compatibility between two related attributes. For example, row 2 of the table says that a motherboard that contains four USB (Universal Serial Bus) ports can be regarded as one with two USB ports. Thus, a retrieved case containing an attribute having this VRelationship with the user query can become a similar case.

(3) Conditionally downward-compatible VRelationship: This is very similar to the case above, except that the compatibility is subject to some conditions. For example, row 3 of the table shows that a 1.8 GHz Pentium 4 CPU can be regarded as one with 1.4 GHz, but cannot be regarded as one with 866 MHz, which is a Pentium III CPU format. In this case, the retrieved case is treated as a similar case with extra conditions.

5.2. Case reuse

When Case Retriever recognizes similar cases, Case Reuser steps in to check whether a similar case can be directly reused or needs some adaptation before being reused. Specifically, if the VRelationship between the user query and the similar case is:

(1) Downward-compatible, then the case is directly treated as a legal solution.

(2) Conditionally downward-compatible, then the case is treated as a reference case and sent to Case Reviser for case adaptation according to the inference rules.

5.3. Case adaptation

A case adaptation process involves three operations. First, Case Reviser finds a reason for adaptation according to the PC ontology. Then, it selects proper adaptation rules according to the reason to perform case adaptation. Finally, it produces an adapted solution for the user.

Table 3 illustrates a scenario of case adaptation, which contains a user query and two reference cases retrieved from ODAC. Note that the “conditionally downward-compatible VRelationship” relates the two reference cases to the user query. The first adaptation step is to use the PC ontology to identify a feature that is constrained by the VRelationship between the user query and the reference cases. This constrained-feature explains the semantics of the specified relationship, and thus may suggest proper adaptation. The example shows that the feature CPU_Clock_Rate is constrained by the “conditionally downward-compatible” relationship. Thus, we should check whether there are adaptation rules referring to feature CPU_Clock_Rate in Adaptation Rule Base. Suppose we have an adaptation rule as shown below:

- IF concept ∈ “Motherboard” and constrained-feature = “CPU_Clock_Rate”
- THEN adaptation attribute = “Support_CPU_Clock_Range”

The rule says if the concept is about an instance of “Motherboard” and the constrained-feature is “CPU_Clock_Rate”, then we can focus on attribute “Support_CPU_Clock_Range” for adaptation. Case Reviser then checks whether the adaptation feature exists in the reference cases. If yes, it goes one step further to check whether the feature values meet the requirement of the user query. For example, Table 3 shows three motherboard instances, “P4S333, P4S433, and P4S533”, which are derived from the two reference cases. It also shows their respective values with respect to the adaptation feature “Support_CPU_Clock_Range”. We find P4S333 supports up to 1.5 GHz, which cannot meet the user requirement - 1.8 GHz. Therefore P4S333 has to be removed from the set of solutions, leaving the final solutions to be P4S433 and P4S533.

Table 2 Examples of VRelationship.

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Feature</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutually exclusive</td>
<td>MB_Provider</td>
<td>华硕 (ASUS)</td>
<td>A motherboard cannot belong to two different producers at the same time</td>
</tr>
<tr>
<td>Downward-compatible</td>
<td>MB_PCI_Num</td>
<td>PCI-4</td>
<td>A motherboard which contains four USB ports can be regarded as one with two USB ports</td>
</tr>
<tr>
<td>Conditionally downward-compat</td>
<td>CPU_Clock_Rate</td>
<td>Pentium 4 1.4G</td>
<td>A 1.8 GHz Pentium 4 CPU can be regarded as one with 1.4 GHz, but cannot be regarded as one with 866 MHz, a Pentium III CPU format</td>
</tr>
</tbody>
</table>

Fig. 12. Case retrieval.
5.4. Case maintenance

Fig. 13 shows some examples of cases stored in ODAC. Note that each case is associated with a survival value (the last column), which represents how active the case is in the system, and serves as our case maintenance basis. The increment or decrement of the survival value of a case depends upon its satisfaction degree. Table 4 illustrates the system-defined five degrees of satisfaction for the user to provide his feedback on the proposed solutions.

The maintenance of survival values depends on whether the case is a solution, a reference case, an adapted case, or a new solution case from the backend Answerer Agent, to be detailed below.

5.4.1. Solution cases

For a solution case, we can directly reflect the user feedback into the survival value of the case. Eq. (1) defines how we refresh the survival value $SR(c)$ of a solution case $c$.

$$SR^{(new)}(c) = SR^{(old)}(c) + \Delta SR(c)$$

where

$$\Delta SR(c) = (Sat(c) - 0.45) \times \alpha$$

Eq. (2) is used instead to assign a survival value to an adapted case, where $Sat(c)$ stands for the representative value of the satisfaction degree of case $c$ as defined in Table 4, and $\alpha$ represents the learning rate, set to 0.1 in our system for slowly adjusting $SR(c)$.

5.4.2. Reference cases

In a reference case, we have to take into account the similarity of the reference case against the adapted solution. Eq. (3) first defines the solution similarity.

$$Sim(a, c) = |S_a \cap S_c| / |S_a \cup S_c|$$

where ‘$a$’ is an adapted case, ‘$c$’ is a reference case, $S_a$ is the answer part of case ‘$a$’, and $S_c$ is the answer part of ‘$c$’. $Sim(a, c)$ measures the extent to which the reference case contributes to the case adaptation. Table 5 illustrates the solution similarities between the reference cases and the adapted case of Table 3. Now we can use Eq. (1) to refresh the survival value of a reference case, after $\Delta SR(c)$ is redefined by Eq. (4).

$$\Delta SR(c) = (Sat(c) - 0.45) \times Sim(a, c) \times \alpha$$

5.4.3. Adapted cases

We retain an adapted case in ODAC only when its satisfaction degree is over a pre-defined satisfaction threshold and its similarity is less than a pre-defined similarity threshold. Both thresholds are set to 0.5 in our system in order to balance user satisfaction and case similarity in retaining an adapted case. They can be changed according to how the system prefers user feedback or case similarity. We then use Eq. (5) to assign the retained adapted case an initial survival value $SR(a)$.

$$SR(a) = \frac{\sum_{i=1}^{n} Sim(a, c_i) \times SR(c_i)}{n}$$

where $SR(c_i)$ is the survival value of the reference case ‘$c_i$’, $n$ is the number of reference cases contributing to the adaptation of ‘$a$’, $c_i$ is ith reference case, and $Sim(a, c_i)$ is the solution similarity between ‘$a$’ and ‘$c_i$’.

5.4.4. New solution cases from the backend process

Since a new solution case from the backend process has no ‘old’ survival value, we cannot use Eq. (1) to refresh its survival value. Eq. (6) is used instead to assign a survival value to $C_{new}$, the new solution case, where $SR_{ave}$ is the averaged survival value of all the cases in ODAC, as defined in Eq. (7).

$$SR(C_{new}) = SR_{ave} + Sat(C_{new})$$

$$SR_{ave} = \frac{\sum_{i=1}^{n} SR(C_i)}{n}$$

6. Third-tier proxy: rule-based reasoning

The third-tier proxy employs RBR to provide a broadest range of search for solutions than the first two tiers. Fig. 14 illustrates the solving architecture of RBR. Inside the architecture, Rule Miner is responsible for mining rules from ODAC. Rule Validator is responsible for testing and verifying those rules mined by Rule Miner and filtering out the valid rules. Rule Generalizer then generalizes those valid rules filtered by Rule Validator and accordingly stores the generalized rules into Rule Base. Finally, Solution Reasoner is responsible for taking out the proper rules to reply to the user query from Rule Base, putting to the proof of the validation of the solution part of the rule supported by PCDIY Ontology, and finally producing the valid solutions to the user. In short, the job...
functions of RBR can be divided into two parts: off-line rule mining and on-line reasoning. The former employed Rule Miner, Rule Validator, and Rule Generalizer for mining out the rules to suitably do rule-based reasoning and then store those rules into Rule Base for doing on-line reasoning. When Solution Finder turns to RBR for asking for solutions, the latter can employ Solution Reasoner to pick out the best suitable rules to answer to the user query from Rule Base, under the support of PCDIY Ontology to carry rule-based reasoning out, and accordingly provide the reasoned solutions to Solution Finder.

