Extraction Techniques for Mining Services from Web Sources

Hasan Davulcu, Saikat Mukherjee, I.V. Ramakrishnan

Dept. of Computer Science
SUNY Stony Brook
Stony Brook, NY 11794, USA
{davulcu,saikat,ram}@cs.sunysb.edu

Abstract

The Web has established itself as the dominant medium for doing electronic commerce. Consequently the number of service providers, both large and small, advertising their services on the web continues to proliferate. Such web presences can range from a simple reference to the service provider in a referral page containing many such references to a full-blown web site of the service provider. Creating queriable service directories by mining such web presences will add impetus to e-commerce activities on the web. In this paper we describe new extraction algorithms for mining service directories from web pages. Services are characterized by an ontology consisting of a taxonomy of service concepts, their associated attributes (such as names and addresses), type descriptions for the attributes and attribute identifier functions to locate occurrences of a service's attributes in a web page.

Two central extraction issues that arise in the mining of service directories from web documents are:

(i) How to group the attributes related to a service provider entity especially when there are multiple entities in a page as is the case with referral pages?

(ii) How to increase the recall by extracting those attribute occurrences that the identifier function fails to locate especially when the domain of the attribute is not completely known or is unbounded.

To address the first issue we develop a novel propagation technique for identifying and accumulating all of the attributes related to a service entity in a web page. For increasing the recall we develop a novel unsupervised learning technique that infers the "context" of the attributes in the page and uses the inferred contexts to locate additional attribute occurrences that the identifier functions fail to detect.

In contrast to page-specific wrapper techniques the distinguishing aspect of our ontology-directed technique is that it does not rely on any relationship between markup tags and attribute occurrences in the page. Hence our technique is scalable, i.e. it can handle any page relevant to the intended service domains. Our technique is also versatile since it is easy to tune the ontology to cover different types of service domains.

We provide experimental results of the effectiveness and versatility of our extraction techniques by mining a database of veterinarian service providers and real estate agencies from web sources.

keywords: web mining, information extraction, service directories

Contact Author: Saikat Mukherjee
E-mail: saikat@cs.sunysb.edu

1 Introduction

The web has established itself as the dominant medium for doing electronic commerce. Realizing that its global reach provides significant market and business opportunities, service providers, both large and small are advertising their services on the web. A number of them operate their own web sites promoting their services at length while others are merely listed in a referral site. Figure 1(a) below is an example of a home page fragment of a service provider and a referral page fragment listing several service providers is shown in Figure 2(a).

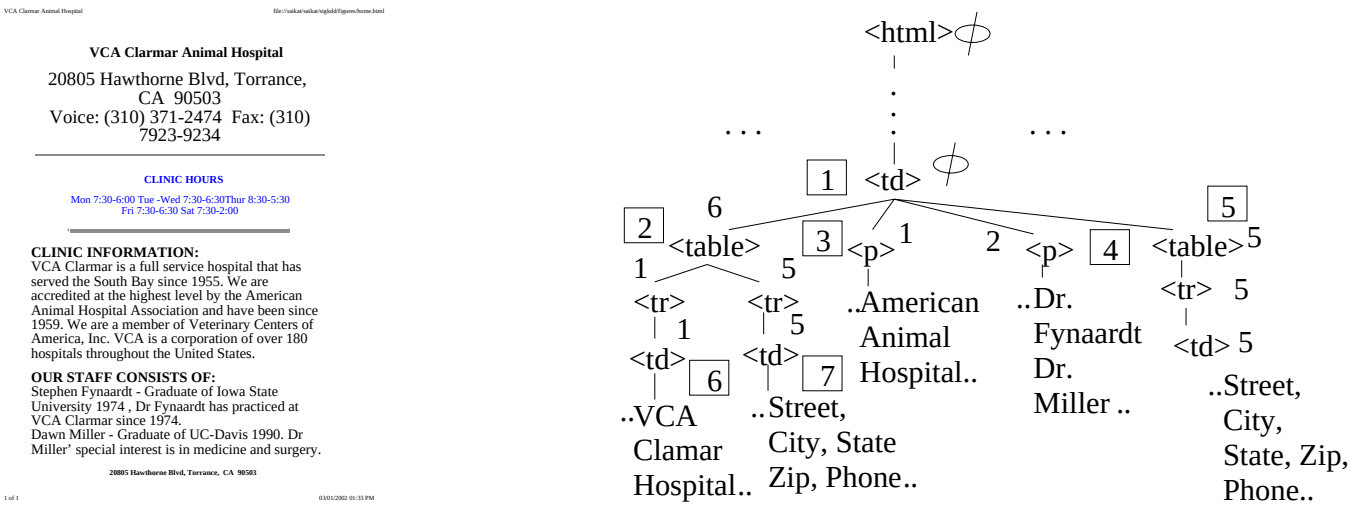


Figure 1: (a) A veterinarian service provider home page (b) Fragment of the DOM tree of the page

Motivation: Aggregating all of the providers into a queriable service directory makes it easy for customers to locate the best suited servicies for their needs. In fact such service directories become important in the context of emerging web service applications e.g. [6, 5, 4, 7]. Such a perceived need can be addressed by electronic versions of traditional yellow pages. Such directories may not reflect the richness of content that a service provider's web page will usually possess, such as emails, specialties, etc. So directory builders, such as Verizon's $www.superyellowpages.com^{TM}$, provide forms at their web sites for service providers to list their services in the directory. But the information that gets into it is pre-determined apriori by the forms and changes to the form require the service providers themselves to update their information. A more attractive solution is to create the service directory by mining the web for service providers. Such a solution has several merits. Firstly, the information content will be rich. Secondly, unlike form-based approaches it does not need any explicit participation by the service provider and hence is scalable. Thirdly, since the process is automated and repeatable the content can always be kept current. Finally, the same process can be readily adapted to different domains. Hence developing the tools and techniques for mining service directories from web sources is a technically interesting problem with a practical dimension and constitutes the topic of this paper.

Problem Formulation: We characterize services by an ontology consisting of a taxonomy of service concepts, their associated attributes (such as names and addresses) and type descriptions for the attributes. In addition the ontology also associates an attribute identifier function with each attribute. Applying the function to a web page will locate all the occurrences of the attribute in that page. Figure 3 illustrates a services ontology.

