The RoadRunner Web Data Extraction System

Valter Crescenzi[1], Giansalvatore Mecca[2] and Paolo Merialdo[1]

[1] Università Roma Tre, D.I.A., Via Della Vasca Navale 79, Roma, Italy
[2] Università della Basilicata, D.I.F.A., c.da Macchia Romana, Potenza, Italy

(Extended Abstract)

1 Introduction

Extracting data from HTML text files and making them available to computer applications is becoming of utmost importance for developing several emerging e-services. This paper presents RoadRunner, a research project that aims at developing solutions for automatically extracting data from large HTML data sources. We concentrate on *data-intensive* Web sites, that is, sites that deliver large amounts of data through a complex graph of linked HTML pages. The paper describes the top-level software architecture of the RoadRunner *System*, which has been specifically designed to automatize the data extraction process.

The paper is organized as follows. First, Section 2 illustrates an overview of the project and gives an intuition of its key ideas. Then, Section 3 describes the overall architecture of the RoadRunner system. Section 4 concludes the paper discussing related works.

2 The Road Runner Project: Overview

The target of our research are data-intensive Web sites, i.e. sites that publish a large amount of data organized in a complex hypertext structure. These sites usually share a number of common features. First, data are stored in a back-end DBMS, and HTML pages are dynamically generated using *scripts* from the content of the database; simply speaking, these scripts run queries on the database – possibly nesting the original relational tables – and return the result-set in HTML format. Second, the site contains different *classes of pages*, corresponding to different contents in the site; e.g., a fictional e-commerce site may contain different pages for the different items sold through the site – books, software titles, DVDs, – and also pages about customers reviews. Pages in the same class are usually generated by the same script; e.g., the page listing price and other details about ”Microsoft Windows 2000” will be most probably generated by the same script as the page about ”Norton Antivirus”; on the contrary, the page listing books by Stephen King will be generated by a different script.

In this perspective, we see the site generation process as the result of *encoding* the original database in a textual, markup-based format (HTML) [7]. More specifically, we see each script used in the site as performing an encoding function that yields one class of pages in the site; this encoding goes from a database query result-set (possibly nested) to a collection of HTML sources.

Wrappers as Regular Grammars In this respect, we see the wrapping process as a form of *decoding* of the original information from the HTML pages. Loosely speaking, we may say that a wrapper performs a function that is the inverse of the one implemented by the encoding script that generated a given page, i.e., it retrieves the generating query result-set from the HTML code and allows to store it back in structured form. Given the close correspondence between nested database types and regular expressions [7], in our approach a wrapper is a regular grammar that can be parsed against an HTML page to retrieve some data items.

To be more precise, given an alphabet of terminal symbols Σ, and a set of non-terminal symbols N, we consider a subset of the regular grammars, corresponding to regular expressions built over $\Sigma \cup N$ using the following operators (ϵ is the empty sequence): (i) concatenation, of the form $a \cdot b$, i.e., the sequence of a and b; (ii) iteration, of the form a^+, i.e., the repetition of a one or more times; (iii) "hooks" of the form $(a)?$, a shortcut for $a|\epsilon$.

For example, the following regular expression may define a wrapper in our formalism ($ denote non-terminals):

