# Bootstrapping Domain Ontology for Semantic Web Services from Source Web Sites

Wensheng Wu[1], AnHai Doan[1], Clement Yu[2], and Weiyi Meng[3]

[1] University of Illinois, Urbana, USA
[2] University of Illinois, Chicago, USA
[3] Binghamton University, Binghamton, USA

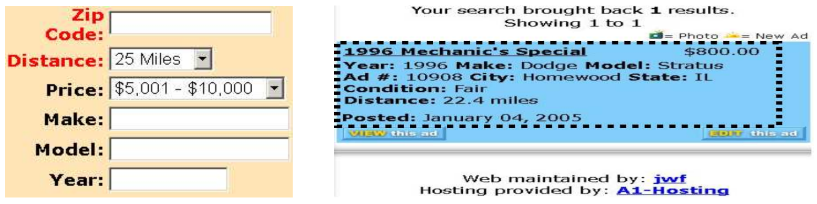
**Abstract.** The vision of Semantic Web services promises a network of interoperable Web services over different sources. A major challenge to the realization of this vision is the lack of automated means of acquiring domain ontologies necessary for marking up the Web services. In this paper, we propose the DeepMiner system which learns domain ontologies from the source Web sites. Given a set of sources in a domain of interest, DeepMiner first learns a base ontology from their query interfaces. It then grows the current ontology by probing the sources and discovering additional concepts and instances from the data pages retrieved from the sources. We have evaluated DeepMiner in several real-world domains. Preliminary results indicate that DeepMiner discovers concepts and instances with high accuracy.

## 1 Introduction

Past few years have seen an increasingly widespread deployment of Web services in the e-commerce marketplace such as travel reservation, book selling, and car sale services [21]. Among the most prominent contributing factors are several XML-based standards, such as WSDL [26], SOAP [20], and UDDI [22], which greatly facilitate the specification, invocation, and discovery of Web services. Nevertheless, the interoperability of Web services remains a grand challenge.

A key issue in enabling automatic interoperation among Web services is to *semantically* mark up the services with shared ontologies. These ontologies typically fall into two categories: service ontology and domain ontology. The *service ontology* provides generic framework and language constructs for describing the modeling aspects of Web services, including process management, complex service composition, and security enforcement. Some well-known efforts are OWL-S [5], WSFL [11], and WSMF [9]. The *domain ontology* describes concepts and concept relationships in the application domain, and facilitate the semantic markups on the domain-specific aspects of Web services such as service categories, semantic types of parameters, etc. Clearly, such semantic markups are crucial to the interoperation of the Web services.

Automatic acquisition of domain ontologies is a well-known challenging problem. To address this challenge, this paper proposes DeepMiner, a system for an



Zip Code:
Distance: 25 Miles
Price: $5,001 - $10,000
Make:
Model:
Year:

Your search brought back **1** results.
Showing 1 to 1

= Photo = New Ad

**1996 Mechanic's Special** $800.00
**Year:** 1996 **Make:** Dodge **Model:** Stratus
**Ad #:** 10908 **City:** Homewood **State:** IL
**Condition:** Fair
**Distance:** 22.4 miles
**Posted:** January 04, 2005

Web maintained by: jwf
Hosting provided by: A1-Hosting

(a) Its query interface (b) A snippet of a data page

**Fig. 1: A car sale Web site: query interface and data page**

incremental learning of domain ontologies for semantically marking up the Web services. DeepMiner is motivated by the following observations. First, we observe that many sources, which may potentially provide Web services, typically have already been providing similar services in their Web sites through query interface (e.g. in HTML form) and Web browser support. To illustrate, consider buying a car through a dealership's Web site. The purchasing may be conducted by first specifying some information on the desired vehicle such as make, model, and pricing, on its query interface (Figure 1.a). Next, the source may respond with the search result, i.e., a list of *data pages* (e.g., Figure 1.b), which typically contain detailed information on the qualified vehicles. The user may then browse the search result and place the order on the selected vehicle (e.g. through another HTML form).