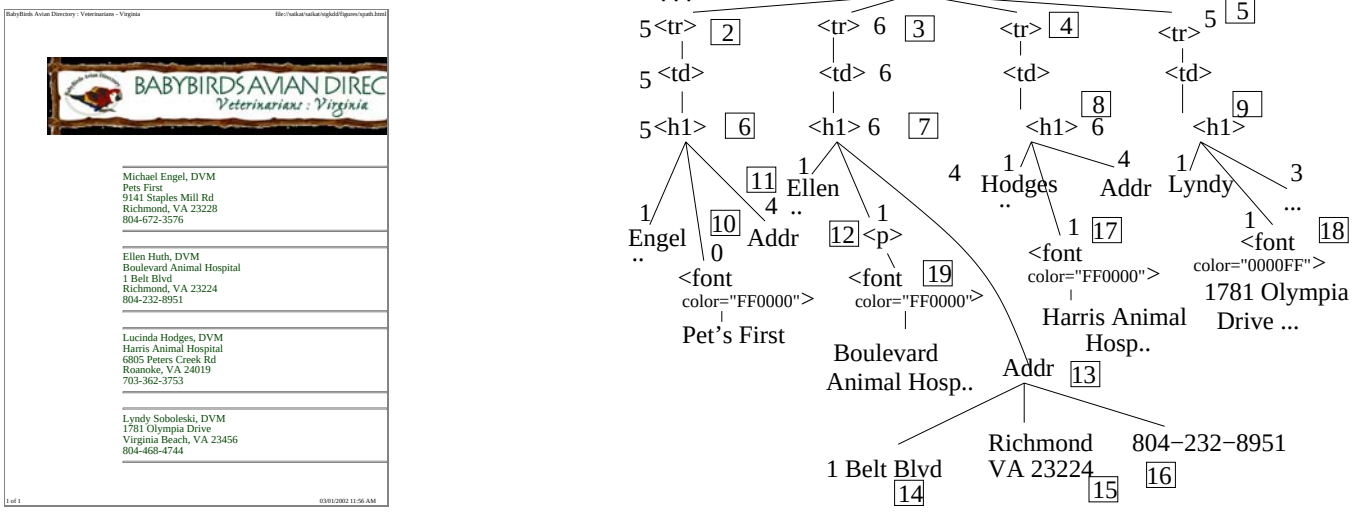
There are two aspects to mining services from web sources, namely, an acquisition and classification phase followed by an extraction phase. In the former phase web documents relevant to the intended domain of services

are identified while in the latter phase a service provider's attributes are extracted from these documents. Our focus in this paper is on the extraction phase.

Each web page is parsed into a DOM (Document Object Model [1]) tree and the identifier functions, specified in the ontology for locating occurrences of the attributes in the page, are applied. The problem now is this: How to group all of the attributes corresponding to each service provider? This can pose difficulties especially in the presence of multiple service providers in a page (as in Figure 2(a)). For example, in the figure we will have to correctly associate the attribute occurrences "Michael Engel, DVM" and "Ellen Huth, DVM" with the 1st and 2nd service provider respectively. Observe that in a service provider referral or home page the attribute occurrences belonging to distinct service entities are present below distinct subtrees in the document parse tree. Based on this observation, we develop a novel *scoring* technique to aggregate attribute occurrences related to the same service entity and a *conflict resolution* technique to separate distinct service entities.

The identifier function may not be "complete" in the sense that it cannot always identify all the attributes in a page. For example in Figure 2(a) an identifier function that depends on finding the keyword "hospital" in a provider's name would have identified "Boulevard Animal Hospital" and "Harris Animal Hospital" and missed "Pet's First". This motivates the second interesting question: How to increase recall by extracting those attribute occurrences that the identifier function fails to locate especially when the domain of the attribute is not completely known or is unbounded.

In this paper we propose a solution to both these problems.



(a) (b)

Figure 2: (a) A veterinarian service provider home page (b) Fragment of the DOM tree of the page

Our Results:

1. We develop a novel ontology-directed propagation technique for identifying and accumulating all of the attributes related to each service entity in a web page. By using a concept of *scoring* and *conflict resolution* to prevent erroneous associations, our algorithm groups the attributes related to each service provider in a web document. In contrast to page-specific wrapper techniques the distinguishing aspect of our technique is that it does not rely on any relationship between markup tags and attribute occurrences in the page. Hence our technique is scalable, i.e. it can handle any page relevant to the intended service domains. Our

technique is also versatile since, as we demonstrate in the case study, it is easy to engineer the ontology to cover different types of service domains.

2. For increasing the recall we develop a novel unsupervised learning technique that infers the "context" of the attributes in the page and uses the inferred contexts to locate additional attribute occurrences that the identifier functions fail to detect (such as "Pet's First" in Figure 2(a).).

3. We provide experimental results of the effectiveness and versatility of our extraction techniques by mining a database of veterinarian service providers and real estate agencies from web sources.

Organization: The paper is organized as follows. Section 2 contains the technical details of our extraction algorithm. The implementation of a system for mining web services using our extraction algorithm appears in Section 3. In Section 4 we present case studies of using the mining system for building directories for two services from web sources, namely, veterinarian and real estate. Section 5 discusses related work spanning information extraction through the use of wrappers and natural language processing, machine learning techniques, and semi-structured data query languages. A critical discussion of our approach and concluding remarks appear in Section 6.

2 Ontology Directed Mining

The concept of an ontology is central to the formalization of a service directory. A concept in an ontology is a type of service, e.g. *Veterinarian*. The ontology associates attributes with service providers (e.g. service provider's name, address, phone, email, vet's name etc). Some of them may be shared across different service domains (e.g. address, phone, email, etc). We will denote a member of a concept as an *entity*. e.g. the service provider "VCA Clarmar Animal Hospital" is an entity that is a member of the concept *Veterinarian*. We associate attributes with an entity. The attributes of an entity can be single and multi-valued. A single-valued attribute means that the entity can have at most one value whereas it can have several values for multi-valued attributes. e.g. the address "20805 Hawthorne Blvd. Torrance, CA 90503" of the entity named "VCA Clarmar Animal Hospital" is single-valued whereas veterinarian name is multi-valued that can assume several values such as "Dr. Fynaardt" and "Dr. Miller" who are the vets in VCA Clarmar Animal Hospital. Formally:

Definition 1 (Ontology) *: A service ontology O is a 10-tuple*
$O =< C, T, D, A_m, A_n, A, \tau, Val_s, Val_m, Attr\_identifier >$
where:

- *C is a set of service concepts.*
- *$T \subset C \times C$ is the taxonomy and denotes the IS-A relationship between concepts.*
- *D is the set of domain types. A domain type can be the set of all strings, set of all integers, etc.*
- *A_s is a set of single-valued attribute names while A_m is a set of multi-valued attribute names.*
- *$A : C \rightarrow 2^{A_m \cup A_s}$ is a function that associates a set of attributes with a concept.*
- *$\tau : A_m \cup A_s \rightarrow D$ is a function that associates a domain type to every attribute.*
- *$val_s : A_s \rightarrow (C \rightarrow \tau(A_s))$ is a function denoting that the attributes in A_s are single-valued.*
- *$val_m : A_m \rightarrow (C \rightarrow 2^{\tau(A_m)}$ is a function denoting that the attributes in A_m can take multiple values.*

