Automatic Web Information Extraction in the roadRunner System

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

[1] D.I.A. – Università di Roma Tre
[2] D.I.F.A. – Università della Basilicata

Abstract. This paper presents roadRunner, a research project that aims at developing solutions for automatically extracting data from large HTML data sources. The target of our research are *data-intensive* Web sites, i.e., HTML-based sites with a fairly complex structure, that publish large amounts of data. The paper describes the top-level software architecture of the roadRunner *System*, and the novel research challenges posed by the attempt to automate the information extraction process.

1 Introduction

Extracting data from HTML pages and making them available to computer applications is becoming of utmost importance for developing several emerging e-services. Many new classes of applications aim at leveraging on the huge amount of data delivered on the Internet as HTML pages. For example, a service which is becoming quite popular on the net is that provided by the so called *shopping agents*, that is software modules that navigate the network looking for better prices of specific items on e-commerce sites. Another important example is represented by the needs of modern portals, whose goal is offering integrated access to several Web services (like home banking, on-line trading, billing, credit card checking etc.) from a single interface.

These applications need to manipulate data delivered into HTML pages by several Web sites. Nevertheless, since the target consumers of Web data sources are humans – rather than machines – the data encoded into HTML pages cannot be directly handled by applications: they need to be extracted from HTML pages and made comprehensible to machines. Data extraction from HTML Web pages is usually performed by software modules called *wrappers*, i.e. programs that are able to extract data items from HTML pages and return them in a structured format (e.g. in XML). However, writing wrappers for HTML by hand may in some cases be a labor-intensive task. Also, manually coded wrappers are usually quite brittle, since even small changes at the site may prevent the wrapper from working properly. Therefore, maintaining applications that need to process data extracted from the Web becomes a costly and complex issue.

roadRunner is a research project that aims at developing solutions for automatically extracting data from large HTML data sources. This paper presents an overview of the project and describes the top-level software architecture of

the ROADRUNNER *System*, which has been specifically designed to automate the data extraction process.

The paper is organized as follows. First, Section 2 gives an overview of the project goals, trying to convey an intuition of the key ideas and of the original features with respect to other proposals in the literature. Section 4 describes the overall architecture of the ROADRUNNER system. The following sections then concentrate on the description of the various modules.

2 Overview

The target of our research are *data-intensive* Web sites, i.e., HTML-based sites that publish large amounts of data in a fairly complex structure. Data are usually 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. Roughly speaking, we may say that each script generates a *class of pages* with homogenous content and layout. These sites usually contain different *classes of pages*, corresponding to different contents in the site, like, for example, pages about MP3 players sold on *amazon.com* or pages about stock quotes on *yahoo.com*. Our goal is that of running database–like queries on these pages, like "name and brand of the MP3 player with the lowest price" or "last variation of Microsoft stock certificates".

To do this, we follow a two–step approach: (*i*) we provide a wrapper layer that extracts relevant pieces of information from the original HTML pages in the sites, and returns them as a collection of XML documents on the same DTD or XMLSchema; to give an example, the XML documents derived for pages about MP3 players on `amazon.com` might contain a list of `<player>` elements, each with `<name>`, `<brand>`, `<price>` subelements, plus an optional sublist of comments and ratings written by customers who purchased that player. (*ii*) Then, based on these wrappers, an XML query language (like XQuery [5], or any other of the other languages that have been recently proposed [3]) can be used to express and evaluate queries on these derived XML documents.

Given the availability of several XML query languages, it can be seen that most of the complexity of this process lies in the generation of wrappers that perform the translation from HTML to XML. In light of this, in this paper we concentrate primarily on this problem, and assume that an external XML–query language engine is available to run queries on the resulting XML docs.

2.1 Supervised Grammar Inference

Generating a wrapper for a set of HTML pages corresponds to deriving a grammar for the HTML code in the page – usually a regular grammar – and then use this grammar to parse the page and extract pieces of data. Grammar inference is a well known and extensively studied problem (for a survey of the literature see, for example, [18]). However, regular grammar inference is a hard problem: first,

it is known from Gold's works [10] that regular grammars cannot be correctly identified from positive examples alone; also, even in presence of both positive and negative examples, there exists no efficient learning algorithm for identifying the minimum state DFA that is consistent with an arbitrary set of examples [11]. As a consequence, the large body of research that originated from Gold's seminal works has concentrated primarily on finding efficient algorithms for *supervised* grammar learning, i.e., algorithms that work in presence of additional information, typically a set of labeled examples or a knowledgeable teacher's responses to queries posed by the learner.