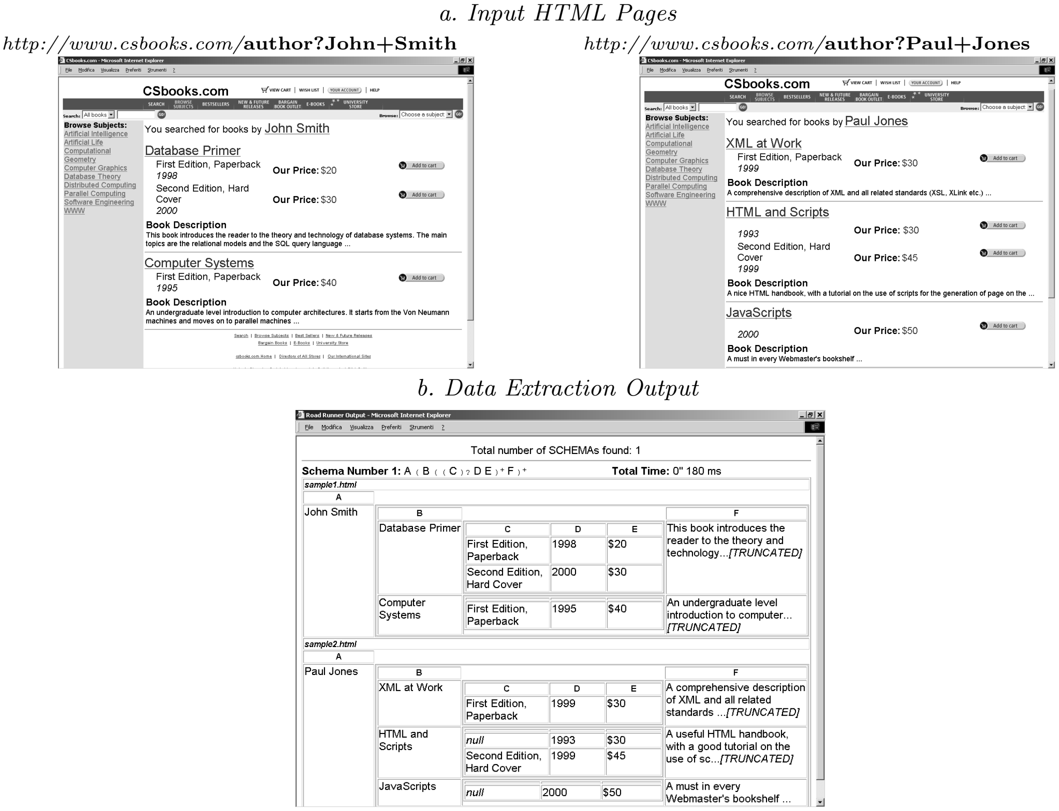
```
<HTML> ... Author: < B > $A < /B > <UL>
  (<LI> <FONT> $B </FONT> (<I> $C </I>)? <BR/>our price: <STRONG> $D </STRONG> </LI>)+
</UL> ... </HTML>
```

Note that, with respect to general regular grammars we allow for a very limited form of disjunction, basically only the ones hidden inside hooks (zero *or* one), and inside iterations (one *or* more). In essence, our wrapper corresponds to union-free regular grammars. This, of course, introduces a limitation in the expressive power of the wrapping language. However, our experience tells that the language includes the patterns that usually occur in fairly structured HTML sources [4].

Inferring a Wrapper: the ACME Technique Since wrappers essentially parse the HTML code, they are targeted at a specific page organization and layout. Therefore, in order to extract data from a site, we need to derive one wrapper for each page class in the site. In RoadRunner, this decoding activity is based on the similarities that are exhibited by HTML pages belonging to the same class, i.e., we try to infer a grammar for a class of pages by looking at a number of samples, and then use this grammar as a wrapper.

We have developed an original technique – called *the ACME matching technique*, for *Align, Collapse, Match and Extract* – to infer a wrapper for a class of pages by analyzing similarities and differences among some sample HTML pages of the class [4]. In essence, given a set of sample HTML pages, our technique compares the source HTML codes, in order to find matching and mismatching parts and, based on this knowledge, progressively refines a common wrapper.

Data Extraction in RoadRunner The output wrapper generated by the ACME technique is a grammar that can be parsed against the pages of the class to extract data items. These ideas are clarified in Figure 1, which refers to a fictional bookstore site. In that example, we assume pages listing all books by one author are generated by a script that queries a database to produce the page for a given author. When applied on these pages (Figure 1.a), the ACME technique compares the HTML codes of the two pages, infers a common structure and a wrapper, and uses that to extract the source dataset. Figure 1.b shows the actual output of the current implementation of ACME after it is run on the two HTML pages in the example. Several things are worth noting here. 1) Since the matching technique is based on the comparison of pages of the same type, one critical assumption is that HTML pages coming from the site have been somehow clustered into


CSbooks.com - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
VIEW CART | WISH LIST | YOUR ACCOUNT | HELP
SEARCH
BROWSE SUBJECTS
BESTSELLERS
NEW & FUTURE RELEASES
BARGAIN BOOK OUTLET
E-BOOKS
UNIVERSITY STORE
Search: All books GO!
Browse: Choose a subject GO!
Add to cart
Search | Browse Subjects | Best Sellers | New & Future Releases
Bargain Books | E-Books | University Store
csbooks.com Home | Directory of All Stores | Our International Sites
Road Runner Output - Microsoft Internet Explorer

a. Input HTML Pages

*http://www.csbooks.com/***author?John+Smith**

CSbooks.com

Browse Subjects:
Artificial Intelligence
Artificial Life
Computational Geometry
Computer Graphics
Database Theory
Distributed Computing
Parallel Computing
Software Engineering
WWW

You searched for books by John Smith

Database Primer

First Edition, Paperback *1998* **Our Price:** $20

Second Edition, Hard Cover *2000* **Our Price:** $30

Book Description
This book introduces the reader to the theory and technology of database systems. The main topics are the relational models and the SQL query language ...