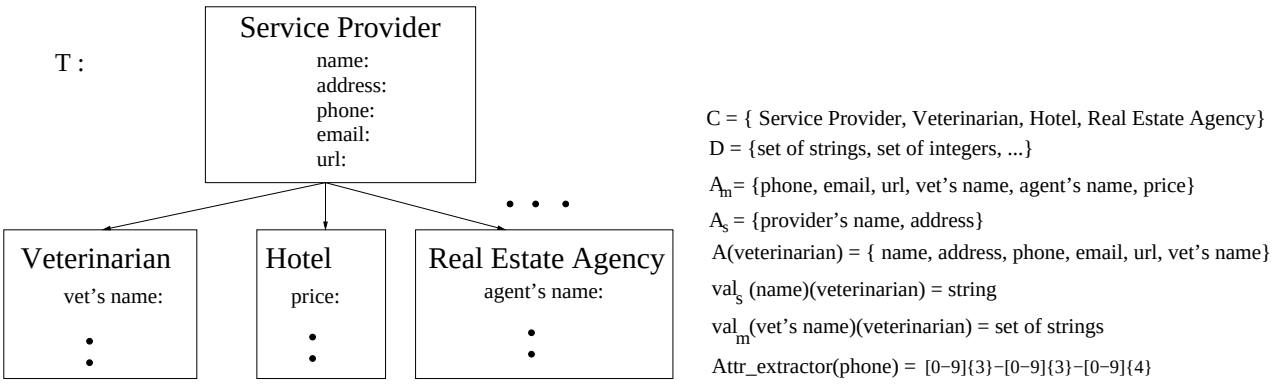


Figure 3: A fragment of a Service Directory Ontology

- $Attr\_identifier : Attr \rightarrow (string \rightarrow 2^{\tau(Attr)}), Attr \in (A_m \cup A_n)$.

Figure 3 shows a fragment of a service ontology with Veterinarians, Real Estate Agency, Hotels as the service concepts. The $Attr\_identifier$ function specifies how to locate the attribute occurrences in a web page. It will include regular expressions (re's), rules, etc. For example we can identify phone numbers by a re [24]. So $Attr\_identifier(phone)$ is the re: "[0-9]{3}-[0-9]{3}-[0-9]{4}".

The problem addressed in this paper is: *how to populate the service ontology with entities that are members of concepts in O, by mining web pages.* All these pages are assumed to be HTML pages.

Each entity is *uniquely* identified by a set of single-valued attributes. We call any such set as a *key*. e.g. for service providers two possible keys are $\{street, city\}$ and $\{street, zip\}$. The attributes in a home page (Figure 1(a)) are associated with a single entity whereas a referral page contains several entities (Figure 2(b)).

Let $\Cup$ denote the bag union of a set of elements. In such a union elements can repeat.

Definition 2 (Consistent Bag) *: Let S be a bag consisting of pairs of the form $< A_i, X_i >$ where A_i is an attribute and X_i is a set of values. S is consistent if and only if, $\forall < A_i, X_i >, < A_j, X_j >$ if $A_i, A_j \in A_s$ then $A_i \neq A_j$.*

For example, the bag (<provider name, {VCA Clamar Animal Hospital}>, <street, {20805 Hawthorne Blvd}>, <city, {Torrance}>) is consistent whereas (<provider name, {VCA Clamar Hospital}>, <street, {20805 Hawthorne Blvd}>, <city, {Torrance}, {provider name, {American Animal Hospital Association}>) is not consistent since the single-valued attribute "provider name" occurs twice.

Let T be the DOM tree of a page. e.g. Figures 1(b) and 2(b) are fragments of the DOM trees for the pages in Figures 1(a) and 2(a) respectively. The leaf nodes in T are text strings. $Parent(n)$ denotes the parent of node n and $children(n)$ denotes all its children. We are interested in identifying subtrees in T in which no single-valued attribute occurs more than once. We use the notion of a *mark* for doing so. We will use c to refer to a particular concept in C.

Definition 3 (Mark of a Node) *: Let n be a node in T.*

$$mark_c(n) = \begin{cases} \textit{if } n \textit{ is a leaf} = \begin{cases} \{< A_i, Attr\_identifier(A_i)(\sigma) > | & A_i \in A(c) \\ & \wedge\ \sigma \textit{ is the text string associated with n} \\ & \wedge\ A_i \in A_s \rightarrow |Attr\_identifier(A_i)(\sigma)| \leq 1\} \\ \phi, & \textit{else} \end{cases} \\ \textit{if } n \textit{ is not a leaf} = \begin{cases} \bigcup_{m \in children(n)} mark_c(m), & \biguplus_{m \in children(n)} mark_c(m) \textit{ is consistent} \\ \phi, & \biguplus_{m \in children(n)} mark_c(m) \textit{ is not consistent} \end{cases} \end{cases}$$

Whenever *mark(n)* is ϕ it means that there exists more than one occurrence of a single valued attribute in its subtree. The definition also suggests how to propagate marks. Specifically, the subtrees rooted at a node can be merged as long as no single-valued attribute occurs in more than one subtree.

For notational simplicity, we use $mark(n)$ in place of $mark_c(n)$ whenever c is known from the context. To associate attributes with entities we use the notion of a maximally marked node:

In Figure 2(b), the node numbers appear within a square. mark(12) is {<provider name, {Boulevard Animal Hospital}>}, mark(11) is {<vet's name, {Ellen Huth, DVM.}>} and the marks of nodes 14, 15 and 16 are {<street, {1 Belt Blvd.}>}, {<city, {Richmond}>, <state, {VA}>, <zip, {23224}>} and {<phone, {804-232-8951}>} respectively. When the mark is propagated up the tree, mark(7) becomes {<vet's name, {Ellen Huth, DVM.}, <provider name, {Boulevard Animal Hospital}>, <street, {1 Belt Blvd.}>, <city, {Richmond}>, <state, {VA}>, <zip, {23224}>, <phone, {804-232-8951}>}. Similarly, mark(8) is {<vet's name, {Lucinda Hodges, DVM}, <provider name, {Harris Animal Hospital}>, <street, {6805 Peters Creek Rd}>, <city, {Roanoke}>, <state, {VA}>, <zip, {24019}>, <phone, {703-362-3753}>}. However, mark(1) is ϕ since the single-valued attribute – provider's name – occurs in the subtrees rooted at nodes 3 and 4.

Definition 4 (Maximally marked node) *: Let n be an internal node.*

$$maximal(n) = \begin{cases} true, & n \textit{ is not leaf} \wedge mark(n) \neq \phi \wedge mark(parent(n)) = \phi \\ false, & \textit{otherwise} \end{cases}$$

In Figure 2(b), nodes 2,3,4 and 5 are maximally marked nodes since they are all marked as $\neq \phi$ while their parent, node 1, is marked ϕ. Intuitively the leafs of a maximally marked node are the attributes of a single entity. e.g. the attributes of the entities named "Boulevard Animal Hospital" and "Harris Animal Hospital" appear on the leafs of the subtrees rooted at nodes 3 and 4 respectively.