Second, we observe that query interfaces and data pages of the sources often contain rich information on concepts, instances, and concept relationships in the application domain. For example, there are six attributes in Figure 1.a, each is denoted with a label and corresponds to a different concept. Some attributes may also have instances, e.g., Distance has instances such as 25 Miles. Further, the data page in Figure 1.b contains many additional concepts such as City, State, Condition, etc., and instances such as Homewood for City and Fair for Condition. Finally, the relative placement of attributes in the interface and data pages indicates their relationships, e.g., closely related attributes, such as Make and Model (both describe the vehicle), Zip Code and Distance (both concern the location of the dealership), are typically placed near to each other.

Based on the above observations, the goal of DeepMiner is to effectively learn a domain ontology from interfaces and data pages of a set of domain sources. Achieving this goal requires DeepMiner to make several innovations.

– **Incremental learning:** As observed above, the knowledge acquired from source interfaces only is often incomplete since data pages of the sources may contain many additional information. Further, different sources may contain a different set of concepts and instances. As such, DeepMiner learns the domain ontology in a snowballing fashion: first, it learns a base ontology from source interfaces; it then grows the current ontology by probing the sources and learning additional concepts and instances from the data pages retrieved from the sources.

– **Handling heterogeneities among sources:** Due to the autonomous nature of sources, the same concept may be represented quite differently over different sources. Another major challenge is thus to identify the semantic correspondences between concepts learned from different sources. To address this challenge, DeepMiner employs a clustering algorithm to effectively discover unique concepts over different interfaces. The learned ontology is then exploited for discovering new concepts and instances from data pages.
– **Knowledge-driven extraction:** Extracting concepts and instances from data pages is significantly more challenging than from query interfaces (since concepts and instances on an interface are typically enclosed in a form construct). To address this challenge, DeepMiner first exploits the current ontology to train concept classifiers. The concept classifiers are then employed to effectively identify regions of a data page where concepts and instances are located, discover presentation patterns, and perform the extraction.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 defines the problem. Sections 4–5 describe the DeepMiner system. Empirical evaluation is reported in Section 6. Section 7 discusses the limitations of the current system. Section 8 concludes the paper.

## 2 Related Work

The problem of semantically marking up Web services is fundamental to the automated discovery and interoperation of Web services and e-services. As such, it is being actively researched [3, 4, 7, 8, 12, 14, 19, 23, 24].

There have been some efforts in learning domain ontology for Web services. Our work is most closely related to [17], but different in several aspects. First, [17] learns domain ontology from the documentations which might accompany the descriptions of Web services, while our work exploits the information from the source Web sites. Second, we extract concepts and instances from semi-structured data over source interfaces and data pages, while [17] learns ontology from natural language texts.

[15] proposes METEOR_S, a framework for annotating WSDL files with concepts from an existing domain ontology. The mappings between elements in the WSDL files and the concepts in the ontology are identified by exploiting a suite of matchers such as token matcher, synonym finder, and n-gram matcher. [10] employs several machine learning algorithms for the semantic annotation of attributes in source interfaces. The annotation relies on a manually constructed domain ontology. Our work is complementary to these works in that we aim to automatically learn a domain ontology from the information on the source Web sites. The learned ontology can then be utilized to annotate the Web services.

There are several previous work on extracting instances and their labels from data pages [1, 25]. A *fundamental* difference between these work and ours is that we utilize existing knowledge in the growing ontology to effectively identify data regions and occurrences of instances and labels on the data pages. We believe that such a *semantics-driven* approach is also more efficient than their template-induction algorithm which can have an exponential complexity [6].

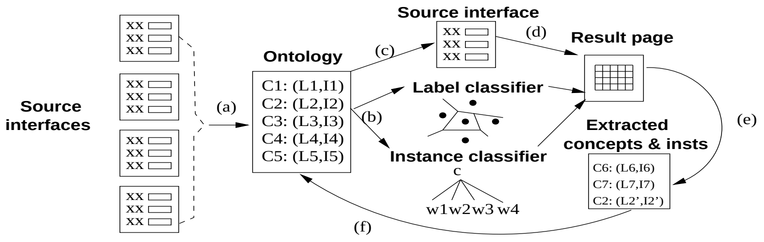


**Fig. 2: The DeepMiner architecture**

The problem of matching interface attributes has also been studied in the context of integrating *deep Web* sources (e.g., [27]). Our work extends these works in the sense that the learned domain ontology can be used to construct a global schema for the sources.