Computer Systems

First Edition, Paperback *1995* **Our Price:** $40

Book Description
An undergraduate level introduction to computer architectures. It starts from the Von Neumann machines and moves on to parallel machines ...

*http://www.csbooks.com/***author?Paul+Jones**

CSbooks.com

Browse Subjects:
Artificial Intelligence
Artificial Life
Computational Geometry
Computer Graphics
Database Theory
Distributed Computing
Parallel Computing
Software Engineering
WWW

You searched for books by Paul Jones

XML at Work

First Edition, Paperback *1999* **Our Price:** $30

Book Description
A comprehensive description of XML and all related standards (XSL, XLink etc.) ...

HTML and Scripts

1993 **Our Price:** $30

Second Edition, Hard Cover *1999* **Our Price:** $45

Book Description
A nice HTML handbook, with a tutorial on the use of scripts for the generation of page on the ...

JavaScripts

2000 **Our Price:** $50

Book Description
A must in every Webmaster's bookshelf ...

b. Data Extraction Output

Total number of SCHEMAs found: 1

Schema Number 1: A (B ((C)? D E)+ F)+ **Total Time:** 0" 180 ms

sample1.html

A	B	C	D	E	F
John Smith	Database Primer	First Edition, Paperback	1998	$20	This book introduces the reader to the theory and technology...*[TRUNCATED]*
		Second Edition, Hard Cover	2000	$30	
	Computer Systems	First Edition, Paperback	1995	$40	An undergraduate level introduction to computer... *[TRUNCATED]*

sample2.html

A	B	C	D	E	F
Paul Jones	XML at Work	First Edition, Paperback	1999	$30	A comprehensive description of XML and all related standards ...*[TRUNCATED]*
	HTML and Scripts	*null*	1993	$30	A useful HTML handbook, with a good tutorial on the use of sc...*[TRUNCATED]*
		Second Edition, Hard Cover	1999	$45	
	JavaScripts	*null*	2000	$50	A must in every Webmaster's bookshelf ...

Fig. 1. Data Extraction in RoadRunner

the different classes they belong to; then, to support the above algorithms, there is the need to adopt some clustering techniques for HTML pages that allow to quickly classify pages based on their type; these techniques should aim at giving a good approximation of the page classes in the site, in order to carry on the decoding step. 2) Sites usually also contain singleton pages, i.e., pages such that there is no other page in the site with the same organization and layout. These pages mainly serve the purpose of offering browsable access paths to data in the site; the ACME technique, which is based on comparing two or more pages and finding similarities, obviously is not applicable to these singleton pages; therefore there is the need to develop specific techniques for wrapping also these pages. 3) As it can be seen from the example in Figure 1.b, whenever we infer a nested schema from the pages belonging to one class, the inferred schema has anonymous fields (labeled by A, B, C, D, etc. in our example). Since our ultimate goal is that automatizing the whole data extraction process, one intriguing problem is to develop methods to automatically discover attribute names.

In light of these ideas, we have designed the software architecture of the RoadRunner system to support the ACME technique for the automatic extraction of data from Web sources.

3 The RoadRunner System Architecture

Figure 2 shows the top-level architecture of the RoadRunner system.

We identify four main modules, each addressing a specific problem, as follows:

