

METEOR: METADATA AND INSTANCE EXTRACTION FROM OBJECT REFERRAL LISTS ON THE WEB

SRINIVAS VADREVVU, FATIH GELGI,
SARAVANAKUMAR NAGARAJAN, HASAN DAVULCU

*Department of Computer Science and Engineering
Arizona State University*

Tempe, AZ 85287-5406, USA

e-mail: {svadrevu,fagelgi,nrsaravana,hdavulcu}@asu.edu

The Web has established itself as the largest public data repository ever available in the history of mankind. Even though the vast majority of information on the Web is formatted to be easily readable by the human eye, “meaningful information” is still largely inaccessible for the computer applications. In this paper we present the METEOR system which utilizes various presentation and linkage regularities from referral lists of various sorts to automatically separate and extract metadata and instance information. Experimental results for the *university* domain with 12 computer science department Web sites, comprising 361 individual faculty and course home pages indicate that the performance of the metadata and instance extraction averages 85%, 88% F-measure respectively. METEOR achieves this performance without any domain specific engineering requirement.

Keywords: Web, Semantic, Metadata, Object, Instance, Extraction.

1 Introduction

Scalable information retrieval¹ based search engine technologies have achieved wide spread adoption and commercial success towards enabling access to the Web. However, since they are based on an unstructured representation of the Web documents, their performance in making sense of the available information is also limited. Hence, in order to make the Web more accessible with database style querying and automated reasoning, we need scalable techniques that can identify various types of entities, their attributes and relationships.

In this paper we present the METEOR system which utilizes various presentation regularities² within Web pages and linkage regularities from referral lists of various sorts to automatically extract metadata and instance information.

Thanks to the HTML format, unlike plain text documents, Web pages organize and present their content within nested hierarchies of HTML structures. In this paper we present an algorithm that can detect various HTML regularities and utilize them to structure the Web page content itself into hierarchical *group* structures which contains blocks of highly regularly presented *instances*. Our algorithm is based on the observation that most group instances are usually presented together consecutively and they are also presented with consistent HTML formatting. Our algorithm is *robust* in a well defined sense; it can accurately identify all group instances even in the presence of certain irregularities.

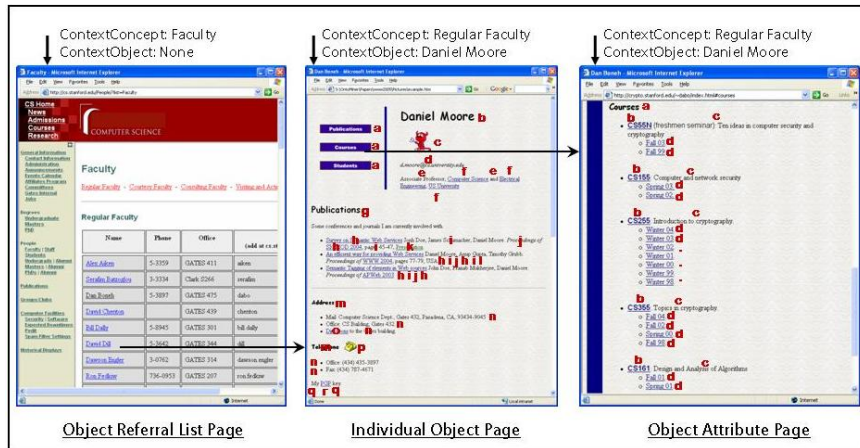


Figure 1. An example of the *object referral list* page, that links individual object pages. The figure shows an individual object page and one of its object attribute pages. The labels in individual object page and the object attribute page are marked with corresponding path identifier symbols.