## 3 Problem Definition

We consider the problem of learning a domain ontology from a given set of sources in a domain of interest. The learned domain ontology should have the following components: (1) *concepts:* e.g. make, model, and class are concepts of the auto sale domain. (2) *instances of concept:* e.g. Honda and Ford are instances of the concept make. (3) *synonyms:* e.g. the concept make may also be denoted by brand, car manufacturer, etc. (4) *statistics:* i.e., how frequent each concept and its instances appear in the domain. (5) *data types:* of the concept instances, e.g., instances of price are monetary values while instances of year are four-digit numbers. (6) *concept relationships:* which include the grouping (e.g, make and model), precedence (e.g., make should be presented before model), as well as the taxonomic relationships between concepts.

In this paper, we describe DeepMiner with respect to learning components (1)–(5). The details on the approaches for learning concept relationships will be given in the full version of the paper.

## 4 The DeepMiner Architecture

The architecture of DeepMiner is shown in Figure 2. Given a set of sources, DeepMiner starts by learning a base ontology $O$ from source interfaces (step a). Then, the ontology-growing cycle (steps b–f) is initiated. At each cycle, first the current ontology $O$ is exploited to train a label classifier $C^l$ and an instance classifier $C^i$ (step b). Next, DeepMiner poses queries to a selected source through its interface (step c) and obtains a set of data pages from the source (step d). The learned classifiers $C^l$ and $C^i$ are then employed to identify data regions in the data pages, from which DeepMiner extracts the occurrences of concepts and their instances (step e). Finally, the obtained concepts and instances are merged with $O$, resulting in a new ontology $O'$ for the next cycle (step f).

The rest of the section describes the process of learning base ontology. The details on the ontology-growing cycle will be presented in Section 5.

Consider a set of source query interfaces in a domain of interest (e.g. Figure 1.a). A query interface may be represented by a schema which contains a set of attributes. Each attribute may be associated with a label and a set of instances. The label and instances of attributes can be obtained from the interface by employing an automatic form extraction procedure [27].

Given a set of interfaces, DeepMiner learns a base ontology $O$ which consists of all *unique* concepts and their instances over the interfaces. Since similar attributes (denoting the same concept) may be associated with different labels (e.g. Make of the car may be denoted as Brand on a different interface) and different sets of instances, a key challenge is thus to identify *semantic* correspondences (i.e. mappings) of different attributes over the interfaces.

For this, DeepMiner employs a single-link *clustering* algorithm [27] to effectively identify mappings of attributes over the interfaces. Specifically, the similarity of two attributes is evaluated based on the similarity of their labels (with the TF/IDF function commonly employed in Information Retrieval) and the similarity of the data type and values of their instances. (For the attributes with no instances, DeepMiner also attempts to glean their instances from the Web.) The data type of instances is *inferred* from the values of instances via pattern matching with a set of type-recognizing regular expressions. Finally, for each produced cluster, DeepMiner adds into its base ontology a new concept which contains the information obtained from the attributes in the cluster, including labels, instances, data type, and statistics as described in Section 3.

## 5 Growing Ontology via Mining Data Pages

Denote the current ontology as $O$ which contains a set of concepts, each of which is associated with a set of labels and instances. This section describes how DeepMiner grows $O$ by mining additional concepts and instances from the data pages of a selected source. Query submission will be described in Section 6.

### 5.1 Training Label and Instance Classifiers

DeepMiner starts by training label classifier $C^l$ and instance classifier $C^i$ with training examples automatically created from $O$. $C^l$ predicts the likelihood that a given string (of words) $s$ may represent a concept in $O$, while $C^i$ predicts the likelihood that a given string $s'$ may be an instance of a concept in $O$.

*Training label classifier:* The label classifier $C^l$ is a variant of the k-nearest neighbor (kNN) classifier [13], which performs the prediction by comparing the string with the concept labels it has seen during the training phase.

Specifically, at the training phase, for each concept $c \in O$ and each of its labels $l$, a training example $(l, c)$ is created and stored with the classifier. Then, given a string $s$, $C^l$ makes predictions on the class of $s$ based on the classes of the stored examples whose similarity with $s$ is larger than $\delta$ (i.e., the nearest neighbors of $s$), by taking votes. The similarity between two strings is their TF/IDF score [18].