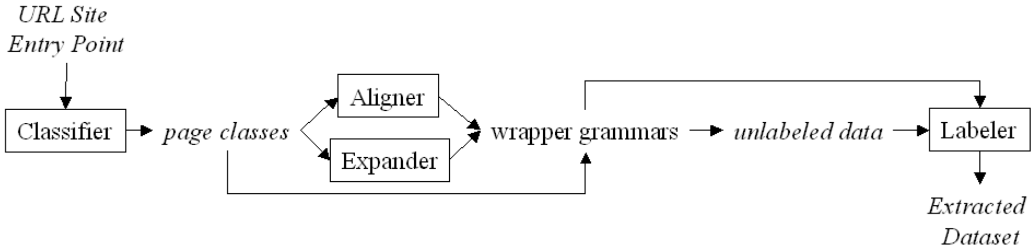


Fig. 2. The Road Runner Architecture

- Pages from the target site are analyzed by the *Classifier* that collects them into clusters with a homogeneous structure, i.e. it tries to identify the page classes offered by the site. This module incorporates a crawler that navigates the target site, and exploits a number of heuristics for producing a good approximation of the page classes in the site; note that some of these classes may contain several candidate pages and will be fed to the Aligner for wrapper generation; other classes will have singleton elements.
- Wrapper generation for classes of similar pages is performed by a module called *Aligner*, which implements the *ACME* technique; for each class of pages, the Aligner compares the HTML sources of some sample pages to infer a grammar to be used as a wrapper for the whole class.
- Classes having singleton pages are fed to a module called *Expander*, which tries to infer a wrapper for them; wrappers generated by this Expander are based on different techniques with respect to those used by the Aligner.
- Finally, the *Labeler* associates a semantic meaning to the data fields that can be extracted by running the wrappers generated by the above modules against the site pages; i.e. it gives an appropriate name to each non-terminal symbol of the wrapper grammar.

In the following we briefly illustrate the main features of the four components. We do not discuss technical details, but rather focus on the ideas that have inspired the design of each module, pointing out the results of some preliminary experiments we have already conducted, and open issues.

3.1 The Aligner

The Aligner implements the ACME technique, which represents the core of our system. In the following we sketch a description of the ACME technique. For a deep description we refer the interested reader to [4], which also reports the results of several experiments.

ACME takes as input HTML page sources as list of tokens (a lexical analyzer transforms pages into lists of tokens); each token is either an HTML tag or a string value. Then ACME works on two objects at a time: (*i*) a list of tokens, called the *sample*, and (*ii*) a wrapper, i.e., one union-free regular expression. Given two HTML pages, to start we take one of the two, for example the first page, as an initial version of the wrapper; then, the wrapper is progressively refined trying to find a common regular expression for the two pages. This is done by progressively solving *mismatches* between the wrapper and the sample. The wrapper can then be further refined by iteratively applying the same technique over the samples of a collection of HTML pages of the same class.

Our experiments show that a small number of sample pages (3-6) is sufficient to infer a grammar wrapper for all the pages of a large class.

3.2 The Classifier

The goal of the Classifier is to efficiently identify the different pages classes in the target sites. For each class, a number of samples will be given as input to the Aligner. The wrappers generated by the Aligner will then be run over all the pages of each class, extracting data from the target site.

The Classifier crawls a site in order to cluster its HTML web pages according to the grammar they obey to. The clustering process is based on known techniques. Roughly speaking we may say that these techniques see the samples to cluster as points in a n-dimensional real space, usually called the *feature space*; clusters are then generated trying to minimize the average distance between points in a cluster (see [8] for a survey on the topic). One key point in this process is choosing the right mapping of a sample to the feature space. This is done by extracting some relevant (numeric) *features* from the sample and using them as coordinates in the feature space.

In our context, determining useful features becomes an intriguing problem, completely different from the classical ones. In fact, we have a radically new notion of similarity between our samples, that is, the compliance to a common regular grammar. To give an intuition of how this new notion of similarity makes our clustering problem different from the classical string clustering problem, consider the following: (*i*) usually two strings are considered similar if they contain common sub-parts. On the contrary, in the Web page case, requiring that two pages contain common sub-parts is not sufficient to guarantee that they can be parsed using the same grammar; (*ii*) strings that are considered similar by classical methods have approximately the same length; in our case, pages in the same class may have largely different sizes due to different cardinalities of patterns under a Kleene's star.

We have therefore identified several features that can give information about the similarity of pages as needed in our context:

- *Tag Probability* it is reasonable to assume that pages complying to the same grammar have a similar "distribution" of tags, i.e., tags appear in the pages with similar probability; we have therefore considered tag probabilities as possible features for the Web page clustering problem;
- *Tag Periodicity* there are cases in which tag probabilities may be misleading, since they do not give information about the relative positions of tags. To complement tag probability with another feature that gives us information on tag distribution, we apply to HTML pages a variant of the classical frequency spectrum method.
- *Distance from the Home Page* this feature aims at catching properties related to the topology of the site graph. If navigation paths in the site are well organized, it is reasonable to assume that pages containing homogeneous information are approximately at the same distance from the home page in the site graph.
- *URL Similarity* in many sites, URLs of pages in the same class follow some common patterns, either due to the fact that the HTML files are stored in a common physical folder of the server, or that the pages are generated by the same script. To take this into account, we extract a measure of similarity about URLs.