Furthermore, many Web pages present their information in the form of labeled lists and tables of various sorts. Consider the first page in the example shown in Figure 1 that lists the faculty instances in a computer science department. Each of these faculty instance links to an individual faculty home page with detailed information. We denote these types of groups as *object referral lists* (ORL). Examples of ORLs are faculty listings in the universities, job listings in company sites, course listings in online schools, hotel and hospital listings in directories etc. Object referral lists follow a highly regular linkage pattern; first they list their instances under an informative label such as *jobs*, *faculty*, *hotels* etc. and then each instance links to an individual detailed object page. The individual object page presents the detailed attributes of an object as shown in the second page of Figure 1. Sometimes the individual object page may present its attribute information by linking to an object attribute page as shown in the third page of Figure 1.

In this paper we present algorithms that can interpret any given ORL to navigate to its instances and extract their attributes and values. For a *faculty* ORL this algorithm would typically extract attributes such as *publications*, *researchareas*, *courses*, *students*, *address* and *telephone*. We currently do not align the extracted metadata and instance information with the knowledge structure as it is hard to obtain the domain ontology automatically and we plan to investigate this problem in future. We also currently do not process plain text inside the Web pages. But the metadata extracted from structured parts of the Web pages can be used to extract relationships from text with

the help of a natural language parser. The rest of the paper is organized as follows. Section 2 gives an overview of the related work. Section 3 presents the grouping algorithm for Web pages. Section 4 presents the interpreter algorithm for the ORL structures. Section 5 presents experimental results and Section 6 concludes the paper. For more details, reader may refer to the technical report ³.

2 Related Work

In this section, we present an overview of the related work from several areas and show how our system is different from them.

Wrapper Development Tools: Wrappers ^{4,5} are scripts that are created either manually or semi-automatically after analyzing the location of the data in the HTML pages. Wrappers tend to be brittle against variations and require maintenance when the underlying Web sites change.

Template based algorithms: RoadRunner ⁶ works with a pair of documents from a collection of template generated Web pages to infer a grammar for the collection using union-free regular expressions. ExAlg ⁷ is another system that can extract data from template generated Web pages. ExAlg uses equivalence classes (sets of items that occur with the same frequency in every page) to build the template for the pages by recursively constructing the page template starting from the root equivalence class. Our work differs from such approaches in a way that it performs extraction without the assumption that the object instance pages should be template-driven.

Ontology based approaches: Ontology based approaches ⁸ require a fully developed domain ontology and utilize matching and disambiguation algorithms to locate and extract the entities and relationships of interest. On the other hand, KnowItAll system ⁹ requires only a set of class and relation names with a small set of generic rule templates with an assessor algorithm to extract and rank facts from free text segments. The Armadillo system ¹⁰ extracts RDF triplets from Web pages that adhere to a pre-specified ontology. Our METEOR approach differs from these approaches in that it is domain independent and it does not require a domain specific ontology.

3 Semantic Partitioning

The Semantic Partitioner infers hierarchical relationships among the leaf nodes of the DOM (Document Object Model) tree of a Web page, where all the document content is stored. Semantic partitioner achieves this through a sequence of two operations: *hierarchical grouping* and *promotion*.

3.1 Hierarchical Grouping

During the preprocessing step, the Web page is parsed and its DOM tree is generated. Next, each leaf node is labeled by its attributed root-to-leaf HTML tag path. Each path is assigned a unique path identifier thus yielding a sequence of path identifiers for each Web page. The hierarchical grouping is based on a regular pattern mining algorithm which yields a hierarchy of groups (G) and their instances (I). The Hierarchical Grouping algorithm in Algorithm 1 identifies all the consecutive group instances and their boundaries. The algorithm works by detecting approximate repeating substrings of a sequence. The standard algorithms to learn regular expressions to identify repeating instances within sequences either require *large* sets of *labeled* examples¹¹ or their assumptions are too limiting for the web page instances. For example, the k-mismatch tandem repeat algorithm¹² requires a pre-estimate of k for each grouping in the Web pages. Since, the seminal work of Gold¹³ showed that the problem of inferring a Deterministic Finite Automata (DFA) of minimum size from positive examples is NP-complete, we have chosen to develop a set of Web page independent assumptions that holds for a large variety of groups and their instances on the Web. Next, we present our assumptions and a grouping algorithm that correctly identifies all *ideal groupings* whenever our assumptions hold.