*Example 1.* Suppose that $O$ contains three concepts $c_1$, $c_2$, and $c_3$. Further suppose that the training examples stored with $C^l$ are ($l_1$, $c_1$), ($l_2$, $c_2$), ($l_3$, $c_3$), ($l_4$, $c_1$), ($l_5$, $c_2$), and ($l_6$, $c_3$). Suppose that $\delta = .2$.

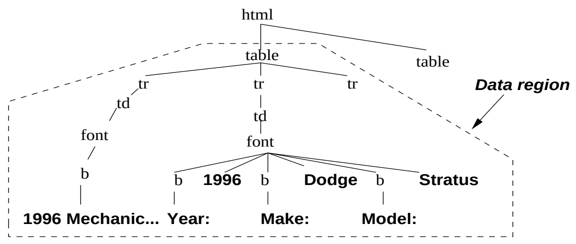


**Fig. 3: The dom tree of Figure 1.b**

Consider a string $s$ and suppose that the labels in the first five training examples (i.e., $l_1$ to $l_5$) are the ones whose similarity with $s$ is greater than .2. Since 2/5 of the nearest neighbors of $s$ are from the concept $c_1$, the confidence score for $c_1$ is .4. The predictions for other concepts are given similarly. □

*Training instance classifier:* The instance classifier $C^i$ is a naive Bayes classifier which performs the prediction based on the frequency of words which occur in the instances of the concepts. Note that $C^i$ may also be implemented as a kNN classifier, but since the number of instances of a concept is likely to be very large, the naive Bayes classifier is typically more efficient since it does not require the comparison with all the instances at the query time.

Specifically, for each instance $i$ of a concept $c$ in $O$, a training example $(i', c)$ is created, where $i'$ is a bag-of-token representation of $i$ with the stopwords in $i$ removed and the non-stop words stemmed. Then, given a string $s$, represented by $(w_1, w_2, \cdots, w_k)$, where $w_i$ are tokens. $C^i$ assigns, for each concept $c$ in $O$, a prediction score $p(c|s)$ computed as $p(c) * p(s|c)/p(s)$, where $p(s)$ is $\sum_{c' in O} p(c') * p(s|c')$. Particularly, $p(c)$ is estimated as the percentage of training examples of class $c$. $p(s|c)$ is taken to be $p(w_1|c) * p(w_2|c) * \cdots * p(w_k|c)$, based on the assumption that tokens of $s$ occur independently of each other given $c$. $p(w_i|c)$ is estimated as the ratio $n(w_i, c)/n(c)$, where $n(w_i, c)$ is the number of times token $w_i$ appears in training examples whose class is the concept $c$, and $n(c)$ is the total number of token positions in all training examples of class $c$.

## 5.2 Mining Concepts and Instances

*Identifying data regions:* A data region is a portion of a data page which contains data records generated by the source, where each record consists of a set of instances and their labels. (Note that some instances may not have labels.) To illustrate, the data region in Figure 1.b is represented by a dashed box. Note that a data page may contain more than one data regions.

To identify the data regions, DeepMiner exploits the following observations. First, the current ontology $O$ can be exploited to recognize data regions which may often contain labels and instances of existing concepts in $O$. Second, the label of a concept and its instances are often located in close proximity and spatially aligned on the data page [1]. This placement regularity can be exploited to associate the label of a concept with its instances.

Motivated by the above observations, DeepMiner starts by seeking the occurrences of concepts of $O$ and their instances in the data page. Specifically, consider a data page $p$ represented by its DOM tree. For example, Figure 3 shows the DOM tree for the data page in Figure 1.b. First, the label classifier $C^l$ is employed to predict, for each text segment $t$ (i.e. text node in the DOM tree), the concept $c$ which $t$ most likely denotes (i.e., $c$ is the concept which $C^l$ assigns the highest score $s$ with $s > .5$). Next, $t$ is further verified to see if it indeed denotes the concept $c$ by checking if the text segment located below or next to $t$ is an instance of $c$. Intuitively, these two positions are the places where instances of $c$ are likely to be located.