2.1 Extraction Algorithm

The algorithm for extracting entities and their attribute values from a page is now described. Let $\sigma(n)$ denote the concatenation of the text strings associated with the leaf nodes of the subtree rooted at n, $Attr$ be the set of attributes of the concept c, $\{k_1, ..., k_n\}$ be the attributes that constitute the key of c; and $R(a_1, ..., a_n)$, denote the tuple of attributes associated with an entity. We will extract one tuple from a home page and several such tuples from a referral page.

We use $score(n)$ to denote $|mark(n)|$. The uncircled numbers in Figures 1(b), 2(b) and 4(c) represent the scores of the nodes. e.g. in Figure 2(b) score(2) is 5 and score(3) is 6.

Algorithm Extract (T, $Attr$)
begin
1. **for**all nodes $n \in T$ **do**
2. $mark(n)$
3. **end**
4. Let Γ = { maximally marked nodes in T } $\cup$ {all leaf nodes marked ϕ }
5. **if** $\exists m_i, m_j \in \Gamma \wedge \{Attr\_identifier(k_1)(\sigma(m_i)), ..., Attr\_identifier(k_n)(\sigma(m_i))\} \neq \{Attr\_identifier(k_1)(\sigma(m_j)), ..., Attr\_identifier(k_n)(\sigma(m_j))\}$ **then**
6. T is a referral page
7. **else**
8. T is a home page
9. **endif**
10. **if** T is a home page **then**
11. R = Extract_Home_Page($Attr$, Γ)
12. **else**if T is a referral page **then**
13. $\{R_1, ..., R_n\}$ = Extract_Referral_Page($Attr$, Γ)
14. **end**
end

Algorithm Extract takes as input the tree of the page and the set of attributes names of the concept c. It outputs either a single tuple containing the values of the attributes if it is a home page or a set of tuples if it is a referral page. In lines 1-3, every node in the tree is marked by the attributes present in the subtree rooted at the node. In line 4, the set of maximally marked nodes in the tree is determined. Line 5 tests for a home page or a referral page. Specifically the nodes in Γ of type home page cannot have different key values indicating distinct entities, otherwise the page is a referral page. Depending on the type of page the appropriate algorithm is invoked (lines 10-14). The algorithm for extraction from home pages is described below.

Algorithm Extract_Home_Page ($Attr$, Γ)
begin
1. pick the node n in Γ with the maximum *score*
2. **for**all $a_i \in Attr \wedge a_i \in A_s$ **do**
3. $R[a_i] = Attr\_identifier(a_i)(\sigma(n))$
4. **end**
5. **for**all $a_i \in Attr \wedge a_i \in A_m$ **do**
6. $R[a_i] = \bigcup_{m_i \in \Gamma} Attr\_identifier(a_i)(\sigma(m_i))$
7. **end**
8. return R
end

Extract_Home_Page takes as input the set of attribute names whose values are to be extracted and the set of maximally marked nodes in the document tree. In line 1, the maximally marked node with the highest score is determined. The values of any single-valued attribute are obtained from this node. This is done in lines 2-4. Values of multi-valued attributes are obtained from all the maximally marked nodes in the tree, which is done in lines 5-7. The extracted tuple containing values of all the attributes is returned in line 8. The intuition behind this algorithm is that the maximally marked node contains the key attributes associated with the service entity and any node in the document tree might contain occurrences of the multi-valued attributes. The complexity of the algorithm is linear in the number of nodes in the document tree.

For referral pages we have to extract the attributes of several entities. The main problem here is associating the extracted attributes with their corresponding entities. We use the notion of a *conflicting* set that will be used in making such an association.

Let Γ be as defined in Algorithm Extract. Observe that Γ is an ordered set of nodes. Let $< m_1, m_2, \ldots, m_q >$ denote the nodes in this ordered sequence. We say that Γ is *conflict-free* whenever $\exists i$, m_i, $m_{i+1} \in \Gamma$ such that mark(m_i) $\Cup$ mark(m_{i+1}) is consistent. Γ is not conflict-free if all pairs of consecutive nodes are mutually inconsistent. Observe that whenever Γ is not conflict-free then any maximally marked node represents a single entity. All we need to do is simply pick the attributes in it and create the tuple for that entity (line 7 in algorithm Extract_Referral_Page). If this is not the case then attributes of an entity may be spread across neighboring nodes. In that case we will have to detect the boundaries separating each entity (line 12). In addition even if Γ is conflict-free the leaf nodes in it will have conflicts and we will have to detect boundaries separating the attributes of entities in the text string at the leaf node (line 4 in the algorithm Extract_ Referral_ Page).

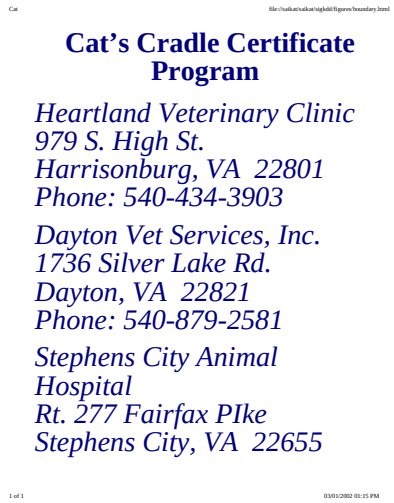
In Figure 2(b) Γ is the node set $\{2,3,4,5\}$ and it is not conflict-free. Hence each of them represents an independent entity. In Figure 4(b) Γ is $\{1,2,3,4,5,6\}$ and it is conflict-free. Observe that the attributes in node 1 and node 2 are consistent– one is the service provider's name and the other is a street. Hence they are conflict free. To get all the attributes of the entity we will have to combine them. In order to identify different entities and their attributes in conflict free sequences of nodes, we have to do boundary detection. This procdure is discussed next.