We use the individual object page shown in Figure 1 to demonstrate the inner-workings of our hierarchical grouping algorithm. The path sequence corresponding to that example as marked in the figure is *aaabcdefeffghijhikhijhilhijhikmnnonmpnnnqrq*, where each symbol in the sequence corresponds to the root-to-leaf HTML tag of the labels in the page. In this path sequence, the substrings *aaa*, *bcdefeff*, *ghijhikhijhilhijhik*, *mnnonmpnnnqrq* correspond to the navigation bar, affiliations, publications, and the contact information respectively.

In order to present our algorithms we need the following definitions.

Atom: Given a sequence and a header identifier, an atom is defined as any substring that starts with an occurrence of a header and extends until its next occurrence. In the above example given a header symbol *a*, the sequence *aaa* yields the atoms *a*, *a* and *a*. Similarly, given header symbol *h*, the substring *hijhikhijhilhijhik* yields the atoms *hij*, *hik*, *hij*, *hil*, *hij*, *hik*.

Atom Compatibility: Two atoms *A* and *A'* are *atom compatible*, denoted by $A \equiv A'$, if their *pattern signature* aligns with only insertions from either one of them to the other. The pattern signature of atoms is computed as regular expressions in a bottom up fashion using our algorithm. For example, the atoms for the path sequence *abcbgcbgcdabcbhcbcdabcd* for the header *a* are *abcbgcbgcd*, *abcbhcbcd*, *abcd* and their corresponding pattern signatures would be $abc(bgc)^*d$, $a(bc)^*bcd$ and *abcd*. Hence, the first and second atoms are compatible with the third atom. However, the first and second atoms are not compatible.

Algorithm 1 Hierarchical Grouping Algorithm

HierarchicalGrouping(S)

Input: S : A sequence of path identifier symbols for a Web page

Output: Pattern Signature P for the string S

```
1:  $pat\_list := \phi; pat := \phi;$ 
2:  $cur := S.first; found := true;$ 
3: while  $cur <> end.of(S)$  do
4:    $atoms[] := get\_atoms(cur, S);$ 
5:    $comp\_atoms := \phi;$ 
6:   if  $|atoms[]| > 1$  then
7:      $cand\_sig := HierarchicalGrouping(atoms[0]);$ 
8:     for  $comp \in atoms[]$  and  $comp \neq atoms[0]$  do
9:        $comp\_sig := HierarchicalGrouping(comp);$ 
10:       $aligned := align(cand\_sig, comp\_sig);$ 
11:      if  $aligned$  then
12:         $comp\_atoms := comp\_atoms \cup comp;$ 
13:      end if
14:    end for
15:    if  $|comp\_atoms| > 0$  then
16:       $pat\_list := maximize(atoms[0],$ 
17:         $comp\_atoms, pat\_list);$ 
18:       $pat := append(pat, pat\_list.last);$ 
19:       $cur := cur + S.index(pat\_list.last);$ 
20:    else
21:       $pat := append(pat, S.element(cur));$ 
22:       $cur := cur + 1;$ 
23:    end if
24:    else
25:       $pat := append(pat, S.element(cur));$ 
26:       $cur := cur + 1;$ 
27:    end if
28:  end while
29: return  $pat$ 
```

Instance: An *instance* is defined as a sequence of one or more contiguous atoms.

Instance compatibility: An instance I is compatible with another instance I' , denoted by $I \equiv I'$, if the following three conditions are satisfied:

- The total number of atoms in both instances I and I' are equal.
- $\forall A_i \in I, \exists A'_i \in I'$, such that $A_i \equiv A'_i$.
- The alignments of corresponding A_i and A'_i should be consistent in the sense that either atoms of I inserts into corresponding atoms of I' or

vice versa. For example, the instances $hijhik$ and $hijhil$ are not instance compatible since their second atoms hik and hil are not atom compatible. However the instances $abcbgcbged$, $abcbhcbcd$ are both instance compatible with $abcd$.