To determine the relative position between two text segments, DeepMiner employs an approach which directly works on the DOM tree of the data page. The approach exploits the following observations on the characteristics of data pages. First, within each data region, the sequence of text segments resulted from a *pre-order* traversal of the DOM sub-tree for the data region often corresponds to the left-right, top-down ordering of text segments when the data page is rendered via Web browsers. Second, since data pages are automatically generated, spatial alignments of text segments are often achieved via the *table* construct of HTML, rather than via explicit white space characters such as "&nbsp;" which are often found in manually generated Web pages, e.g., with some Web page authoring tool. Based on these observations, DeepMiner takes the text segment which follows $t$ in the pre-order traversal of the DOM tree to be the segment next to $t$, denoted as $t_n$. Further, if $t$ is located in the cell $[i, j]$ of a table with $M$ rows and $N$ columns, then all text segments at column $j$ and row $k$, where $i + 1 \leq k \leq N$, are taken to be the text segments *below* $t$, denoted as $t_{b_k}$'s.

Next, the instance classifier $C^i$ is employed to determine, for each text segment $t_x$ among $t_n$ and $t_{b_k}$'s, the concept in $O$ which $t_x$ is most likely to be an instance of. Suppose $t'$ has the largest confidence score among all these text segments and it is predicted to be an instance of class $c'$. Then, the text segment $t$ is determined to denote the concept $c$ only if $c' = c$. For example, State in Figure 1.b is recognized as a label for an existing concept $c$ in $O$ due to the fact that it is highly similar to some known label of $c$ and further that IL (which is a text segment next to state) is predicted to be an instance of $c$ by $C^i$.

The above procedure results in a set of label-instance pairs, each for some known concept in $O$. Data regions are then determined based on these label-instance pairs as follows. Consider such a label-instance pair, denoted as $(L, I)$. If $L$ is located in a table, then the data region induced by $(L, I)$ comprises all content of the table. Otherwise, suppose the least-common-ancestor of nodes for $L$ and $I$ in the DOM tree is $\omega$. The data region induced by $(L, I)$ is then taken to be the subtree rooted at $\omega$. The intuition is that related concepts are typically located near to each other in a data page and thus in the DOM tree of the data page as well.

*Example 2.* The DOM subtree which corresponds to the identified data region in Figure 1.b is marked with a dotted polygon in Figure 3. □

*Discovering presentation patterns:* Once data regions are identified, DeepMiner proceeds to extract concepts and their instances from the data regions. For this, DeepMiner exploits a key observation that concepts and their instances within the same data region are typically presented in a similar fashion, to give an intuitive look-and-feel impression to users. For example, in Figure 1.b, the label of concept is shown in bold font and ends with a colon, and the corresponding instance is located right next to it, shown in normal font. Motivated by this observation, DeepMiner first exploits known concepts and their instances to discover their presentation patterns, and then applies the patterns to extract other concepts and their instances from the same data region.

Specifically, a presentation pattern for a concept label $L$ and its instance $I$ in a data region $r$ is a 3-tuple: $<\alpha, \beta, \gamma>$, where $\alpha$ is the tag path to $L$ from the root of the DOM subtree for $r$, $\beta$ is the suffix of $L$ (if any), and $\gamma$ is the location of $I$ relative to $L$. These patterns are induced from the known occurrences of label-instance pairs in the region $r$ as follows. Denote the root of the DOM subtree for $r$ as $\omega$. For each label-instance pair $(L_x, I_x)$, we induce a pattern. First, $\alpha$ is taken to be the sequence of HTML tags from $\omega$ to the text segment node for $L_x$, ignoring all hyperlink tags (i.e., 'a'). Second, if the text segment for $L_x$ ends with symbols such as ':', '-' and '/', these symbols constitute $\beta$. Third, $\gamma$ has two possible values: *next* and *below*, depending on how $I_x$ is located, relative to $L_x$.

*Example 3.* $\alpha$ for the data region in Figure 3 is (table, tr, td, font, b), $\beta$ is the suffix ':', and the value of $\gamma$ is *next*. □

*Extracting concept labels and instances:* This step employs the learned patterns to extract concept labels and their instances from the data region $r$. In particular, $\alpha$ and $\beta$ of a pattern are first applied to identify labels of other concepts in the region and then the instances of the identified concepts are extracted in the location relative to the labels as indicated by the $\gamma$ part of the pattern.

*Example 4.* The learned pattern from Figure 3 will extract concept-instances pairs from Figure 1.b such as: (Year, {1996}), (Make, {Dodge}), and (Posted, {January 04, 2005}). □

### 5.3 Merging with the Current Ontology

This step merges the label-instances pairs mined from the data pages into the current ontology $O$. Specifically, for each label-instances pair $e = (L, I)$, if $e$ belongs to an existing concept $c$, then $L$ and $I$ are added to the list of labels and instances for $c$, respectively. The corresponding statistics for $c$ are also updated accordingly. Otherwise, a new concept will be created with $L$ as a label and $I$ as a set of instances.