Algorithm Extract_Referral_Page ($Attr$, Γ)
begin
1. **if** Γ is not conflict-free **then**
2. **for**all $m_i \in \Gamma$ **do**
3. **if** m_i is a leaf $\wedge mark(m_i) = \phi$ **then**
4. $\{R_1, ..., R_n\}$ = Boundary_Detection($Attr$, m_i)
5. **else**
6. **for**all $a_j \in Attr$ **do**
7. $R_i[a_j] = Attr\_identifier(a_i)(\sigma(m_i))$
8. **end**
9. **end**
10. **end**
11. **else**
12. $\{R_1, ..., R_n\}$ = Boundary_Detection($Attr$, Γ)
13. **end**
14. return $\{R_1, ..., R_n\}$
end

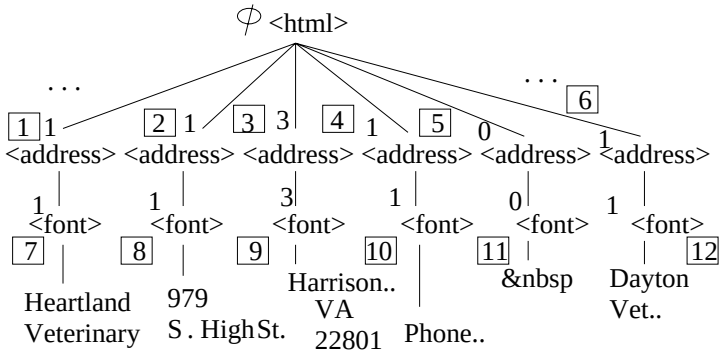
Boundary Detection: We only sketch the ideas underlying Algorithm Boundary_ Detection. In the absence of well-defined boundaries between entities, our problem is to separate the sequence of attribute occurrences into *partitions*. A partition is a sequence of attribute occurrences such that any single-valued attribute occurs at most once in it whereas multi-valued attributes can have many occurrences, provided all such occurrences are consecutive. Observe that the occurences of attributes in every entity follow a consistent order. For example, in fig 4, all the three entities follow the global order $(Hospitalname) \prec (Address) \prec (Phone)$. One can formulate the boundary detection problem as finding the minimum number of partitions such that attribute occurrences in every partition follows a globally consistent ordering. Discovering the global order is a hard problem in general and no efficient algorithms exist for solving it. We propose a heuristic for separating these attribute occurrences into *maximal partitions*. In a maximal partition adding an attribute will violate the above definition of a partition. Our algorithm for boundary_detection greedily discovers maximal partitions. We pick attributes one by one from the sequence. We check to see if it can be added to the current partition. If we cannot then the current partition is maximal and we start a new partition with this element.

Extract_Referral_Page on the tree in Figure 4(b) will yield a Γ consisting of all the children of the root node that have a non-zero score. The sequence of attributes generated from Γ is <"Heartland Veterinary", "979

S. High St.", "Harrison" "VA" "22801", "540-434-3903", "Dayton Veterinary Services Inc", "1736 Silver Lake Rd", ..>. The greedy heuristic will create the first partition consisting of the attributes: "Heartland Veterinary", "979 S. High St.", "Harrison" "VA" "22801", "540-434-3903". It will start the second partition beginning with the attribute "Dayton Veterinary Services Inc". It cannot add it to the 1st partition since doing so will result in assigning two distinct values to a single valued attribute, namely service provider's name, for the same entity.



(a)



(b)

Figure 4: (a) A referral page fragment of veterinarian service providers (b) Fragment of the DOM tree of the page

However the algorithm does have limitations and can generate incorrect partitions. e.g. the sequence $\{< n, a, p, e, a, p, e, n, a, p >\}$ (where *n is provider name,* a is address, *p is phone and* e is email, n and a are single-valued, p and e are multi-valued) will be partitioned into $< n, a, p, e >$, $< a, p, e, n >$ and $< a, p >$. The correct partition is $< n, a, p, e >$, $< a, p, e >$ and $< n, a, p >$. This can be rectified by boundary detection algorithms which take into account the order of the sequence of attribute occurrences.

2.2 Learning to Increase Recall

We define recall of an attribute as the fraction of the number of occurrences correctly extracted to the number of occurrences actually present for that attribute in a web page.

It is difficult to specify robust identifier functions for attributes with unbounded domains (names of doctors, hospitals, hotels, etc.) or when they are misspelt. For example, *Lakes Aminal Clinc, hrs., (1222) 223-3456* instead of *Lakes Animal Clinic, hours, (122) 223-3456.* How do we identify them in the document? Recall that the attributes of service entities in a referral page exhibit "regularity". For example, the name of the hospital may always be in the first column and the name of the doctor in the second column of a table for a particular referral page. We describe an unsupervised learning technique that exploits this regularity in a referral page to identify attributes missed by the identifier functions. We sketch only the main ideas.

Suppose that some occurrences of an attribute, e.g. hospital name, have been identified in the trees rooted at maximally marked nodes. e.g. nodes 3 and 4 in figure 2(b). The indexed paths (in XPath notation [2]) from nodes 3 and 4 nodes to the leaves containing the marked instances of *hospital name* are $\backslash tr[2]\backslash td[1]\backslash h1\backslash p\backslash font[1]\backslash text()$ and $tr[3]\backslash td[1]\backslash h1\backslash font[1]\backslash text()$ respectively. These paths will serve as the positive examples for our learning algorithm. Our algorithm proceeds as follows: First we compute a generalized path expression from the longest common subsequence (lcs) of these path strings. In finding the lcs we ignore the indices of the tags in the path strings and turn the paths into sequence of tags. Since the tags in the lcs appears in each of these strings there

exists an association from every tag in the lcs with a corresponding tag in every other path. e.g. for the above example the lcs would be $tr, td, h1, font, text$. Now we learn a generalized path expression Ω from the lcs as follows: First we will transform the lcs into lcs'. For every tag in the lcs, if the tag has an index and the indices of all the corresponding tags in the path strings are the same then retain this tag along with its index in lcs' otherwise retain only the tag without its index. e.g. for the above lcs, the lcs' would be $tr, td[1], h1, font[1], text$. Now we construct Ω, the generalized path expression for *hospital name* from lcs'. Let P denote the set of path strings from which the lcs was constructed. Let $\alpha_1, \alpha_2, \ldots, \alpha_k$ be the elements in lcs'. Suppose γ_r and γ_s are the elements of a path string in P that correspond to α_i and α_{i+1} respectively. If γ_r and γ_s are not consecutive in any path string then add '$\backslash\backslash$' in between α_i and α_{i+1} in Ω. The '$\backslash\backslash$' operator means that after you see α_i search for α_{i+1} in the subtree rooted at α_i. Otherwise, add a '$\backslash$' operator in between α_i and α_{i+1} in Ω. e.g. for the above example $\Omega = \backslash tr\backslash td[1]\backslash h1\backslash\backslash font[1]\backslash text()$.

The paths that will be matching instances of Ω from maximal nodes will include all the path strings in P as well as some other paths. The missing attributes may occur on the leafs of these other paths e.g. *node 10 of Figure 2(b)*. But it may also include certain unwanted attributes. e.g. *node 18 of Figure 2(b) which is already marked as an address attribute*. The paths to such attributes will form the negative examples N to our learning algorithm. Next, we specialize Ω to Ω_s by identifying and adding an HTML attribute-value pair, (such as color = "#FF0000") that will eliminate the path strings in the negative set from becoming instances of Ω_s and still retain all the positive instances e.g. $\Omega_s = \backslash tr\backslash td[1]\backslash h1\backslash\backslash font[1]@[color = "\#FF0000"]\backslash text()$. If we are unable to find such an attribute-value pair in $P \cup N$ then the learning algorithm would fail meaning that no regularity exists for this attribute in the referral page. The steps sketched above can all be automated and their details are skipped.