6.1. PD with PH algorithm

The mining algorithm combined the Perfect Hashing (PH) technology with Pattern Decomposition (PD) algorithm Zou, Chu, David, & Chiu, 2002, shown in Fig. 15, to get frequent itemsets for off-line rule mining. This algorithm is different from PD algorithm in the Perfect Hash table, which not only can play the role of looking for all \( k \)-item sets in the pattern (step 6), but also can calculate the degree of support in the item sets (step 8–9). The rest of the algorithm is the same as PD algorithm. The main purpose of step 6 of this algorithm is to find out all \( k \)-item sets in the pattern, especially, in the composite pattern. The steps 8–9 then employed another perfect hash table, i.e., \( PH_k \), to increase the occurrence value of \( k \)-item sets in the pattern. Finally, the step 12 directly scanned this perfect hash table for getting \( L_k \) and \( \neg L_k \).

**Fig. 14.** Architecture of RBR.

**Fig. 15.** PD with PH algorithm.

**Fig. 16.** Flowchart of mining association rules.
6.2. Off-line rule mining

In RBR solving mechanism, the role of Rule Miner is important to the processing of off-line rule mining, in which Rule Miner is responsible for mining association rules from the cases in ODAC for RBR according to the frequent itemsets mined by PD with PH algorithm. A mixed version of Apriori algorithm (Agrawal & Srikant, 1995) and Eclat algorithm (Zaki, Parthasarathy, Ogihara, & Li, 1997) is properly modified to perform the rule-mining task, as shown in Fig. 16. Rule Miner is invoked whenever the number of new cases in ODAC reaches a threshold value. In addition, it is worthwhile to be mentioned that the rule mining process is not for whole ODAC but for different question types such as ‘What’, ‘How’, ‘Why’, ‘If’, and etc. Those mined rules comparatively possess the semantic meaning and convenient for management.

6.3. On-line rule reasoning

If no solution obtains from Solution Predictor and CBR, RBR is triggered by Solution Finder, which makes rule-based reasoning to generate possible solutions supported by PCDIY ontology and accordingly provides those solutions to the user via Solution Finder. Fig. 17 shows the reason steps of RBR.

7. System evaluation

Our Proxy Agent was developed using Borland JBuilder 5.0 on Microsoft Windows XP, as shown in Fig. 18. The database management system is Microsoft SQL Server 2000, and the ontology development tool is Protégé2000. We collected in total 517 FAQs from the FAQ website of one famous motherboard factory in Taiwan and then transformed the query-answer pairs of each FAQ for storage in ODAC. Our first experiment was to learn how the Predictor works well. We used in total 200 user query scenarios of the same user level as the training data set. We set the minimal support to 3% and minimal confidence to 60%. The Full-Scan-with-PHP algorithm was used to construct 36 frequent queries for storage in Cache Pool and 43 rules in Prediction Model. Prediction Pool only kept the most recent three predicted queries, in order to avoid over-fitted prediction, which produces a solution similarity to rote learning of lots of prefetched queries. We then randomly selected 100 query scenarios from the training data set as the testing data to test the performance of the Predictor. Table 6 illustrates the five-time experiment results. It shows, on average, that Query Prediction is invoked to predict next query for 60.2 queries, among which it can correctly predict next query for 38 queries, a 65.8% average success rate. It also shows, on average, out of 310 queries,
38 queries can be taken care of by the query prediction mechanism (around 12.2%), while 59.2 queries can be taken care of by the query cache mechanism (around 19.1%). In summary, around 31.3% of the user queries can be answered by the first-tier proxy.

Our second experiment is to learn how well the overall three-tier Proxy Agent works. We used the same Cache Pool and Prediction Model as the first experiment for the first-tier proxy. We then randomly selected about 500 FAQs from OD, extracted proper query keywords from their question parts, and randomly combined the keywords into a set of 345 queries cases for ODAC for experiment with the second-tier case-based reasoning. We conducted the experiment for five times. After each experiment, some 15 new cases are retained into ODAC by Cache Retainer and the system trigged Rule Miner to do rule re-mining in ODAC for the next experiment with the help of user feedback from the experiment conductor. Table 7 illustrates the five-time experiment results. It shows, on average, 31.3% (12.2 + 19.1%) of the user queries can be answered by the user-oriented query prediction and cache technique, while 47.8% (39.8 + 8%) of the user queries can be taken by the ontology-supported CBR and RBR. In short, the experiment shows around 79.1% of the user queries can be answered by Proxy Agent, leaving about 20.9% of the queries for Answerer Agent to take care, which can effectively alleviate the overloading problem usually associated with a backend server. This is a very impressive improvement in query response.

8. Related works

The emerging knowledge discovery techniques, including knowledge discovery in databases (KDD) (Frawley, Picatetsky-Shapiro, & Matheus, 1992; Mendonça & Sunderhaft, 1999; Shortland & Scarle, 1995), Association Rules (Agrawal & Srikant, 1995), Classification (Ruggieri, 2002), Clustering (Tsai, Wu, & Tsai, 2002), Sequential Patterns (Agrawal & Srikant, 1995), etc. has inspired a lot of work on user action prediction. For example, Web Watcher (Joachims, Freitag, & Mitchell, 1997) anticipates the next selected hyperlink by using a model built through reinforcement learning. Transparent Search Engine (Bota, Corno, Farinetti, & Squillero, 2002) evaluates the most suitable documents in a repository using a user model updated in real time. A different approach to Web pages prediction is the ‘Path-based’ systems. For example, the work of Yang, Zhang, & Li (2001) presents an application of Web log mining to obtain web-document access patterns and uses these patterns to extend the well-known GDSF (Greedy-Dual-Size-Frequency) caching policies and prefetching policies. PPS (Proxy-based Prediction Service) (Lou & Lu, 2002) applies a new prediction scheme which employs a two-layer navigation model to capture both inter-site and intra-site access patterns, incorporated with a bottom-up prediction mechanism which exploits reference locality in proxy logs. The work of Bonino, Corno and Squillero (Bonino, Corno, & Squillero, 2003) proposes a method to exploit user navigational path behavior to predict, in real-time, future requests using the adoption of a predictive user model based on Finite State Machines (FSMs) together with an evolutionary algorithm that evolves a population of FSMs for achieving a good prediction rate. In the paper, we adopt the technique of sequential patterns mining to discover the user query behavior from his query history and accordingly to offer efficient query prediction and query cache services, just like (Oikonomopoulou, Rigou, Sirmakesis, & Tsakalidis, 2004) in which differently mined from server log files and either (Gopalratnam & Cook, 2007) using different sequential prediction algorithm, say Active LeZi.

Intelligent agents are programs that act on behalf of their human users to perform laborious information-gathering task. The last ten years have seen a surging interest in agent-oriented technology, spanning applications as diverse as information retrieval, intelligent document filter, etc. We noticed that case-based reasoning (CBR) has played an important role in agent development. For example, Liu and Leung (2001) present a Web-based case-based reasoning model to assist investors to determine stock trend. The model conforms to a web-based agent framework, forming part of an advisory system for financial forecast. Different cases are collected based on the theory of wave features and their combination. The agent framework supports processes including knowledge generation, wave units mining and wave pattern recognition, and case revise and learning. Aktas, Pierce, Fox, and Leake (2004) develops a recommender system which uses conversation case-based reasoning with semantic web markup languages providing a standard form of case representation to aid in metadata discovery. Lorenzi, dos Santos, and Bazzan (2005) presents the use of swarm intelligence task allocation among cooperative agents applied to a case-based recommender system to help in the process of planning a trip. In this paper, the CBR technique is used as a problem solving mechanism in providing adapted past queries. It is also used as a learning mechanism to retain high-satisfied queries to improve the problem solving performance. We further present a hybrid approach which combine CBR with RBR for providing solutions, just as Shi and Barnden (2005) in which differently diagnosing multiple faults.

9. Conclusions

We have proposed a three-tier Proxy Agent, composed of a first-tier Predictor, a second-tier CBR, and a third-tier RBR to work as a mediator between the users and the backend Answerer Agent of an FAQ service system. The Predictor makes use of an improved sequential patterns mining algorithm, Full-Scan-with-PHP, to discover user query behavior as a basis to construct Cache Pool and Prediction Model, which can support efficient query prediction and query cache. The CBR employs an improved case-based reasoning technique to reason about adapted solutions for a given user query, with the help of domain ontology, from a case library, which is then fine-tuned according to the user feedback.
information. The RBR makes use of a frequent itemsets mining algorithm, a PD with PH algorithm, and utilizes an ontology rule-based reasoning technique to generate possible solutions supported by PCDIY ontology and accordingly provides those solutions to the user.