We have experimented the Classifier over the pages from several sites. The results are encouraging and suggest the lines along which improving the method. We have identified some new features that could help to produce more reliable clusters. In particular, we expect the method can be improved by introducing new features extracted from the DOM tree associated with each page.

3.3 The Expander

The Expander is responsible for extracting data from singleton pages. Usually the role of these pages in a site is to collect links to other pages, and the anchor of these links is a value that is repeated in the destination page. For example, consider a bookstore site; a possible singleton page in the site is the one that presents a list of all the book genres sold by the store. This page offers links that lead to pages presenting books of a given genres. For the sake of usability, the names of the various genres compare both in the singleton page and in the destination pages. Then, we do not need extract data from singleton pages, since the data they offer is redundant with that provided by other (richer) pages.

However, these pages might be useful to efficiently maintain the extracted data up-to-date with the Web source. So far we have described the data extraction as a one-shot process made by the following steps: 1) download all the site pages, 2) build wrappers for them, 3) apply the wrapper to extract relevant data. In order to keep data up-to-date, we should periodically repeat this process, every time downloading the whole site. An alternative strategy is to leave data in the site, and to build a *virtual dataset*. Whenever one query is issued against the dataset, the system has to navigate the site and extracts data relevant to the query. As we argue in [11], singleton pages providing access paths can efficiently support this approach. Therefore inferring a wrapper also for singleton pages may become a relevant issue.

Here the problem has specific objectives: we are not interested in extracting data provided by these pages, but we aim at building a wrapper for these pages in order to allow the system to navigate the whole site. This goal can be achieved by applying wrapper inference techniques developed in other contexts. The crucial point is that we may assume to have at disposal some sample data for singleton pages: since they offer redundant data, they contain data that can be extracted elsewhere in the site by applying the ACME technique. In other words, we have positive examples to infer the wrapper grammar. This is the task addressed by several techniques known in the literature, such as, for example, Nodose [1], or Stalker [12] (see Section 4). Thus, we have planned to integrate one of these techniques into the RoadRunner architecture in order to support the Expander.

3.4 The Labeler

The Aligner is able to infer a wrapper and its associated schema for a class of pages; however, anonymous names are assigned to attributes over the schema, as shown in the example of Figure 1.b. The goal of the Labeler is to associate a meaningful name to each attribute of the extracted dataset. Clearly this step could be done manually; however, in order to automatize every facet of the data extraction process we are currently investigating techniques to discover an appropriate name for each field.

One possible solution to the problem relies on the adoption of knowledge representation techniques: analyzing the extracted data, and exploiting the knowledge managed by some domain ontology, it may be possible to deduct a meaning for the fields. However, important information about data is available on the Web pages themselves. Since Web sites are intended to be browsed by humans, it is a common practice that the data published into HTML pages are accompanied by textual descriptions to help the user directly and correctly interpret the underlying information. In many cases these descriptions are indispensable to correctly present data to the user. For example, consider how prices are presented in e-commerce Web sites: without the help of strings such as "our price" and "saving", information would be completely misunderstood. Also, textual descriptions are often associated to data items organized into tables; usually, the first row reports labels to describe the contents of the columns.

Based on these observations we are setting up several methods to analyze the HTML code of the sample pages processed by the Aligner in order to find strings that represent good candidates for naming the fields of the data set. Our methods are based on a generalized notion of closeness between wrapper's tokens and non-terminal symbols. In particular, we deal with the twofold nature of an HTML wrapper: on the one hand the wrapper can be seen as a sequence of symbols; on the other hand, since the wrapper keeps the nested structure of an HTML document, it can be seen as a DOM tree as well, introducing special nodes to denote iterations and hooks.

When searching for the name of a given field the Labeler explores the sequential representation of the wrapper, looking for a text string which is adjacent to the non-terminal. The "adjacency" is defined with respect to this specific context, and several characteristics of the HTML format concur at defining this properties. When analyzing non-terminal symbols which are included in a repeated pattern, the Labeler also considers the DOM tree representation of the wrapper. The idea here is to check whether the pattern sub-tree is adjacent with some isomorphic sub-tree. In this case, the leaves of the discovered tree can be selected as names for the non-terminals of the pattern tree. Also in this case the notions of adjacency and isomorphism are defined with respect to our specific context.