3 Service Directory Mining System Implementation

In this section we describe our implementation of a mining system based on the extraction algorithms of the previous section to create a service directory from web pages

The system consists of three main components: an *acquisition* component, a *classification* component and an *extraction* component. The acquisition component retrieves HTML pages from the web that are likely to be relevant for the intended domain of services. This is done by doing a keyword search for the service with a web search engine. The search engine returns a number of urls pointing to pages that match the keywords. All of these web pages are fetched by the acquisition component. The classification component filters the retrieved pages into a set of web pages that the classifier has judged to be actually relevant for the intended service.

The extraction component, driven by the service ontology, does unsupervised extraction of attribute values from classified pages and builds the services direcxtory.

The mining system based on these components and the process flow are illustrated in Figure 5 above.

For classification we used a open source Naive Bayes classifier, namely, the Naive Bayes classifier in the Rainbow package [3]. For training the classifer one hand picks examples of web pages that are relevant to the internded domain of services. These serve as the positive examples. One must also choose pages unrelated to the service as the negative examples.

A Naive Bayes classifier constructs a bag-of-words representation of every document. It associates a probability of occurrence with every word in the document based on the frequency of its occurrence. So words which are more relevant to the domain being mined have a higher information content in the statistical model that is constructed upon completion of training. The user will use this model to pick the words that have a high degree of mutual information content [34]. They serve as the keywords for the web search engine. The idea is that a search based on these keywords is likely to yield many more relevant pages.

The acquisition component is implemented by a web retrieval agent. This is illustrated in Figure 5. The

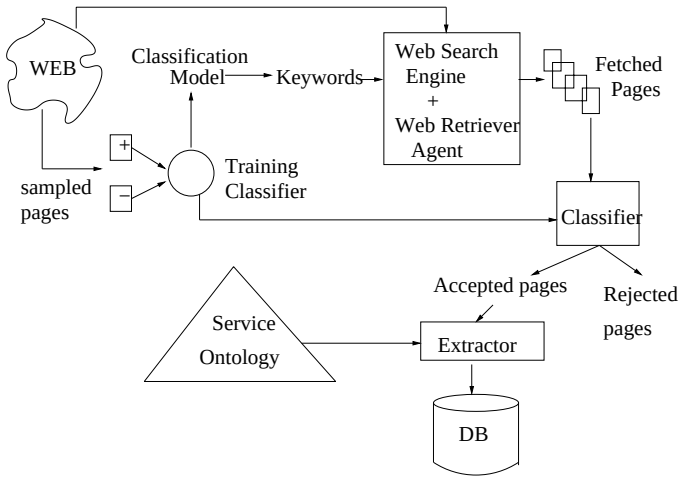


Figure 5: The System Architecture

agent takes as input a list of urls and automatically fetches them. These are fed to the trained classifer that partitions them into "accept" and "reject" categories. The accepted pages are the relevant ones.

The extraction component extracts attribute values associated with service providers from the pages in the accepted category. It is based on the extraction algorithm described in the previous section.

We remark that the mining system admits an open architecture in the sense that its components are interchangeable, i.e., any other acquisition or classification component can be readily "plugged" into our system. We chose Naive Bayes since it was readily available as an open source software.

In the next section we discuss a case study we conducted using this mining system.

4 Building Service Directories: A Case Study

We used the mining system described in the previous section to create a service directory for veterinarians[1]. Through this exercise we provide evidence of the practicality of our extraction algorithms. Specifically we provide quantitative figures of its precision and extraction yield. We also provide evidence of the effectiveness of our learning algorithm for recall. To validate the ease with which we can tune the ontology for other service domains we repeated the exercise of directory building, albeit on a much smaller scale, for real estate services.

For veterinarian service providers, we built an ontology consisting of two concepts: the *Service Provider* concept at the root, and the concept *Veterinarian*. The *Service Provider* concept consists of the attributes *service provider name, street, city, state, zip, phone, email, url*. Note that these attributes are common across any service domain. The concept *Veterinarian* consists of the attribute *vet's name*. In addition, *Veterinarian* inherits all the attributes of *Service Provider*. The attributes *phone*, *email* and *vet's name* are multi-valued while the other attributes are single-valued. Regular expressions were used to identify *phone number, email, state* and *zip* in a page. Rules were used to identify *street*, *vet's name* and *service provider name*. A database of titles in the names of veterinarians (like Dr., Drs., D.V.M., etc), and commonly occurring words in street names (like St., Blvd., Avenue, etc), and in veterinarian provider names (like Hospital, Clinic, etc) was used to locate them in a page. The text within the nearest enclosing HTML tag that contains any word in this database is "part-of-speech" tagged using the Brill's Tagger [13]. The chunk of words tagged as a noun phrase is extracted as the value of the attribute. In addition, we assume that a street name begins with a P.O. Box or Apt. number followed by the name of the street. Thus the rule for extracting the street attribute also collects any chunk of words tagged as cardinals that precede the noun phrase.

We trained the classifier by picking roughly equal number of veterinarian home/referral and non-veterinarian

[1]This directory was built for a large pet food manufacturer.

web pages. Specifically we picked 371 of the former and 303 of the latter as the positive and negative examples respectively for the training set. We examined the classification model that was built using the training set and picked the top ten words that the model identified as possessing high mutual information content. These became the keywords for doing a web search using Google. The search yielded 13,691 distinct pages. The trained classifier was used to select the relevant pages from them. Classification identified 3400 pages as positive or relevant to the veterinarian domain. The extraction algorithm was run on all of the 3400 pages.

We provide experimental results of this case study below. From these 3400 positively classified pages, 950 were identified as home pages of veterinarian service providers while 1900 were identified as referral pages by the extraction algorithm. The $< city, state, zip >$ triple was used as the key. There were about 550 pages with missing zips that were discarded. Table 1 shows the statistics of different attribute values collected for these 2850 pages.

Attribute	Number of Records	
	Home Pages	Referral Pages
City	950	12300
State	950	12300
Zip	950	12300
Street	950	12300
Phone	806	10930
Email	938	780
Doctor Name	711	3930
Hospital Name	856	12300
URL	950	-

Table 1: Number of records extracted for each attribute for home and referral pages

The *precision* and *recall* statistics for each of the attributes in both home and referral pages are shown below in Table 2.We define precision for an attribute as the fraction of the number of records correctly extracted to the total number of records extracted for that attribute. Recall for an attribute has been defined as the fraction of the number of records correctly extracted to the number of records actually present for that attribute. The precision and recall figures have been estimated by sampling over all the 2850 home and referral pages. Table 2(a) shows the precision and recall statistics for home pages while Table 2(b) shows the statistics for referral pages. For referral pages, the data shows the impact of learning XPath expressions to discover missed attributes.