The Agent is interesting in the following respects. Firstly, it performs fast user-oriented mining and prediction. The Predictor uses the user query history stored in the User Model Base to discover frequent queries and predicted queries. The sequential pattern mining algorithm is made more efficient by the techniques of perfect hashing and database decomposition. Secondly, it performs an ontology-directed case-based reasoning. The semantics of PC ontology, in particular the VRelationships relationships, are used in determining similar cases, performing case adaptation, and case retaining. Thirdly, it self-improves by tuning the survival value of each case in the case library. This tuning process involves user satisfaction and case similarity as two major factors, which are both user-oriented and ontology-supported. Finally, it fulfills an ontology-supported rule-based reasoning. To combine the PH technology with PD algorithm is used in mining the frequent itemsets, and then a modified version of Apriori algorithm and Eclat algorithm to perform the rule-mining task, and accordingly employs the reasoning technique to pick out the best suitable rules to answer possible solutions to the user query under the support of PCDIY Ontology. Our experiments show that the Agent can share up to 79.1% of the query loading from the backend process, leaving about 20.5% of the queries for the backend Answerer Agent to take care, which stands for a big improvement of the overall query performance.

For easy demonstration of the techniques in our system, the current implementation runs on a specific ‘PC’ domain. However, we believe even if the domain is scaled up, our techniques are still applicable. The idea is this: we are not directly scaling up our ontology; instead we can create a complex system by integrating a set of simple systems through a multi-agent architecture, which is supported by a set of simple domain ontologies. By exploiting the capability of Protégé2000, which supports creation, extension and cooperation of a set of domain ontologies, we really need not make much change to our system in order to transform it into a complex system. We need only to recollect and reconstruct ODAC and prediction models for supporting the first-tier proxy, rebuild the adaptation rule base for supporting the second-tier proxy, re-mine the frequent itemsets to perform rule-mining task for supporting the third-tier proxy, and devise a mechanism to make sure a set of ontology-supported systems can cooperate effectively, which is under our investigation.

Acknowledgements

The authors would like to thank Y.H. Chiu, Y.H. Chang, and F.C. Chung 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


Chuang, F. C. (2003). Ontology-supported and ranking technique-enhanced FAQ system. Master Thesis, Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan.
Yang, S. Y. (2007a). An ontological website models-supported search agent for web services. In Accepted for publication in expert systems with applications (On-line).


OntoPortal: An ontology-supported portal architecture with linguistically enhanced and focused crawler technologies

Sheng-Yuan Yang *

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

ARTICLE INFO

Keywords:
Ontology
Web portals
Automatic annotation
Focused crawlers

ABSTRACT

This paper proposed the techniques of ontology and linguistics to develop a fully-automatic annotation technique, coupling with an automatic ontology construction method, could play a key role in the development of Semantic Portals. An ontology-supported portal architecture: OntoPortal was proposed according to this technique, in which three internal components Portal Interface, Semantic Portal, and OntoCrawler was integrated to rapidly and precisely collect information on Internet and capture true user’s intention and accordingly provide high-quality query answers to meet the user requests. This paper also demonstrated the OntoPortal prototype which defined how a semantic portal is interacting with the user by providing five different types of interaction patterns such as including keyword search, synonym search, POS (Part-of-Speech)-constrained keyword search, natural language query, and semantic index search. The preliminary experiment outcomes proved the technology proposed in this paper to be able to really up-rise the precision and recall rates of webpage searching and accordingly showed that it can indeed retrieve better semantic-directed information to meet user requests.

© 2009 Elsevier Ltd. All rights reserved.

1. Introduction

The Web has been drastically changing the availability of electronically available information. An information portal, which can help people effectively use this voluminous repository, is becoming ever important. Tim Berners–Lee visions “Semantic Web” as the next generation of web by allowing the web pages on the Semantic Web to be machine-readable (Lee, Hendler, & Lassila, 2001). Techniques to realize this vision include the techniques centered on domain ontology (Lee et al., 2001), which help domain knowledge conceptualization. Webpage annotation is another important technique, which adds relevant domain knowledge and meta-information into a web page to enhance its semantics (Albanese et al., 2005). Logic inference is yet another important technique, which allows the development of various web services. These techniques can effectively relieve the bottlenecks of the current Web (Fensel, Hendler, Lieberman, & Wahrler, 2000; Stollberg & Thomas, 2005). Automatic webpage annotation can even reduce the high cost of pre-processing of web pages into a semantic web by humans (Benjamins, Contreras, Corcho, & Gomez-Perez, 2002; Kiyavitskaya, Zeni, Cordy, Mich, & Mylopoulos, 2005).

We also notice that the concept of crawler is mostly used in the Web systems that work on information gathering or integrating to improve their gathering processes or the search results from ubiquitous information environments. For instance, Dominos (Hafri & Djeraba, 2004) can crawl several thousands of pages every second, in which includes a high-performance fault manager, especially, the system possesses platform independent and is able to adapt transparently to a wide range of configurations without incurring additional hardware expenditure. Ganesh, Jayaraj, Kalyan, and Aghila (2004) developed an ontology-support Web crawler with an 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. Ubi–Crawler (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.

We conjectured that these techniques can equally help us overcome the problems associated with general portals and employed the techniques of ontology and linguistics to develop a fully-automatic annotation technique (Yang, 2007), which, coupling with an automatic ontology construction method, could play a key role in the development of Semantic Portals (Chang, 2003). An ontology-supported portal architecture: OntoPortal was proposed according to this technique, which integrated three internal components Portal Interface, Semantic Portal, and OntoCrawler (Yang, Chen, & Wu, 2008) to rapidly and precisely collect information on Internet and capture true user’s intention and accordingly provide high-quality query answers to meet the user requests. This paper also demonstrated the OntoPortal prototype which defined how a semantic portal is interacting with the user by providing five different types...
of interaction patterns such as including keyword search, synonym search, POS (Part-of-Speech)-constrained keyword search, natural language query, and semantic index search. The preliminary experiment outcomes proved the technology proposed in this paper to be able to really up-rise the precision and recall rates of webpage searching and accordingly showed that it can indeed retrieve better semantic-directed information to meet user requests.

The rest of the paper is organized as following: Section 2 develops the domain ontology. Section 3 explains OntoPortal architecture. Section 4 reports the system demonstrations and evaluations. Section 5 compares the work with related works, while Section 6 concludes the work. The Scholar: Einstein domain is chosen as the target application of OntoPortal and will be used for explanation in the remaining sections.

2. Domain ontology as the down-to-the-earth semantics

Ontology provides complete semantic models which means in specified domain all related entities, attributes and base knowledge among entities owning sharing and reusing characteristics used for solving the problems of common sharing and communication (Yang & Ho, 1999). Protégé http://protege.stanford.edu/ (Noy et al., 2001) was a free, open-source ontology editor and knowledge-based framework developed by Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine. Protégé was not only one of the most important platforms to construct ontology but also the most frequently adapted one (Grosso et al., 1999). Protégé 3.3.1 was adapted in this paper, which supported two main ways of modeling ontologies via the Protégé-Frames and Protégé-OWL (adapted by us, shown in Fig. 1) editors. Protégé ontologies can be exported into a variety of formats including RDF(s), OWL, and XML Shema and lead knowledge workers to constructing knowledge management system based on ontology; furthermore, users could transfer to different formats of ontology such as RDF(S), OWL XML or directly inherit into database just like MySQL and MS SQL Server (adapted by us and detailed later), which have better supported function than other tools (Chien, 2006).

Nowadays the research on ontology can be branched into two fields: one is to configure huge ontology in a specified field and through it to assistant the knowledge analysis in this field; the other is to study how to construct and express precisely with ontology (Chien, 2006). In this paper, we adapted the former in which took advantage of built ontology database of some scholars to support OntoCrawler for querying webpage of related scholars. First of all, we conducted statistics and survey of homepage of related scholars to fetch out the related concepts and their synonym appearing in the homepage.