Now we can state our assumptions as follows:

Instance Header Assumption: All instances of a group and their sub-group instances always begins with the same header. For the example mentioned above, the header h for the substring that corresponds to the Publications section in the Web page satisfies our assumption yielding the the instances $hijhik|hijhil|hijhik$, which are individual publications in the Web page.

Group Assumption: A *group* is defined as a sequence of two or more contiguous instances such that $\forall I \in G, \exists I' \in G$, such that $I \equiv I'$. The two possible groupings for the substring $hijhikihijhilhijhik$ are $hijhik|hijhil|hijhik$ and $hij|hik|hij|hil|hij|hik$. In these two groupings, the first grouping breaks the publications section into individual publications, whereas the second grouping breaks the individual publications.

Our group assumption relies on the intuition that a sequence of mismatching instances can still be made into a group, if for each instance there exists another compatible instance anywhere in the group that can act as its witness. Hence, our group definition is a *robust* one in the sense that, it can accurately identify all group instances even in the presence of irregularities whenever each type of irregular instance is witnessed by at least another instance.

Since any pairwise compatible atom grouping constitutes a grouping, the following measure enables our algorithm to prefer larger multi-atom instances over simple atom groupings.

Ideal Grouping: A grouping is said to be an *ideal grouping* if it maximizes the pairwise compatibility among its instances after all the pairwise incompatibilities are accounted for. The degree of compatibility for a group is defined as

The degree of PairwiseCompatibility(G) =

$$\sum_{I_i, I_j \in G, i < j} |\{(I_i, I_j) | I_i \equiv I_j\}| - |\{(I_i, I_j) | I_i \not\equiv I_j\}|$$

For example, the pairwise compatibilities for some of the possible groupings of the sequence $hijhikihijhilhijh$ are:

$$\text{PairwiseCompatibility}(hijhik|hijhil|hijhik) = -1 \text{ and}$$

$$\text{PairwiseCompatibility}(hij|hik|hij|hil|hij|hik) = -7.$$

The first group maximizes the number of atoms in each instance and hence it is preferred, which in fact identifies the correct instances of publications in the individual object page in Figure 1.

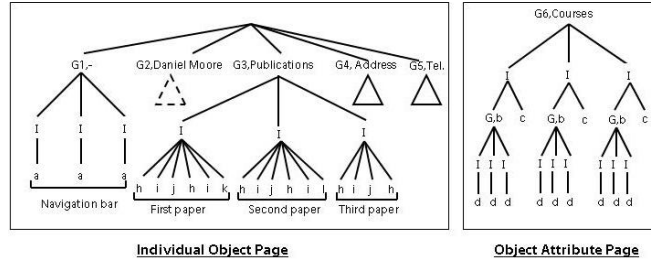


Figure 2. The group structures for the individual object page and the object attribute page in Figure 1. The groups $G_1, G_2, G_3, G_4, G_5, G_6$ correspond to the navigation bar, affiliations, publications, address, the telephone information and the courses respectively.

Theorem: The *Hierarchical Grouping Algorithm* presented in Algorithm 1 computes an ideal grouping G for any given sequence S that satisfies the Instance Header and Group assumptions.

Demonstration with Example: We illustrate the inner-workings of the Algorithm 1 using the sequence $aaabcdefefghijhikhijhikhikmnnonmpnqrq$ from the individual object page in Figure 1 and explain the process.

The *HierarchicalGrouping* algorithm attempts to standardize the path sequence as a regular expression which can be used to parse the original sequence into hierarchical group structures, presented in Figure 2. It utilizes the *maximize* subroutine that finds the maximum number of consecutive atoms that can be appended to the current atom. For example, initially the subpattern $(a)^*$ is computed using the *maximize* subroutine by recursively invoking the *HierarchicalGrouping* algorithm. This pattern corresponds to the regularly presented structure, the navigation bar of the individual object page shown as G_1 in Figure 2. Then the algorithm continues to find the subpatterns $bcd(e(f)^*)^*$, $g(hij(hik|hil|hik))^*$ that corresponds to the affiliations and the publications presented in the page shown as G_2 and G_3 , and appends to the previous subpattern $(a)^*$. The algorithm eventually generates nested group structures presented in Figure 2 from the final pattern. The complexity of the algorithm is $O(n^3)$ where n is the length of the input string.