## 6 Empirical Evaluation

We have conducted preliminary experiments to evaluate DeepMiner. The experiments use an e-commerce data set which contains sources over automobile, book and job domains, with 20 sources in each domain [2]. Each source has a query interface represented by a set of attributes. The average number of attributes for the interfaces in the auto, book and job domains is 5.1, 5.4, and 4.6, respectively.

| **Domains** | **Base Ontology** | | **Data Regions** | | **Concept-Instances** | |
|---|---|---|---|---|---|---|
| | Prec. | Rec. | Prec. | Rec. | Prec. | Rec. |
| Auto | 100 | 98.9 | 6/7 | 6/6 | 41/43 | 41/41 |
| Book | 100 | 90.4 | 8/8 | 8/8 | 41/41 | 41/43 |
| Job | 94.6 | 91.2 | 5/5 | 5/5 | 22/22 | 22/23 |

**Table 1: The performance of DeepMiner**

First, we evaluated the performance of DeepMiner on discovering unique concepts over source interfaces. The performance is measured by two metrics: *precision*, which is the percentage of correct mappings of attributes among all the mappings identified by the system, and *recall*, which is the percentage of correct mappings among all mappings given by domain experts. In these experiments, the clustering threshold is set to .25, uniformly over all domains. Results are shown in columns 2–3 in Table 1.

It can be observed that attribute mappings are identified with high precision over all domains, with a prefect precision in the auto and book domains and around 95% for the job domain. Furthermore, good recalls are also achieved, ranging from 90.4% in the book domain to 98.9% in the auto domain. Detailed analysis indicates the challenge of matching some attributes in the book domain, e.g., DeepMiner failed to match attributes section and category since their instances have very little in common. A possible remedy is to utilize the instances obtained from data pages to help identify their mapping.

To isolate the effects of different components, we manually examined the mapping results and corrected mismatches. This process takes only a couple of minutes, since there are very few errors in each domain.

Next, we evaluated the performance of DeepMiner on identifying data regions. For this, we randomly select five sources for each domain. For each source, query submission is made by automatically formulating a query string which consists of form element names and values, and posing the query to the source. If an attribute does not have instances in its interface, the instances of its similar attributes (available from the base ontology) are used instead. This probing process is repeated until at least one valid data page is returned from the source, judged based on several heuristics as employed in [16]. For example, pages which contain phrases such as “no results” and “no matches” are regarded as invalid pages.

For all data pages retrieved in each domain, we first manually identified the number of data regions in the pages, and use it as the gold standard. DeepMiner's performance is then measured by the number of data regions it *correctly* identified, over all data regions it identified (i.e. precision), and over all the expected data regions as given by the gold standard (i.e. recall). Results are shown in columns 4–5 of Table 1. It can observed that DeepMiner is very accurate in identifying data regions: only one is incorrectly identified in the auto domain.

Finally, we evaluated DeepMiner's performance on discovering concepts and their instances from data pages. This was done by first manually determining the number of concept labels and their instances in all data pages, and then comparing the concept-instances pairs discovered by DeepMiner with this gold standard. Results are shown in the last two columns of Table 1.

We observe that DeepMiner achieves very high accuracy consistently over different domains. We looked at the data pages it made mistakes and examined the reasons. In particular, we note that there is a concept with label job description: in www.aftercollege.com, but its instance is located in the same text segment as the label, although the label does contain a delimiter ':'. It would be interesting to extend DeepMiner to handle such cases. DeepMiner also made some errors in Amazon.com. For example, currently it is difficult for DeepMiner to recognize that only Prentice Hall in Prentice Hall (feb 8, 2008) is an instance of publisher. We are currently developing a solution which exploits the existing ontology to perform *segmentation* on the text segments.

## 7 Discussions & Future Work

We now address the limitations of the current DeepMiner system. The first issue to address is to make the learning of presentation patterns more robust, e.g., handling possible non-table constructs. Currently, the relative positions of attributes and their values are obtained by analyzing their appearance in the DOM trees. An alternative is to render the data page with a Web browser and obtain the spatial relationships (e.g., pixel distances and alignments) of attributes and values from the rendered page. But this approach has a potential disadvantage of being time-consuming.