The second stage of ontology constructing of scholars is to transfer the ontology built with Protégé into MS SQL database. The procedures are as following:

1. With Protégé to define the scholars’ ontology analyzed from the first stage so as to share the ontology with other interesting researchers.
2. Exporting an XML file constructed with Protégé knowledge base and then importing into MS Excel for correcting. That was the strong evidence of knowledge reusing and fast embedding within Protégé.
3. Finally importing MS Excel into MS SQL Sever to finish the ontology construction of this system.

Fig. 1 indicated the structure of domain ontology of scholars in Protégé, taking the middle frame of the screen for instance, the related concept “Education” was linking behind with “M.S.”, “Ph.D.”, and “B.S.”. In application, we defined those as related concepts and that means “Education” is nothing but a combination of these related concepts that would be conveniently interpreted by OntoCrawler to compare with content of the queried webpage, and if the compared outcomes were corresponding to any item among the four, we would infer the related concept “Education” as matched condition for web page querying.

For simplification and demonstration of all portal-related techniques, we narrow down the ontology scope to the Scientist ontology (i.e., Einstein), as shown in Fig. 2, the Scientist Ontology hierarchy was developed based on the method described in Uschold and Gruninger (1996). In the figure, nodes represent ontology concepts; links labeled with “is A” denotes the parent–child relationship between concepts, which allows inheritance of features from parent classes to child classes; links labeled otherwise represent some reference relationships from a concept to another concept, where “*” denotes multiple instance reference (Chang.

![Fig. 1. The ontology structure of scholars in Protégé.](image-url)
In addition to the Scientist ontology, we also defined a query ontology to facilitate the description of queries. Fig. 3 shows the query ontology, where the root node represents class “Query” which contains a set of fields defining the semantics of the class, each field representing one attribute of “Query”, e.g., QueryType, Name, etc. The links labeled with “io” represent relationship “instance_of”, relating an instance to the query class. The figure shows four query instances, including Who, What, Which, and Where, each standing for a specific solution pattern. For example, query instance “Who” has “People” as the value of its QueryType attribute, which means the answer to the query instance must belong to the People class. Finally, we have used Protégé’s APIs http://protege.stanford.edu/doc/pdk/api/ to develop a set of ontology services, which provide primitive functions to support the applications 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.

3. OntoPortal architecture

Fig. 4 illustrates the architecture of OntoPortal. It integrated three internal components: Portal Interface, Semantic Portal and OntoCrawler to rapidly and precisely collect Web information on Internet and capture true user’s intention and accordingly provide high-quality query answers to meet the user requests. Firstly, OntoCrawler employed the ontology-supported technique for actively providing comparison and verification for user’s keywords so as to up-rise precision rate of webpage searching. This technique has practically been installed in Google and Yahoo search engines and furthermore searched some related scholars’ personal WebPages and accordingly stored the results into some system databases to let the front-end systems to do advanced processes. Semantic Portal is responsible for producing three supporting knowledge bases to support Portal Interface to provide a better semantic portal. Finally, Portal Interface applies those knowledge bases during the interaction with the user by providing five different types of search mechanisms for retrieving high-quality query answers to meet the user requests.

3.1. OntoCrawler architecture

Fig. 5 shows the operation system structure of OntoCrawler, and related techniques and functions of every part were described as below:

1. **Action**: to transfer internal query into URI code, and then embed into Google’s query URL: an example as follow: http://www.google.com.tw/search?hl=zh-TW&q=%E5%AD%B8%E8%80%8B&meta=

![Fig. 5. System structure of OntoCrawler.](image-url)
(2) **LinkToGoogle**: to declare an URL object and add Google query URL on well transferred URI code, and then used a BufferedReader to read and used while loop to add String variable “line” line by line. Finally, output “line” as text file as final analysis reference. The file content was the html source file of the webpage.

(3) **RetrieveLinks**: to set an “int” variable “count” as 0, making the display of search progress convenient. Then the system would use regular expression to search for whether there are matched URL. But it could not retrieve all the linkages in one time out because of the Google webpage editing with indenting. So we used a “for” loop and ran for twice. The semantic of the two in regular expression were slightly different so as to completely fetch out related hyperlinks corresponding to the conditions. Finally, added all hyperlinks on String variable “RetrieveLink” and output the txt file to provide the system for further processing.

(4) **RetrieveContent**: wasting too much time on RetrieveContent processing and executing order would be problems to make system interface seem to entirely stop. So, system designing with “thread” got free from Swing thread events, and this made it possible to do some proper change of the interface when querying webpage. Afterwards, we set a Boolean variable “crawling” as “true”, it stood for executing the activity of webpage query and the initial condition of query bar was 0%. The Button (Java Class) caption was set “stop”; Textarea (Java Class) was set “not editable”. And then we used BufferedReader (Java Class) to read in “RetrieveLink” with “while” loop line by line, that meant we checked one URL link once a time and really linked the URL. After judging what kind coding of the webpage was, we read in the html source file of webpage with correct coding added on String variable s3 and output it as text file so as to let system conduct further processing. After completing all procedures mentioned above, we could use SearchMatches method (described later) to judge whether the webpage was located in the range we hoped to query; supposed the answer was “yes”, we would execute RemoveHTMLLabel (described later) to delete the html label from source file and remained only the text content so as to let system conduct further processing and analyzing. Finally, we collected the number of queried webpage and divided with total of the webpage and the mean we got was the percentage of query processing. Remember to clear RetrieveLink lest the next query should not link to the current URL. After judging what kind coding of the webpage was, we read in the html source file of webpage with correct coding added on String variable s3 and output it as text file so as to let system conduct further processing.

(5) **SearchMatches**: supporting RetrieveContent internal calling service to judge whether the webpage was the range we queried. It linked the ontology database and fetched out the content to compare content of s3 when using this linking method. If there were any return value corresponding to the value we set (usually we set 6 and this indicated 50% queried webpage matched up our query condition), and then system would return one “true” Boolean variable “matches”. That meant the webpage matched our query condition, on the other hand, if returned “false” meant the webpage did not match our query condition.

(6) **RemoveHTMLLabel**: just like SearchMatches, it supported RetrieveContent internal calling service and deleted html label in the html source file. In this system we used a serial of regular expression to remove html labels in s3 during internal calling function, and then we could get a webpage with pure text file.

3.2. Semantic Portal architecture and Portal Interface workflow

Fig. 6 illustrates the architecture of semantic-portal. OntoCrawler is responsible for collecting Web information and removing HTML tags and punctuations from the original web pages, and storing them in “Pre-processed WebPages Database.” Semantic WebPages generator then bases on the database to produce three supporting knowledge bases. The first knowledge base is “Stemmed WebPages Database” created by Stemmer (detailed later), which stems words and removes stop and noisy words on a pre-processed web page. The second is “POS (Parts-of-Speech)-attached WebPages Database” created by the POS Tagger (detailed later), which analyzes the POS of the words on a web page and transforms the web page into a list of words with POS. The third is “Annotated WebPages Database” created by Automatic Annotator (detailed later), which automatically identifies the concepts and relationships contained in a web page, derives a content summary for the web page, and constructs a semantic index for the web page. These knowledge bases, along with Ontology Database, support Portal Interface (Fig. 7) to provide a better Semantic Portal. In other words, Portal Interface applies these five knowledge bases during the interaction with the user by providing five different types of search mechanisms, including keyword search, synonym search, POS-constrained keyword search, natural language query, and semantic index search. The user can choose one of the five patterns to input his/her queries. Answer Generator finally produces semantic-directed answers to respond to the user query. Finally, Ontology Database provides the most fundamental semantics, as illustrated in Fig. 2.

3.2.1. Semantic Webpage Generator

Semantic Webpage Generator bases on the pro-processed WebPages database to produce three supporting knowledge bases for doing WebPages annotation through three components Stemmer, POS Tagger, and Automatic Annotator.
3.2.1.1. Stemmer. Fig. 8 illustrates the workflow of Stemmer. Qtag [http://www.english.bham.ac.uk/staff/omason/software/qtag.html] is used here to help remove stop words, passing only major terms to WordNet-based Stemmer. Specifically, Qtag employs probability with a sliding window (Tufis & Mason, 1998) to determine POS of the words in a web page. The output of Qtag contains detailed POS tags (Examples shown in Table 1), which are further categorized into five categories, namely, noun, verb, adjective, adverb, and DC (for Do not Care) for the purpose of removing stop words. Finally, WordNet-based Stemmer employs WordNet, which is a comprehensive vocabulary (WordNet 2.1., 2005), to transform the major terms into their stems and stores them in Stemmed WebPages Database.