3.2 Promotion

The final step in semantic partitioning is *promotion*. After hierarchical grouping, all the content of the Web page is still at the leaf nodes of the hierarchical group tree and hence promotion of some of the leaf nodes is necessary in order to organize them into a semantic hierarchy. The promotion algorithm identifies those leaf nodes that should be promoted above their siblings. A label is promoted if it satisfies one of the following rules:

- A (sub)group structure can be labeled with its nearest preceding *emphasized* node if there is no other (sub)group structure between them;
- A value instance can be labeled with its previous *emphasized* node

A node label is said to be *emphasized*, if it satisfies one of the following conditions:

- It's labeled text is fully capitalized, or
- There is a bold tag (such as , <bold>, <h1> etc.) on its DOM tree tag-path, or
- It's immediate consecutive group structure corresponds to an HTML list structure (or)

In the individual object page in Figure 1 the nodes satisfying the *emphasized* condition are *Daniel Moore*, *Publications*, *Address*, and *Telephone*. Since all of them satisfy the preconditions for promotion they are all promoted as *group headers* as shown Figure 2.

Once the hierarchical grouping and promotion steps are completed we obtain the final semantic partitioning for the given Web page. In the next Section we present an algorithm to separate and extract metadata and instance information from such semantic structures.

4 Interpreting the Semantic Structures

The semantic partitioning algorithm presented in Section 3 utilizes presentation regularities encoded with the HTML markup of the Web pages to transform them into hierarchical group structures and identify their instances. In this section we describe how these semantic structures can be interpreted by utilizing linkage regularities that exist within the context of an ORL in order to separate and extract their metadata and instances.

4.1 Extracting the Value Types

The atomic value types of a group may denote attribute names, attribute ranges, subconcepts of a concept or member objects of a concept. For example, the group structure G_3 of the object page in Figure 2 contains different types of values that corresponds to the ranges of various attributes of a *Publication* such as its *Author*, *Title*, *Conference*, *PageNo* and *Location*.

Value types are retrieved from the atomic leaf nodes of the group structures by utilizing a segment of their HTML path information. Let G be a group structure and v_i be a value that belongs to an instance I of G . Let L be the lowest common ancestor of all of the values of G in the corresponding HTML tree. For each v_i its *path index* p_i , is defined to be its order-indexed HTML tag path from L to v_i itself. The value types are then obtained by grouping all atomic values, v_i 's of a group structure, by their corresponding

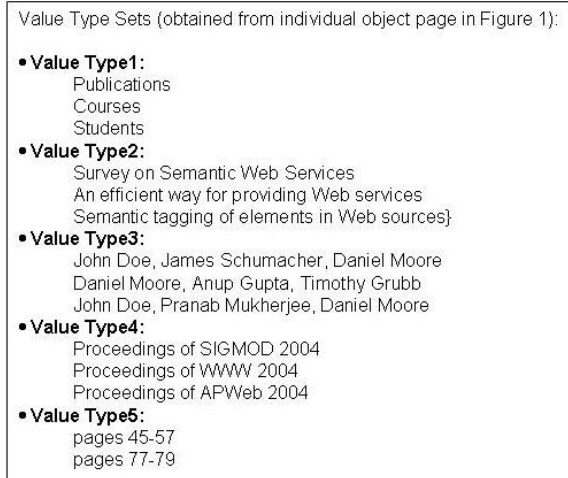


Figure 3. Extraction of Value Types from Group Structures

path indexes. Figure 3 presents some of the value types obtained from groups G_1 and G_3 of the individual object page of Figure 1.