Attribute	Precision(%)	Recall(%)
City	100	100
State	100	100
Zip	100	100
Street	95	96
Phone	85	100
Email	69	100
Doctor Name	95	98
Hospital Name	92	100

(a) Data for Home Pages

Attribute	without XPath		with XPath	
	Precision(%)	Recall(%)	Precision(%)	Recall(%)
City	100	100	100	100
State	100	100	100	100
Zip	100	100	100	100
Street	87	100	87	100
Phone	100	100	100	100
Email	100	100	100	100
Doctor Name	91	88	93	95
Hospital Name	93	63	96	100

(b) Data for Referral Pages

Table 2: Experimental Results for the Veterinarians Domain

For comparison we retrieved a total of 650 email addresses and 990 urls of veterinarian service providers

listed in http://vetquest.com, http://vetworld.com and the yellow pages in http://www.superpages.com. In contrast our mining system yielded 1718 emails and 950 urls of home pages.

We also tested the scalability of our extraction algorithms by building a small directory of real estate agents. The mining system, as described in the previous section, was run on a sample of 25 home pages and 25 referral pages of real estate agents. This yielded us recall and precision figures, for all the service attributes, comparable to that of the veterinarian domain.

Re-engineering the ontology: The veterinarian ontology was re-engineered to build an ontology for real estate agents. The only change required was in creating the new concept of *Real Estate Agency* and ascribing to it the multi-valued attribute *agent's name*. The attribute identifier functions used for *city, state, zip, phone, email* and *url* remained the same. A new rule for extracting the *service provider name* based on a new set of keywords like Real, Estate, Agency etc., was created. These keywords were easily identified by sampling a few real estate web pages. Thus our experience indicates that re-engineering an ontology built for one service domain and adapt it to another domain turned out to be a very easy exercise.

5 Related Work

Information extraction and mining from web sources [30] has emerged as an important area of study. In this section we briefly overview the research on this topic and contrast it with the work presented in this paper.

Retrieval using keyword-based search engines is immensely popular. They are used to find relevant information about available services. But they do not extract structured entities from web pages and hence users browse through the links to discover attributes of services. Search engines have high recall but suffer from low precision [16]. For mining services from web sources, we can exploit the recall characteristics of search engines for retrieving web documents relevant for the intended service domain prior to extraction. Our experiments suggest that such a process generates high extraction yield (i.e. total number of attributes extracted).

Extraction from semi-structured sources by *wrapper generation* methods is an extensively researched topic [35]. The general idea behind wrappers is to parse the document source and create rules for extracting attributes. This is done either manually [11, 28, 26] or semi-automatically [9, 10, 36, 31, 40, 12, 38, 27, 25]. Wrappers have several disadvantages: (a) a significant amount of work is required to generate the rules, and (b) they are document specific as they rely on the syntactic relationship between HTML tags and the attribute value for proper extraction. Wrappers are therefore brittle to changes in the document structure. In contrast our extraction algorithms are independent of any page specific relationships between HTML tags and attribute values. All that is needed is an ontology for the intended service domain. With such an ontology extraction from any document relevant to that domain can be carried out.

Techniques from Natural Language Processing (NLP) have also been applied to information extraction [17]. These techniques typically construct extraction rules either automatically [37] or by learning from labeled examples [39]. The extraction rules are generated based on the semantics of the document rather than syntax as is the case with wrapper-based methods. The attribute identifier functions used in our ontology correspond to such extraction rules. The rules generated by NLP techniques can be used in our ontology as the identifier functions. Thus NLP techniques can enhance the richness of our ontology and contribute to higher precision during extraction.

Information extraction techniques as embodied in [23, 19, 20, 18, 15] use machine learning methods. These techniques use supervised algorithms for creating extraction rules. Observe that the creation of the ontology is the only supervised step in our approach. The attribute identifier rules in our ontology can range from simple keyword-based regular expressions to complex rules learnt from labeled examples. Thus the complexity of these rules govern the effort required for engineering an ontology. Regardless of the ontology, our algorithm for associating attribute values to their corresponding service entity in a web document is unsupervised. Supervised

machine learning techniques that will handle multiple entities in a document are as yet not known.

Query languages for semi-structured documents constitutes an important class of extraction techniques [8, 14, 29, 33, 32]. They all assume that the document schema is known a priori. In our approach we make no such assumptions. In fact given the large and diverse collection of documents from which service entities are mined, it is not practically feasible to know the schema of these documents.

The work that comes closest to ours is [21, 22]. In this work the extraction problem is formulated as one of detecting boundaries between records [22] and hence is applicable to referral pages. The boundary detection is based on several different heuristics. These heuristics assume that: (a) all the attributes appear as the immediate children of the node (in the parse tree of the document) with the highest fan-out, (b) all the attributes are laid out below a single node in the document tree, and (c) there exists an unique HTML tag that separates the records.

The first assumption can result in incorrectly identifying a node as containing all the attributes of all the entities in the document. For example, there may be four service entities and one of them, say e, may contain six attributes. Then the method in [22] will incorrectly identify the node e as the parent of all the service entity nodes. Identification of the unique separator tag is also based on heuristics driven by assorted statistical assumptions about tag occurrences between the records. Our algorithm makes no such assumptions. In fact, boundary detection emerges as a special case in our algorithm. Since referral pages are usually template driven, there exists an implicit consistent ordering among the attribute occurrences in every entity. Our boundary detection algorithm exploits the existence of such an order to generate a correct partition. Lastly it is worth mentioning that the work in [22] is applicable only to referral pages and does not apply to home pages.

Finally note that, to the best of our knowledge, our unsupervised learning technique for increasing recall has not been previously explored in the literature.

6 Conclusion

Critical Analysis of our approach: Recall that the attribute identifier functions in the ontology locate attribute occurrences in a web document. Engineering an ontology is largely dictated by the the complexity of the identifier functions. In the two case studies reported in this paper we used regular expressions as the identifier extraction functions.