3.2.1.2. POS Tagger. Fig. 9 illustrates the workflow of POS Tagger, which is very similar to the Stemmer. Qtag is used here to actually produce a POS for each word of a web page. WordNet-based Stemmer is then used as in Stemmer to do word stemming.

3.2.1.3. Automatic Annotator. Fig. 10 illustrates the architecture of Automatic Annotator. Concept Identifier and Relationship Identifier work together to extract the concepts and relationships contained in a web page, supported by the domain ontology. Text Summarizer employs Text-Miner, developed by IBM Intelligent Systems Corporation.

Table 1
Part of POS tags used by Stemmer and POS Tagger.

<table>
<thead>
<tr>
<th>POS Tag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BE</td>
</tr>
<tr>
<td>2</td>
<td>BEDR</td>
</tr>
<tr>
<td>3</td>
<td>BEDZ</td>
</tr>
<tr>
<td>4</td>
<td>BEG</td>
</tr>
<tr>
<td>5</td>
<td>BEM</td>
</tr>
<tr>
<td>6</td>
<td>BEN</td>
</tr>
<tr>
<td>7</td>
<td>BER</td>
</tr>
<tr>
<td>8</td>
<td>BEZ</td>
</tr>
<tr>
<td>9</td>
<td>CC</td>
</tr>
<tr>
<td>10</td>
<td>DO</td>
</tr>
<tr>
<td>11</td>
<td>DOD</td>
</tr>
<tr>
<td>12</td>
<td>DON</td>
</tr>
<tr>
<td>13</td>
<td>DOG</td>
</tr>
<tr>
<td>14</td>
<td>DOZ</td>
</tr>
<tr>
<td>15</td>
<td>DT</td>
</tr>
<tr>
<td>16</td>
<td>IN</td>
</tr>
<tr>
<td>17</td>
<td>MD</td>
</tr>
<tr>
<td>18</td>
<td>TO</td>
</tr>
<tr>
<td>19</td>
<td>WDT</td>
</tr>
<tr>
<td>20</td>
<td>WP</td>
</tr>
<tr>
<td>21</td>
<td>WPS</td>
</tr>
<tr>
<td>22</td>
<td>WRB</td>
</tr>
</tbody>
</table>
Miner for Text (1998), to identify important sentences in order to
derive a summary for the web page. Semantic Index Generator
identifies the most important sentence in the summary as the
semantic index, supported by the domain ontology again. Once
the semantic index is identified, it is treated as the center theme
of the webpage and we can use it to refine the summary. Basically,
we employ the vector space model (Salton, Wong, & Yang, 1975) to
calculate the correlations between the semantic index and the sen-
tences in the webpage summary, and accordingly adjust their
appearance order or remove some less relevant sentences. This
helps us develop a summary which contains sentences that are
not only highly-weighted but also highly relevant to the center
theme. Finally, XML Translator adds all of these into a web page
and transforms it into an annotated web page. Note that Automatic
Annotator not only annotates a web page, but also adds relevant
synonyms into the annotated web page, with the help of Concept
Identifier and the domain ontology (Chang, 2003).

4. System evaluations and demonstrations

The evaluation of the overall performance of our proposed sys-
tem involves lots of manpower and is time-consuming. Here, we
first focus on the performance evaluation of the most important
modules, for instance OntoCrawler of the OntoPortal architecture.
Our philosophy is that if OntoCrawler can precisely collect domain
related WebPages and then we not only can precisely and effect-
tively produce solutions with better semantics, but also can effi-
ciently improve the quality of the retrieved to meet the user
requests, which would be clear and detailed from system
demonstrations.

4.1. Evaluations on OntoCrawler

In this experiment, we compared our query technique with the
most popular query engines Google and Yahoo. We used ten differ-
ent keywords as webpage query of related information. First of all,
we keyed in the same set of keywords in Google and Yahoo. The
comparison result was shown in Table 2, in which NWs means the
number of total returned WebPages; NWc meant number of
correct returned WebPages; NWr meant number of related
returned WebPages but they were not necessarily the correct web-
page. Here, we used Eqs. (1) and (2) to define Precision Rate, Rp
and Recall Rate, Rr. After comparing returned webpage one after
another, we got the average Rp and Rr of Google were 1.90% and
3.70% while Yahoo were 1.80% and 3.90%, respectively.

\[
Rp = \frac{NW_c}{NW_r} \quad (1)
\]

\[
Rr = \frac{NW_c + NW_r}{NW_t} \quad (2)
\]

When we used Google and Yahoo as base of webpage query, but on
OntoCrawler we keyed in the same set of keywords. The compari-
son results were shown in Table 3. After comparing returned web-
page one after another, we got the average Rp and Rr of OntoCrawler
on Google were 21.93% and 38.51% while on Yahoo were 14.48% and
35.33%, respectively. After comparing (Tables 2 and 3), the query
precision and recall rate for Google researching engine through
assistance of OntoCrawler has up-rise around 91% (21.93%/14.48%)
and 38.51%/35.33% while conditions of Yahoo are about 89% (38.51%/35.
33%). From the above comparison, it indicated that OntoCraw-
lver offered more precision and recall rate than Google and Yahoo on
webpage searching: in addition, the technique we proposed has its
availability.

4.2. Demonstrations on OntoPortal prototype

First, suppose the user chooses to query by keywords, Query
Manager will directly invoke Keyword Matcher, which uses full-
text search to compare the keywords with the words in all of the
web pages stored in Pre-Processed WebPages Database. This pro-
cess is the same as the classical keyword match employed by most
general portals. Two major issues are associated with the keyword-
based query method, however. First, it only retrieves web pages
which contain syntactically the same keywords. Second, keywords of
different parts of speech may represent different meanings. The
second issue can be easily coped with by allowing the user to use
the Word+POS Match method, which requires the user to associate
a part-of-speech with each input keyword. Query Manager then
invokes Keyword+POS Matcher to search in POS-attached WebPages

<table>
<thead>
<tr>
<th>No.</th>
<th>Affiliation</th>
<th>Keyword</th>
<th>NWc</th>
<th>NWr</th>
<th>NWt</th>
<th>Rp (%)</th>
<th>Rr (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Google</td>
<td>1 Dept. of Computer Science and Information Engineering, National Taiwan University</td>
<td>郭大巍 (Tei-Wei Kuo)</td>
<td>2</td>
<td>100</td>
<td>100</td>
<td>1.00</td>
<td>3.00</td>
</tr>
<tr>
<td>2 Dept. of Chemical Engineering, National Cheng Kung University</td>
<td>劉瑞祥 (J.H. Liu)</td>
<td>3</td>
<td>100</td>
<td>100</td>
<td>3.00</td>
<td>4.00</td>
<td></td>
</tr>
<tr>
<td>3 Dept. of Electronics Engineering, National Tsing Hua University</td>
<td>洪善豐 (Sheng-fu Horng)</td>
<td>2</td>
<td>100</td>
<td>100</td>
<td>2.00</td>
<td>3.00</td>
<td></td>
</tr>
<tr>
<td>4 Dept. of Photonics, National Chiao Tung University</td>
<td>許振穎 (Ken Y. Hsu)</td>
<td>1</td>
<td>4</td>
<td>100</td>
<td>100</td>
<td>1.00</td>
<td>5.00</td>
</tr>
<tr>
<td>5 Dept. of International Business, National ChengChi University</td>
<td>賴文傑 (Jyh-Shen Chiu)</td>
<td>2</td>
<td>1</td>
<td>100</td>
<td>2.00</td>
<td>2.00</td>
<td></td>
</tr>
<tr>
<td>6 Dept. of Business Administration National Central University</td>
<td>林明杰 (Ming-ji James Lin)</td>
<td>3</td>
<td>3</td>
<td>100</td>
<td>3.00</td>
<td>6.00</td>
<td></td>
</tr>
<tr>
<td>7 Graduate Institution of Finance, National Taiwan University of Science and Technology</td>
<td>林丙耀 (Bing-Huei Lin)</td>
<td>3</td>
<td>3</td>
<td>100</td>
<td>3.00</td>
<td>6.00</td>
<td></td>
</tr>
<tr>
<td>8 Dept. of Business Management, National Taipei University of Technology</td>
<td>謝慶榕 (Ching-Jui Keng)</td>
<td>1</td>
<td>3</td>
<td>100</td>
<td>1.00</td>
<td>4.00</td>
<td></td>
</tr>
<tr>
<td>9 Dept. of Electrical Engineering, National Chung Cheng University</td>
<td>吳承霖 (Cheng Shong Wu)</td>
<td>1</td>
<td>0</td>
<td>100</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>10 Dept. of Information Management, National Sun Yat-Sen University</td>
<td>鄭寶強 (Bing-Chiang Jeng)</td>
<td>2</td>
<td>1</td>
<td>100</td>
<td>2.00</td>
<td>3.00</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.90</td>
<td>3.70</td>
</tr>
</tbody>
</table>