4.2 The Semantic Structure Interpreter

This section provides an algorithm to interpret any given group structure originating from an ORL. For convenience, the extracted metadata and object instances are represented as F-logic¹⁴ facts. F-logic provides a means to represent both the metadata information and objects in a concise way and it allows reasoning with rules¹⁵. It is also very easy to transform extracted F-logic facts into the semantic Web standard RDF and RDFS to enable sharing. We use a subset of the F-logic alphabet that consists of function symbols, constants, and first-order terms over them. The details of the F-logic syntax can be found in¹⁵.

As an example to illustrate the interpretation, consider the ORL group structure G in Figure 1 that lists the instances of the *Regular Faculty* concept. As the group structure presents the members of the ‘Faculty’ concept and its value types are found by the Hierarchical Grouping algorithm, the following F-logic statements are extracted from the ORL group G .

‘Regular Faculty’ : concept.
‘Alex Aiken’ : ‘Regular Faculty’.
‘Daniel Moore’ : ‘Regular Faculty’. ...
‘Regular Faculty’[‘Name’ \Rightarrow {‘Alex Aiken’, ‘Daniel Moore’, ...}].
‘Alex Aiken’[‘Name’ \rightarrow ‘Alex Aiken’].

Website	# of Pages	Semantic Partitioner			Metadata Extraction			Value Types Ext.		
		<i>P</i>	<i>R</i>	<i>F</i>	<i>P</i>	<i>R</i>	<i>F</i>	<i>P</i>	<i>R</i>	<i>F</i>
<i>Faculty</i>										
cs.wisc	32	88%	83%	85%	71%	80%	74%	83%	95%	88%
cs.cmu	33	65%	98%	78%	89%	79%	81%	79%	77%	76%
cs.unc	41	100%	71%	83%	94%	96%	94%	90%	100%	94%
cs.umd	44	86%	68%	76%	100%	89%	85%	91%	97%	94%
cs.stanford	38	86%	80%	83%	92%	75%	81%	85%	97%	87%
cs.washington	40	68%	77%	72%	73%	83%	77%	82%	90%	85%
Average		83%	75%	80%	87%	84%	84%	85%	93%	87%
<i>Courses</i>										
cs.wisc	26	73%	93%	82%	91%	87%	89%	91%	80%	85%
cs.uiuc	41	87%	76%	86%	89%	84%	91%	84%	91%	88%
eeecs.mit	12	91%	88%	82%	100%	60%	75%	83%	100%	90%
cs.bu	23	69%	67%	71%	93%	78%	85%	85%	85%	85%
cs.princeton	18	74%	82%	78%	100%	85%	92%	88%	100%	94%
cs.rpi	13	83%	77%	80%	71%	71%	83%	83%	83%	83%
Average		80%	81%	80%	91%	78%	86%	86%	90%	88%

Table 1. Experimental Results for Semantic Partitioner, Metadata Extraction, and Value Types Extraction for *Faculty* and *Courses* domains from computer science department Web sites.

‘Regular Faculty’[‘Phone’ \Rightarrow {‘5-3359’, ‘3-3334’, ...}].
‘Alex Aiken’[‘Phone’ \rightarrow ‘5-3359’]. ...

The interpretation proceeds for the group structures in the individual object page by providing the appropriate context. Whenever, the object instance pages present their attributes using a link group, each group structure G'' within the object attribute pages is interpreted and corresponding value types are extracted.

5 Experimental Results

In this section we present the experimental results for the semantic partitioning and the ORL interpretation algorithms.

5.1 Experimental Setup

The data set we used to test our algorithms consists of ORL pages for the *Faculty* and *Course* domains in 12 computer science department Web sites, comprising 228 individual faculty and 133 individual course pages. For each web site we identified the corresponding faculty and course ORL pages that present link groups that link to individual faculty and course home pages in

that university.