The context within which these expressions were applicable was captured by simple rules in the ontology (for example, the address and name identifier functions). Incorporating complex rules in the ontology can result in improving the precision of extraction considerably. This is an area worthy of investigation. For instance, right now the address identifier rules assumes a sequence in apartment number, street name, city, state, zip which is usually valid in US postal addresses. However, international addresses may not follow such a sequence. It would be interesting to incorporate such simple rules, corresponding to different types of sequences, into a complex rule. Our *scoring* and *marking* algorithm depends on the presence of single-valued attributes to isolate service entities in a referral page. The algorithm will combine attributes corresponding to different entities, and thus generate an incorrect extraction, if the single-valued attribute assumption is violated. In Section 2, we formalized the boundary detection problem as one of partitioning the attribute occurrences such that every partition follows a globally consistent order among the attribute occurrences. We gave a heuristic solution to this problem given our definition of a maximal partition. It would be interesting to investigate the complexity of the general problem and look into various sub-categories (for example, when the key attribute in every entity is identified by the ontology) where the problem can be efficiently solved. In Section 2.2, we proposed an unsupervised algorithm for increasing recall by learning the structural context in which an attribute occurs in a web page. However, we have not explored the complexity issues of the algorithm and it is a topic for future work.

We described techniques for mining service directories from web sources using a services ontology and implemented a system based on these techniques. We reported our experience using this system for building a

directory of veterinarian service providers. Our search was conservative. By relaxing it we have obtained a ten fold increase in the number of HTML documents. Mining them is currently in progress and we expect to achieve a proportionate increase in the yield. Our requirement for the existence of a key to distinguish between home and referral pages resulted in misclassifying some referral pages. Relaxing this requirement is a topic of future work. Using the ordering information for developing a more robust boundary detection algorithm is also a worthwhile problem.

References

[1] In *http://www.w3.org/DOM/DOMTR*.

[2] In *http://www.w3.org/TR/xpath*.

[3] In *http://www-2.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes.html*.

[4] Daml-s. In *http://www.daml.org/services*.

[5] Universal discovery, description and integrationa. In *http://www.uddi.org*.

[6] Web services description language. In *http://www.w3.org/TR/wsdl*.

[7] Web services: Why and how. In *http://www.research.ibm.com/people/b/bth/00WS2001/nagy/pdf*.

[8] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semistructured data. 1(1):68—88, April 1997.

[9] B. Adelberg. NoDoSe: A tool for semi-automatically extracting structured and semi-structured data from text documents. In *ACM SIGMOD Conference on Management of Data*, pages 283–294, Washington, 1998. ACM.

[10] N. Ashish and C. Knoblock. Wrapper generation for semi-structured internet sources. *ACM SIGMOD Record*, 26(4):8–15, 1997.

[11] P. Atzeni and G. Mecca. Cut & paste. In *ACM Symposium on Principles of Database Systems*, pages 117–121, Arizona, June 1997. ACM.

[12] B.Ribeiro-Neto, A.H.L. Laender, and A.S. da Silva. Extracting semi-structured data through examples. In *Proceedings of the International Conference on Knowledge Management*, November 1999.

[13] Eric Brill. Some advances in transformation-based part of speech tagging. In *AAAI, Vol. 1*, pages 722–727, 1994.

[14] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *ACM SIGMOD Conference on Management of Data*, pages 505—516, 1996.

[15] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.

[16] S. Chakrabarti. Data mining for hypertext: A tutorial survey. *SIGKDD: SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery and Data Mining, ACM*, 1, 2000.

[17] J. Cowie and W. Lehnert. Information extraction. 39(1):80–91, January 1996.

[18] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom M. Mitchell, Kamal Nigam, and Sean Slattery. Learning to construct knowledge bases from the world wide web. *Artificial Intelligence*, 118(1-2):69–113, 2000.

[19] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew K. McCallum, Tom M. Mitchell, Kamal Nigam, and Seán Slattery. Learning to extract symbolic knowledge from the World Wide Web. In *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, pages 509–516, Madison, US, 1998. AAAI Press, Menlo Park, US.

[20] Mark Craven, Sean Slattery, and Kamal Nigam. First-order learning for web mining. In *European Conference on Machine Learning*, pages 250–255, 1998.

[21] David W. Embley, Douglas M. Campbell, Randy D. Smith, and Stephen W. Liddle. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *Proceedings of the International Conference on Knowledge Management*. ACM, 1998.

[22] D.W. Embley, Y. Jiang, and Y.-K. Ng. Record-boundary discovery in Web documents. In *ACM SIGMOD Conference on Management of Data*, pages 467–478. ACM, 1999.

[23] Dayne Freitag. *Machine Learning for Information Extraction in Informal Domains*. PhD thesis, Carnegie Mellon University, 1998.

[24] Jeffrey Friedl and Andy Oram. *Mastering Regular Expressions: Powerful Techniques for Perl and Other Tools*. O' Reilly.

[25] Jean-Robert Gruser, L. Raschid, M. E. Vidal, and L. Bright. Wrapper generation for web accessible data sources. In *Proceedings of the Third International Conference on Cooperative Information Systems (CoopIS98)*, pages 14–23, New York City, New York, 1998.

[26] A. Gupta, V. Harinarayan, and A. Rajaraman. Virtual database technology. 26(4):57–61, 1997.

[27] J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting semistructured information from the web. In *In Proceedings of the Workshop on Management of Semistructured Data*, pages 18–25, Tucson, Arizona, May 1997.

[28] J. Hammer, Hector Garcia-Molina, S. Nestorov, Ramana Yerneni, Markus M. Breunig, and Vasilis Vassalos. Template-based wrappers in the tsimmis system. In *ACM SIGMOD Conference on Management of Data*, pages 532–535. ACM, 1997.

[29] D. Konopnicki and O. Shmueli. W3qs: A query system for the world wide web. In *Intl. Conference on Very Large Data Bases*, pages 54—65, 1995.

[30] Raymond Kosala and Hendrik Blockeel. Web mining research: A survey. *SIGKDD Explorations*, 2(1):1–15, 2000.

[31] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence*, volume 1, pages 729–737, Nagoya, Japan, 1997.

[32] L. Lakshmanan, F. Sadri, and I. Subramanian. A declarative language for querying and restructuring the web. In *International Workshop on Research Issues in Data Engineering*, 1996.

[33] A. Mendelzohn, G.A. Mihaila, and T. Milo. Querying the world wide web. In *Proceedings of PDIS*, 1996.

[34] Tom M. Mitchell. *Machine Learning*. Mc Graw Hill.

[35] I. Muslea. Extraction patterns for information extraction tasks: A survey. In *AAAI-99 Workshop on Machine Learning for Information Extraction*.

[36] M. Perkowitz, R.B. Doorenbos, O. Etzioni, and D.S. Weld. Learning to understand information on the internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2):133–153, March 1997.

[37] E. Rillof. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 811–816, 1993.

[38] A. Sahuguet and F. Azavant. Web Ecology: Recycling HTML pages as XML documents using W4F. In *ACM SIGMOD Workshop on Database the Web and Databases (WebBD'99)*, pages 31–35, Philadelphia, Pennsylvania, June 1999.

[39] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. Crystal: Inducing a conceptual dictionary. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995.

[40] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.