| (b) Yahoo | 1 Dept. of Computer Science and Information Engineering, National Taiwan University | 郭大巍 (Tei-Wei Kuo) | 3 | 1 | 100 | 1.00 | 4.00 |
| 2 Dept. of Chemical Engineering, National Cheng Kung University | 劉瑞祥 (J.H. Liu) | 2 | 2 | 100 | 2.00 | 4.00 |
| 3 Dept. of Electronics Engineering, National Tsing Hua University | 洪善豐 (Sheng-fu Horng) | 3 | 1 | 100 | 3.00 | 4.00 |
| 4 Dept. of Photonics, National Chiao Tung University | 許振穎 (Ken Y. Hsu) | 2 | 2 | 100 | 2.00 | 4.00 |
| 5 Dept. of International Business, National ChengChi University | 賴文傑 (Jyh-Shen Chiu) | 1 | 1 | 100 | 1.00 | 2.00 |
| 6 Dept. of Business Administration National Central University | 林明杰 (Ming-ji James Lin) | 0 | 5 | 100 | 0.00 | 5.00 |
| 7 Graduate Institution of Finance, National Taiwan University of Science and Technology | 林丙耀 (Bing-Huei Lin) | 5 | 2 | 100 | 5.00 | 7.00 |
| 8 Dept. of Business Management, National Taipei University of Technology | 謝慶榕 (Ching-Jui Keng) | 0 | 3 | 100 | 0.00 | 3.00 |
| 9 Dept. of Electrical Engineering, National Chung Cheng University | 吳承霖 (Cheng Shong Wu) | 1 | 0 | 100 | 1.00 | 1.00 |
| 10 Dept. of Information Management, National Sun Yat-Sen University | 鄭寶強 (Bing-Chiang Jeng) | 3 | 2 | 100 | 3.00 | 5.00 |
| Average | | | | | | 1.80 | 3.90 |
Database for web pages which contain the keywords appearing in
the correct part of speech, as illustrated in Fig. 11.

We employ two methods to cope with the first issue in our sys-
tem. The first method not only invokes Synonyms Matcher to
search in Annotated WebPages Database for web pages containing
the synonyms of the entered keywords, but also tackles morpho-
logical changes by returning a web page containing “writing”, gi-
gen input keyword “wrote”, as shown in Fig. 12.

Alternatively, we allow the user to post natural language query
in order to deal with the first issue. The user can place a natural
language query by using NL-Based Query method, when Query
Manager invokes Pseudo NL Answerer to propose answers.

Table 3
The comparison of the front 100 queries between based on Google and Yahoo.

<table>
<thead>
<tr>
<th>No.</th>
<th>Affiliation</th>
<th>Keyword</th>
<th>NW_T</th>
<th>NW_N</th>
<th>NW_E</th>
<th>R_T (%)</th>
<th>R_N (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) OntoCrawler on Google</td>
<td>Dept. of Computer Science and Information Engineering, National Taiwan University</td>
<td>賴大維 (Tei-Wei Kuo)</td>
<td>1</td>
<td>1</td>
<td>9</td>
<td>11.11</td>
<td>22.22</td>
</tr>
<tr>
<td>1</td>
<td>Dept. of Chemical Engineering, National Cheng Kung University</td>
<td>劉瑞祥 (J.H. Liu)</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>50.00</td>
<td>75.00</td>
</tr>
<tr>
<td>2</td>
<td>Dept. of Electronics Engineering, National Tsing Hua University</td>
<td>洪勝霖 (Sheng-fu Horng)</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>28.57</td>
<td>28.57</td>
</tr>
<tr>
<td>3</td>
<td>Dept. of Photonics, National Chiao Tung University</td>
<td>韋根玉 (Ken Y. Hsu)</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>16.67</td>
<td>33.33</td>
</tr>
<tr>
<td>4</td>
<td>Dept. of International Business, National ChengChi University</td>
<td>邱志聖 (Yih-Shen Chiou)</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>25.00</td>
<td>25.00</td>
</tr>
<tr>
<td>5</td>
<td>Dept. of Business Administration, National Central University</td>
<td>林明杰 (Ming-ji James Lin)</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>25.00</td>
<td>75.00</td>
</tr>
<tr>
<td>6</td>
<td>Graduate Institution of Finance, National Taiwan University of Science and Technology</td>
<td>林輝 (Bing-Huei Lin)</td>
<td>2</td>
<td>3</td>
<td>11</td>
<td>18.18</td>
<td>45.45</td>
</tr>
<tr>
<td>7</td>
<td>Dept. of Business Management, National Taipei University of Technology</td>
<td>莊耀瑞 (Ching-Jui Keng)</td>
<td>1</td>
<td>3</td>
<td>14</td>
<td>7.14</td>
<td>28.57</td>
</tr>
<tr>
<td>8</td>
<td>Dept. of Electrical Engineering, National Chung Cheng University</td>
<td>吳承恩 (Cheng Shong Wu)</td>
<td>1</td>
<td>0</td>
<td>11</td>
<td>9.09</td>
<td>9.09</td>
</tr>
<tr>
<td>9</td>
<td>Dept. of Information Management, National Sun Yat-Sen University</td>
<td>鄭炳強 (BingChiang Jeng)</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>28.57</td>
<td>42.86</td>
</tr>
<tr>
<td>10</td>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>21.93</td>
<td>38.51</td>
</tr>
<tr>
<td>(b) OntoCrawler on Yahoo</td>
<td>Dept. of Computer Science and Information Engineering, National Taiwan University</td>
<td>賴大維 (Tei-Wei Kuo)</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td>12.50</td>
<td>50.00</td>
</tr>
<tr>
<td>1</td>
<td>Dept. of Chemical Engineering, National Cheng Kung University</td>
<td>劉瑞祥 (J.H. Liu)</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>28.57</td>
<td>42.86</td>
</tr>
<tr>
<td>2</td>
<td>Dept. of Electronics Engineering, National Tsing Hua University</td>
<td>洪勝霖 (Sheng-fu Horng)</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>22.22</td>
<td>33.33</td>
</tr>
<tr>
<td>3</td>
<td>Dept. of Photonics, National Chiao Tung University</td>
<td>韋根玉 (Ken Y. Hsu)</td>
<td>2</td>
<td>3</td>
<td>18</td>
<td>11.11</td>
<td>27.78</td>
</tr>
<tr>
<td>4</td>
<td>Dept. of International Business, National ChengChi University</td>
<td>邱志聖 (Yih-Shen Chiou)</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>20.00</td>
<td>40.00</td>
</tr>
<tr>
<td>5</td>
<td>Dept. of Business Administration, National Central University</td>
<td>林明杰 (Ming-ji James Lin)</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0.00</td>
<td>60.00</td>
</tr>
<tr>
<td>6</td>
<td>Graduate Institution of Finance, National Taiwan University of Science and Technology</td>
<td>林輝 (Bing-Huei Lin)</td>
<td>4</td>
<td>2</td>
<td>13</td>
<td>30.77</td>
<td>46.15</td>
</tr>
<tr>
<td>7</td>
<td>Dept. of Business Management, National Taipei University of Technology</td>
<td>莊耀瑞 (Ching-Jui Keng)</td>
<td>0</td>
<td>4</td>
<td>19</td>
<td>0.00</td>
<td>21.05</td>
</tr>
<tr>
<td>8</td>
<td>Dept. of Electrical Engineering, National Chung Cheng University</td>
<td>吳承恩 (Cheng Shong Wu)</td>
<td>1</td>
<td>0</td>
<td>14</td>
<td>7.14</td>
<td>7.14</td>
</tr>
<tr>
<td>9</td>
<td>Dept. of Information Management, National Sun Yat-Sen University</td>
<td>鄭炳強 (BingChiang Jeng)</td>
<td>2</td>
<td>2</td>
<td>16</td>
<td>12.50</td>
<td>25.00</td>
</tr>
<tr>
<td>10</td>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14.48</td>
<td>35.33</td>
</tr>
</tbody>
</table>