5.2 Evaluation

Table 1 presents the experimental results for the semantic partitioner and the interpretation algorithm for separating and extracting the metadata and the value types for faculty and course collections. The results are produced for all the Web pages reachable from the ORLs. The precision and recall of Hierarchical Grouping algorithm for a given Web page is calculated by comparing the transitive closure of parent-child relationships inferred by the *algorithmically generated* hierarchies with those implied by the *gold-standard* hierarchies. The *gold-standard* hierarchy for each page and taxonomy was created manually by an evaluator and then the transitive closure of all parent-child relationships were computed. The precision and recall is calculated by the following formulas: $Precision, P = \frac{|T \cap T'|}{|T|}$; $Recall, R = \frac{|T \cap T'|}{|T'|}$ where T and T' are the sets of transitive closure of parent-child relationships implied by the *algorithmically generated* and *gold-standard* hierarchies respectively.

5.3 Discussion

It can be observed from the experimental results that the performance of the semantic partitioner, metadata and value types extraction algorithms average 84% F-measure for *faculty* domain and 85% F-measure for the *course* domain respectively. The semantic partitioner algorithm was able to perform well on many of the individual object pages whenever there was a regularity in the presentation and layout of the content. But the algorithm is still sensitive to certain types of irregularities in the data sources. On the other hand, the interpretation algorithm was able to perform with high accuracy whenever the corresponding semantic partitioner algorithm was successfully able to identify the appropriate group structures along with their header labels.

6 Conclusions and Future Work

In this paper, we presented the METEOR system that can automatically separate and extract metadata and instance information from *object referral lists*. The experimental results indicate that the METEOR system was able to extract the metadata and the instance information with high accuracy. In our future work, we propose to develop automated algorithms for finding all the ORL structures within any Web site.

Acknowledgements. This work was partially supported by the Office of Naval Research (ONR) under its Multidisciplinary Research Program of the University Research Initiative (MURI) under Grant No. N00014-04-1-0723.

References

1. R.Baeza-Yates and B.Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
2. G.Yang, W.Tan, S.Mukherjee, I.V.Ramakrishnan, and H.Davulcu. On the power of semantic partitioning of web documents. In *Workshop on Information Integration on the Web*, Acapulco, Mexico, 2003.
3. S.Vadrevu, F.Gelgi, S.Nagarajan, and H.Davulcu. Meteor: Metadata and instance extraction from object referral lists on the web. Technical Report TR-05-009, Arizona State University, 2005.
4. J.Hammer, H.Garcia-Molina, S.Nestorov, R.Yerneni, M.M.Breunig, and V.Vassalos. Template-based wrappers in the tsimmis system. In *ACM SIGMOD Conf. on Management of Data*, 1997.
5. N.Kushmerick, D.S.Weld, and R.B.Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence*, pages 729–737, 1997.
6. V.Crescenzi, G.Mecca, and P.Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.
7. A.Arasu and H.Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD Conf. on Management of Data*, San Diego, USA, 2003.
8. S.Dill, N.Eiron, D.Gibson, D.Gruhl, R.Guha, T.Kanungo, A.Jhingran, S.Rajagopalan, A.Tomkins, J.A.Tomlin, and J.Y.Zien. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *WWW Conference*, 2003.
9. O.Etzioni, M.Cafarella, D.Downey, S.Kok, A.Popescu, S.Soderland, T.Shaked, D.S.Weld, and A.Yates. Web-scale information extraction in knowitall (preliminary results). In *WWW Conference*, 2004.
10. F.Ciravegna, S.Chapman, A.Dingli, and Y.Wilks. Learning to harvest information for the semantic web. In *Proceedings of the 1st European Semantic Web Symposium*, Heraklion, Greece, 2004.
11. R.C.Berwick and S.Pilato. Learning syntax by automata induction. In *Machine Learning 2*, pages 9–38, 1987.
12. D.Gusfield. *Algorithms on Strings, Tree, and Sequences*. Cambridge University Press, 1997.
13. E.M.Gold. Complexity of automaton identification from given sets. In *Information and Control*, pages 37:302–320, 1978.
14. M.Kifer, G.Lausen, and J.Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, July 1995.
15. G.Yang, M.Kifer, and C.Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. Catania, Sicily, Italy, 2003.