Second, we plan to perform additional experiments with the system and further examine its performance. Preliminary results indicated that data pages are typically rich in attributes and values, and that a dozen of data pages per Web site are often sufficient for learning a sizable ontology. As such, we expect our approach to be scalable to a large number of Web sites.

Finally, it would be interesting to combine our approach with the approaches of learning concepts and instances from the Web services already existing in the B2B domain (e.g. [17]). Further, the ontology learned with our approach can be utilized to train concept and instance classifiers, which can then be employed to markup the Web services by the approaches such as [15].

## 8 Conclusions

We have described the DeepMiner system for learning domain ontology from the source Web sites. The learned ontology can then be exploited to mark up Web services. Its key novelties are (1) incremental learning of concepts and instances; (2) effective handling of the heterogeneities among autonomous sources; and (3) a machine learning framework which exploits existing ontology in the process of learning new concepts and instances. Preliminary results indicated that it discovers concepts and instances with high accuracy.

We are currently investigating several directions to extend DeepMiner: (a) employ the learned ontology to segment complex text segments and recognize instances in the segments; (b) utilize the instances gleaned from data pages to assist in matching interface attributes; and (c) combine DeepMiner with the approach of learning domain ontology from texts.

**Acknowledgment:** This research is supported in part by the following grants from NSF: IIS-0414981 and IIS-0414939.

## References

1. L. Arlotta, V. Crescenzi, G. Mecca, and P. Merialdo. Automatic annotation of data extracted from large Web sites. In *WebDB*, 2003.
2. http://metaquerier.cs.uiuc.edu/repository/.
3. B. Benatallah, M. Hacid, A. Leger, C. Rey, and F. Toumani. On automating web services discovery. *VLDB Journal*, 14(1), 2005.
4. F. Casati and M. Shan. Models and languages for describing and discovering e-services. In *Tutorial, SIGMOD*, 2001.
5. The OWL-S Services Coalition. OWL-S: Semantic Markup for Web Services. http://www.w3.org/Submission/OWL-S/.
6. V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large Web sites. In *Proc. of VLDB*, 2001.
7. G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for daml web services: Annotation and matchmaking. In *ISWC*, 2003.
8. M. Dumas, J. O'Sullivan, M. Hervizadeh, D. Edmond, and A. Hofstede. Towards a semantic framework for service description. In *DS-9*, 2001.
9. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce: Research and Applications*, 1, 2002.
10. A. Heß and N. Kushmerick. Machine learning for annotating semantic web services. In *AAAI Spring Symposium on Semantic Web Services*, 2004.
11. F. Leymann. WSFL (Web Service Flow Language), 2001.
12. B. Li, W. Tsai, and L. Zhang. Building e-commerce systems using semantic application framework. *Int. J. Web Eng. Technol.*, 1(3), 2004.
13. T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
14. M. Paolucci and K. Sycara. Semantic web services: Current status and future directions. In *ICWS*, 2004.
15. A. Patil, S. Oundhakar, A. Sheth, and K. Verma. METEOR-S: Web service annotation framework. In *WWW*, 2004.
16. S. Raghavan and H. Garcia-Molina. Crawling the hidden Web. In *VLDB*, 2001.
17. M. Sabou, C. Wroe, C. Goble, and G. Mishne. Learning domain ontologies for web service descriptions: an experiment in bioinformatics. In *WWW*, 2005.
18. G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McCraw-Hill, New York, 1983.
19. K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding semantics to web services standards. In *ICWS*, 2003.
20. SOAP. http://www.w3.org/TR/soap/.
21. http://uddi.microsoft.com/.
22. UDDI. http://www.uddi.org/.
23. D. VanderMeer, A. Datta, et al. FUSION: A system allowing dynamic Web service composition and automatic execution. In *CEC*, 2003.
24. L. Vasiliu, M. Zaremba, et al. Web-service semantic enabled implementation of machine vs. machine business negotiation. In *ICWS*, 2004.
25. J. Wang and F. Lochovsky. Data extraction and label assignment for Web databases. In *WWW*, 2003.
26. WSDL. http://www.w3.org/TR/wsdl/.
27. W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the Deep Web. In *SIGMOD*, 2004.