In addition to the above four user interface methods, we define
a new keyword query method called semantic index match. The
user can choose the method and enter usual keywords; Query
Manager will invoke Semantic Index Searcher to search in Anno-
tated WebPages Database for all web pages whose semantic index
contains the keywords and accordingly displays the semantic indi-
ces for the user, as illustrated in Fig. 14. Note that the return page
contains a list of URLs, each followed by its semantic index. This
approach is different from general keyword-based search engines,
which usually return the title and the first sentence of each web
page.
5. Related works and comparisons

The Semantic Web or semantic portals heavily rely on webpage annotation, which inevitably becomes one of the hottest research topics. SHOE (Simple HTML Ontology Extension) is equipped with a manual Knowledge Annotator SHOE, 2000, which is a Java program allowing the user to annotate web pages graphically. OntoMat–Annotizer (OntoMat, 2003) is a user-friendly interactive webpage annotation tool. It supports the user with the task of creating and maintaining ontology-based DAML + OIL markups, e.g., creation of DAML-instances, attributes and relationships. Annotea (2001) in LEAD (Live and Early ADoption) uses special RDF annotation schema to add comments (a kind of annotation) as metadata. S-CREAM (Semi-automatic CREAtion of Metadata) (Handschuh, Staab, & Ciravegna, 2002) allows creation of metadata and is trainable for a specific domain. SEAL (SEmantic portAL) (Hartmann & Sure, 2004) is a generic approach for developing semantic portal. PASS (Portal with Access to Semantic Search) (Pinheiro & Moura, 2004) uses components of domain ontologies stored in the portal to expand terms during the search process. ONTODELLA (Viljanen, Kansala, Hyvonen, & Makela, 2006) presents a logic-based approach for creating view projections and semantic linking for creating semantic enhanced web applications, such as semantic portals. Khan & Malik (2007) propose an approach to leverage the development life cycle of a semantic portal through the use of Object Management Group’s Model Driven Architecture. The above annotation mechanisms are either manual or semi-automatic, which impose heavy labor on humans facing a voluminous amount of web pages.

Some automatic annotation mechanisms appeared in the literature. For instance, (Lerman, Getoor, Minton, & Knoblock, 2004) describes two algorithms which use redundancies in the content of table and detail pages to help information extraction. A strategy was proposed based on active recommendation to support the semantic annotation of contested knowledge to promote annotators’ interest (Sereno, Shum, & Motta, 2004). Conflict detection pat-
terns were devised based on different datum, ontology at different inference levels and proposed the corresponding automatic conflict resolution strategies for image annotation (Lee & Soo, 2006).

Amilcare (2002) is an information extraction system which helps automatic annotation with learning capability. Its power of linguistic analysis stems from Gate, which performs tokenization, sentence identification, POS tagging, gazetteer lookup and named entity recognition. Gate functions virtually in the same way as the combination of Qtag, Concept/Relationship identifier, and Text-Miner in our system during automatic annotation. However, since Gate is easier to be used, it will be planned to employ in our future semantic-based systems. In addition, Amilcare is salient in employing the (LP)² algorithm, a supervised algorithm that falls into a class of Wrapper Induction System using LazyNLP. (LP)² can induce rules which help insert annotations in the texts and rules which correct mistakes and imprecision in the annotations generated by the former rules. We reckon that as an information extraction system, Amilcare works very similarly to part of our system. Its adaptive feature, supported by the underlying learning capability, however, provides yet another level of automation in webpage annotation and deserves more attention, such as growing hierarchical self-organizing maps: an unsupervised learning approach mentioned in Antonio et al. (2008).

6. Conclusions and discussions

An ontology-supported portal architecture: OntoPortal was proposed, which integrated the techniques of ontology, linguistics, and focused crawler to rapidly and precisely collect information on Internet and capture true user’s intention and accordingly provide high-quality query answers to meet the user requests. Second, Semantic Portal is a fully automatic annotation tool. The tool not only helps develop annotated web pages during the construction of the portal, but also provides the user with an automatic online annotation process. The user can use the process to annotate the web pages specific to his/her interests. Second, it enriches webpage annotation by content summary and semantic index as two facets of representative semantics of a web page. Finally, it improves the traditional keyword-based retrieval mechanism by providing word-stemming, parts-of-speech, natural language processing, and semantic index search. These extensions help capture the true intention of the user and return better information with semantics.

For easy demonstration of the techniques in our semantic-portal prototype, the current implementation runs on a very simple “Scholar: Scientist” domain. However, we believe even if the domain is scaled up, our techniques still be applicable. The idea is this: we are not directly scaling up our ontology. Instead we can create a complex system by integrating a set of simple systems through a multi-agent architecture which is supported by a set of simple domain ontologies. By exploiting the capability of Protégé-2000, which supports the creation, extension and cooperation of a set of domain ontologies, we really need not make much change to our ontology-supported semantic portal in order to transform it into a complex portal. What we really need to focus on is how to make a set of ontology-supported systems cooperate effectively, which is under our investigation.

The ontology plays a very important role in our annotation technique. One major difficulty is its construction, which currently still relies on the help of domain experts. We are planning to investigate the technique of automatic ontology construction in the future, such as ontological engineering of Web portals mentioned in Francisco, Rodrigo, Leonardo, Jesualdo, and Dagoberto (2006). We believe our automatic annotation technique, when coupled with the automatic ontology construction technique, can help proliferate the techniques of Semantic Web and in turn promote the development of better Semantic Portals.

Acknowledgments

The author would like to thank Y.H. Chang, T.A. Chen, and C.F. Wu for their assistances in system implementation. This work
References


Developing of an Ontological Focused-Crawler for Ubiquitous Services

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

Abstract

This paper proposes the use of several ontological techniques to provide a semantic level solution for a search agent so that it can provide fast, precise, and stable search results in ubiquitous information environments. An ontological focused-crawler had been developed according to these techniques, which can benefit both user requests and domain semantics. Equipped with these techniques, the crawler can manifest the following interesting features: ontology-supported construction of website models, website models-supported website model expansion, and website models-supported webpage retrieval.

1. Introduction

In ubiquitous information environments, the user expects to spend short time in 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 overwhelsms the user. The user usually gets frustrated after a series of visits on the entries, when he discovers that dead entries are everywhere, irrelevant entries are equally abundant, what he gets is not exactly what he wants, etc. 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 [20].

In general, current search engines face two fundamental problems. Firstly, the index structures are usually very different from what the user conjectures about his problems. Secondly, 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 [12], RDF [4], DOM [1], Dublin metatag [21], and WOM [14], 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 [17], DAML [6], DAML+OIL [7] and the concept of ontology stand for a possible rescue to the attribution of information semantics. This paper proposes the use of several ontological techniques to provide a semantic level solution for a search agent so that it can provide fast, precise, and stable search results in ubiquitous information environments.

We notice that the concept of crawler is mostly used in the Web systems that work on information gathering or integration to improve their gathering processes or the search results from ubiquitous information environments. For instance, Dominos [10] 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 et al. [9] developed an ontology-support Web crawler with an 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. Ubi-Crawler [3], 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. This paper developed an ontological focused-crawler using website models as the core technique, which can help search agents successfully tackle the problems of search scope and user interests. The Personal Computer (PC) domain is chosen as the target application of our focused-crawler and will be used for explanation in the remaining sections.

2. Domain Ontology as the Down-to-the-Earth Semantics

![Fig. 1 Part of PC ontology](image)

Ontology is a method of conceptualization on a specific domain [16]. It plays diverse roles in developing intelligent systems, for example, knowledge sharing and reusing [8,11], semantic analysis of languages [15], 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 [2]. We had outlined a principle construction procedure in [22]; following the procedure we had developed an ontology for the PC domain. Fig. 1(a) 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 defined them as the detailed ontology of the corresponding class. Fig. 1(b) exemplifies the detailed ontology for the concept of “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,
3. Website Model and Construction

A website model contains a website profile and a set of webpage profiles. Fig. 2(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. 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.