These methods assume that for each field a description is present in the page (and then into the wrapper grammar). However, there are many situations in which data are published in the Web page leaving their meaning implicit. For example, in a page presenting the details of a sold book, it might not be necessary to explicitly associate the book's title a string to describe that what follows is the book title; it is assumed that the user is able to interpret the right semantics to that data item. Clearly, in these cases the above techniques fail. To face these cases, we are currently studying an approach that relies on the richness of the Web itself, and which is inspired to Brin's DIPRE technique [3]. The Web can be considered as a huge knowledge base, where a particular piece of information may be published by thousands of independent sites. Clearly each site may adopt different presentation policies. Therefore, with respect to our specific problem, it is possible that in some page a given data item is associated with some information describing its meaning. Intuitively, for those non-terminal symbols whose names have not been derived by the above techniques, the system could extract some sample data item, and give them as input to a Web search engine. It is reasonable that in some of the pages retrieved by the search engine, the input value is explicitly associated with some descriptive text.

4 Related Work

Early attempts to automate the wrapper generation process heavily relied on the use of heuristics. For example, in [2], the authors develop a practical approach to identify attributes in a HTML page; the technique is based on the identification of specific formatting tags (like the ones for headings, boldface, italics etc.) in order to recognize semantically relevant portions of a page.

More recently, other proposals have attacked the problem under a different perspective. One example in this category is the work in [10, 9]. The authors develop a machine–learning approach to wrapper induction. The starting point for these works is the identification of several simple classes of wrapper–specification formalisms, that are easily learnable and yet sufficiently expressive. Wrappers then inferred from a set of multiple–record HTML sample documents. A similar approach is pursued in Stalker [12]. Given an arbitrarily nested, multiple–record document, Stalker relies on a hierarchical description of the page content, corresponding to the nesting of lists of records inside the page (i.e., a list of restaurants, with name and address and a list of dishes, with prices etc.).

Another wrapper generation system from labeled examples is represented by NoDose [1]. In this work, the wrapper is derived by making the system interact with the user through a graphical interface. The user manually starts the semi-automatic structuring phase by defining the logical scheme in the page in some data model; then, s/he labels a few occurrences of the relevant attributes in a page, and then asks the system to generalize those examples and infer the wrapper.

One final example of wrapper generating systems from labeled examples comes from [5, 6]. These works have a conceptual–modeling background, and base the data-extraction process on the use of domain-specific ontologies. The target are multiple-record HTML pages coming from specific domains for which an ontology is available. The ontology provides a concise description of the conceptual model of data in the page and also allows for recognizing attribute occurrences in the text. It thus allows for labeling a number of examples in the target page, and try to infer the wrapper on those occurrences.

References

1. B. Adelberg. NoDoSE – a tool for semi-automatically extracting structured and semistructured data from text documents. SIGMOD'98, 1998.
2. N. Ashish and C. Knoblock. Wrapper generation for semistructured Internet sources. *Workshop on the Management of Semistructured Data (SIGMOD'97)*, 1997.
3. D. Brin. Extracting patterns and relations from the World Wide Web. WebDB'98, 1998.
4. V. Crescenzi, G. Mecca, and P. Merialdo. roadRunner: Towards automatic data extraction from large Web sites. Technical Report RT-DIA-64-2001, D.I.A. - Università di Roma Tre, March 2001.
5. D. W. Embley, D. M. Campbell, Jiang Y. S., S. W. Liddle, Ng Y., D. Quass, and Smith R. D. A conceptual-modeling approach to extracting data from the web. ER'98, 1998.
6. D. W. Embley and Jiang Y. S.and Ng Y. Record-boundary discovery in web documents. SIGMOD'99, 1999.
7. S. Grumbach and G. Mecca. In search of the lost schema. ICDT'99, 1999.
8. A. K. Jain, N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.
9. N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118:15–68, 2000.
10. N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper induction for information extraction. IJCAI'97, 1997.
11. G. Mecca, A. Mendelzon, and P. Merialdo. Efficient queries over Web views. EDBT'98, 1998.
12. I. Muslea, S. Minton, and C. A. Knoblock. A hierarchical approach to wrapper induction. In *Third Annual Conference on Autonomous Agents*, 1999.