Update Time/Date remembers when the webpage was modified last time. The statistics information section stores statistics about HTML tag properties. 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. Domain Mark is used to remember whether the webpage belongs to a specific domain. 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’s turn to the website profile. WebNo identifies a website. 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. WebType identifies one of the six Web types as used in the webpage profile. Tree_Level Limit keeps the search agent from exploring too deeply. Update Time/Date remembers when the website was modified last time. 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. 2(b) illustrates how website profiles and webpage profiles are structured.

During the construction and expansion process of a website model, we need to extract primitive webpage information as well as to perform statistics. We also need to transform the original webpage into a tag-free document for annotation of ontology information. These activities involve intensive consultation of domain ontology. In order to facilitate these activities, we have re-organized the ontology structure into Fig. 3, which stresses on how concept attributes are related to class identification. In the figure, each square node contains a set of representative ontology features for a specific concept, while each oval node contains related ontology features between two concepts or among more concepts. This design clearly structures semantics between ontology classes and their relationships and can serve as a fast semantics decision mechanism for website expansion.

The semantics decision process sometimes involves the decision of the class for a webpage or a website. We had proposed an ontology-directed classification mechanism, namely, OntoClassifier [24] to solve the problem. It classifies a webpage in two stages. The first stage uses representative ontology features. We employ the level threshold to limit the number of ontology features to be involved in this stage. The basic idea of classification of the first stage is defined by Eq. (1). In the equation, OntoMatch(d,C) is defined by Eq. (2), which calculates the number of ontological features of class C that appears in
webpage \(d\), where \(M(w|c)\) returns 1 if word \(w\) of \(d\) is contained in class \(C\). Thus, Eq. (1) returns class \(C\) for webpage \(d\) if \(C\) has the largest number of ontology features appearing in \(d\). Note that not all classes have the same number of ontology features; we have added \(\#w_C\), the number of words in each class \(C\), for normalization. Also note that Eq. (1) only compares those classes with more than three ontology features appearing in \(d\), i.e., it filters less possible classes. As to why classes with less than three features appearing in \(d\) are filtered, we refer to Joachims’ concept that the classification process only has to consider the term with appearance frequency larger than three [13].

\[
\begin{align*}
\text{OntoMatch}(d,C) &= \sum_{w \in d} M(w,C) \\
\text{OntoMatch}(d,C) &= \frac{\text{OntoMatch}(d,C)}{\#w_C} > 3
\end{align*}
\]

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

\[
\begin{align*}
\text{HOntoTFIDF}(d,C) &= \arg \max_{c} \text{OntoTFIDF}(d,C) \\
\text{OntoTFIDF}(d,C) &= \sum_{c \in C} \frac{TF(w|C)}{L_w} \sum_{w \in d} \frac{TF(w|C)}{TF(w|F)} \sum_{w \in C} \frac{TF(w|F)}{TF(w|F)} \sum_{w \in C} \frac{TF(w|F)}{TF(w|F)}
\end{align*}
\]

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; detailed descriptions and architectures please refer to [25].

4. Website Models Application

4.1 Focused Web Crawling Supported by Website Models

In order to effectively use the website models to narrow down the search scope, we proposed a new focused-crawler as shown in Fig. 4, which can feature 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.

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

where \(U\) represents a URL or search string; and \(F\) identifies the way \(U\) is obtained as shown in Table 1, 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 the 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. 5 (to be explained in Section 4.2) to calculate \(S_F(U)\) for each \(U\). The assignment of \(S_F(U)\) in the latter case is more complicated; we will describe it in Section 4.3.

<table>
<thead>
<tr>
<th>Table 1 Basic Weighting for URLs to be explored</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Type</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>Search Strings</td>
</tr>
<tr>
<td>Website Model URLs</td>
</tr>
</tbody>
</table>

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 query 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.

4.2 User-Oriented Web Search supported by Website Models
Conventional Information Retrieval research has mainly been based on (computer-readable) text [19] 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 2 illustrates a simple example. More complicated expansion adds ontology concepts according to their relationships with the query terms. The most used relationships follow the inheritance structure. For instance, if more than half of the sub-concepts of a concept appear in a user query, we add the concept to the query too.

Table 2 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</td>
<td>Mainboard Motherboard</td>
</tr>
<tr>
<td>ABIT</td>
<td>ABIT</td>
</tr>
</tbody>
</table>

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 oriented 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. 5 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 the nature.

$$R_{S,D} = \frac{N_{S,D}}{N_S}$$  \hspace{1cm} (6)

### 4.3 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 4-phase progressive strategy to autonomously expand the website models. The first phase uses Eq. (7) to calculate $S_f(U)$ 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 $S$.

$$S_f(U) = \sum_{C \in D} R_{S,D} \times \left(1 - P_{S,D}(C)\right), \hspace{1cm} U \in S \text{ and } P_{S,D}(C) \neq 0$$  \hspace{1cm} (7)

where, $C$ is a concept of domain $D$, $R_{S,D}$ was defined in Eq. (6), and $P_{S,D}(C)$ is defined by Eq. (8). $P_{S,D}(C)$ 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. 6 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.

$$P_{S,D}(C) = \frac{N_{S,C}}{N_{S,D}}$$  \hspace{1cm} (8)

Fig. 5 User-oriented webpage expansion strategy supported by the website models

Fig. 6 Algorithm for calculating $N_{S,C}$

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 $P_{S,D}(C)$. In summary, the first phase prefers expanding 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. 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 $S_f(U)$ for each outbound hyperlink $U$ of some webpages that is stored in an indefinite
\[ S_2(U) = \frac{\sum_{C \in D} \text{Anchor}(U, C) \times R_{S,D} \times P_{S,D}(C)}{T} \]

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 \( D \). 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 outbound 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. In short, the third phase tends to collect every webpage that is referred to by the webpages in the website models.

\[ S_2(U) = \sum_{C \in D} \text{Anchor}(U, C) \times R_{S,D} \times P_{S,D}(C), \quad U \in T \]

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 [5] and define a weight for each web type as shown in Table 3. This phase then uses Eq. (11) to assign a \( S_2(U) \) to each \( U \) which belongs to a specific website type \( T \).

4.4 Webpage Retrieval from Website Models

Webpage retrieval concerns the way of providing most-wanted 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 [19]. 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 PageRank [18] 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. 7 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 domain concepts the user are really interested in and use the information to fast locate user interested webpages. In short, we use the second stage of OntoClassifier along with a threshold, said \( THU \), to limit the best classes (concepts) 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. Finally, we can employ the identified ontology features in a user query to properly rank the webpages for the user using the ranking method [20].

5. User-Satisfaction Evaluation

Table 4 shows the comparison of user satisfaction of our system prototype against other search engines. In the table, \( S_p \) 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 ontological focused-crawler, 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 prototype system, in general, retrieves more correct webpages in almost all classes.

6. Conclusions
We have described how ontology-supported website models can effectively support Web search, which is different from website model content, construction, and application over our previous works [23]. 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 an ontological focused-crawler, which employs domain ontology-supported website models as the core technology to search for Web resources that are both user-interested and domain-oriented in ubiquitous information environments. It features the following interesting characteristics. 1) Ontology-supported construction of website models. By this, we 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 [24]. 2) Website models-supported Website model expansion. By this, we take into account both user interests and domain specificity. The core technique here is to employ progressive strategies to do user query-driven webpage expansion, autonomous website expansion, and query results exploitation to effectively expand the website models [25]. 3) Website models-supported Webpage Retrieval. By this, we leverage the power of ontology features as a fast index structure to locate most-wanted webpages for the user. In addition, our ontology construction is based on a set of pre-collected webpages on a specific domain; it is hard to evaluate how critical this collection process is to the nature of different domains. We are planning to employ the technique of automatic ontology evolution to help studying the robustness of our ontology.

Acknowledgements

The author would like to thank Yung Ting, Jr-Well Wu, Chung-Min Wang, and Yai-Hui Chang for their assistance in system implementation. This work was supported by the National Science Council, R.O.C., under Grants NSC-89-2213-E-011-059, NSC-89-2218-E-011-014, and NSC-95-2221-E-129-019.

References