The fact that regular expressions cannot be learned from positive examples alone, and the high complexity of the learning even in presence of additional information have limited the applicability of the traditional grammar inference techniques to Web sites, and have recently motivated a number of pragmatical approaches to wrapper generation for HTML pages. These works have attacked the wrapper generation problem under various perspectives, going from machine–learning [9, 20, 15, 13, 16] to data mining [1] and conceptual modeling [8, 19]. Although these proposals differ in the formalisms used for wrapper specification, they all inherit the "supervised learning" approach of traditional grammar inference, and share a number of common features:

- *manual selection of sample pages*: first, it is assumed that a human supervisor selects a collection of homogeneous HTML pages from which data should be extracted; the wrapper generation system usually derives the wrapper by working on a single page at a time; then, the wrapper is used to parse the remaining pages, possibly generalizing the grammar when this is needed;
- *availability of labeled examples*: second, the wrapper generator works by using additional information on the content of the page under exam, typically a set of labeled samples provided by the user or by some other external tool; the wrapper is inferred by looking at these positive examples and trying to generalize them;
- *a priori knowledge about the target schema*: third, it is usually assumed that the wrapper induction system has some *a priori* knowledge about the page organization, i.e., about the schema of data in the page; most works assume that the target pages contain a collection of flat records; in other cases [1] the system may also handle nested data, but it needs that the user specifies what are the attributes to extract and how they are nested.

In essence these works deal with a data extraction problem that we may summarize as follows: "given one or more HTML pages, a target schema (usually a list of flat records) for these pages, and a set of labeled examples, find a set of extraction rules that allow to parse the HTML code and retrieve data items according to the target schema".

2.2 Our Approach

Our research departs quite significantly from these proposals and investigates the wrapper generation problem under a new perspective. In particular, we aim

at automating the wrapper generation process to a larger extent; for this reason, we investigate *unsupervised* wrapper generation, as follows:

- first, *we do not assume that sample pages are manually selected* from a human designer; on the contrary, we assume that the system is able to automatically cluster pages in a (portion of a) site into homogeneous classes;
- second, *our system does not rely on user-specified labeled examples*, and does not require any interaction with the user during the wrapper generation process; this means that wrappers are generated and data are extracted in a completely automatic way;
- finally, *we do not assume any a priori knowledge about the target schema*, i.e., our system does not know the schema according to which data are organized in the HTML pages: this schema will be inferred along with the wrapper; moreover, our system is not restricted to flat records, but can handle arbitrarily nested structures;

In essence, we may say that, with respect to other works in the literature, we deal with a different problem, a *schema finding and data extraction problem* [12], which we may summarize as follows: "given a set of HTML pages, find a schema (arbitrarily nested) for the content of these pages, and a set of extraction rules that allow to parse the HTML code and retrieve the data according to the discovered schema". This change of perspective on the overall problem of information extraction from Web pages represents the main source of originality of this research.

In fact, on the one side, this forced us to reconsider the problem of grammar inference for HTML pages, and devise new techniques to deal with it; the main intuition behind our proposal is that pattern discovery can be based on the study of similarities and dissimilarities between the pages; in particular, we have proposed [6] a novel approach to wrapper inference for HTML pages, in which, in order to tell meaningful patterns from meaningless ones, our system works with two HTML pages at a time, and mismatches are used to identify relevant structures.

On the other side, we had to face a number of side research problems that had not been considered before, since the presence of user-provided inputs made them irrelevant. These problems go from that of selecting collections of homogeneous pages in a site, i.e., pages with the same structure, to that of labeling attributes in the target schema once this has been derived (i.e., stating that string "$15.99" is a price, and "Java for Dummies" is a book title).

In the following sections we describe the various modules that compose the architecture or our system, and how these concur to the solution of these problems.

3 The Wrapper Generation Algorithm

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>...
  Player: <B>$Name</B> $Brand  - our price: <STRONG>$Price</STRONG>
          <UL> (<LI> <FONT>$Rating</FONT> (<I>$Review</I>)?)+ </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 many of the typical patterns that occur in fairly structured HTML sources [6].

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 recently developed an original technique to infer a wrapper for a class of pages by analyzing similarities and differences among some sample HTML pages of the class [6]. 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. The output wrapper 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, pages listing all books by one author are generated by a script; the script first queries a database to produce a nested dataset (Figure 1.a) by nesting books and their editions inside authors; then, it serializes the resulting tuples into HTML pages (Figure 1.b). When applied on these pages, the Aligner 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.c shows the actual output of the current implementation of the Aligner after it is run on the two HTML pages in the example. (For the sake of readability, the extracted dataset is produced in HTML format. As an alternative, it could be formatted in XML, or stored in a database.)

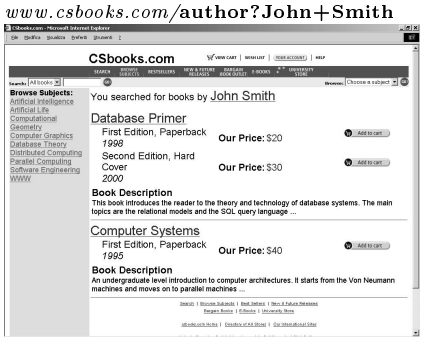
Several things are worth noting here. First, since the matching technique is based on the comparison of pages of the same type, one critical assumption is

a. Source Dataset

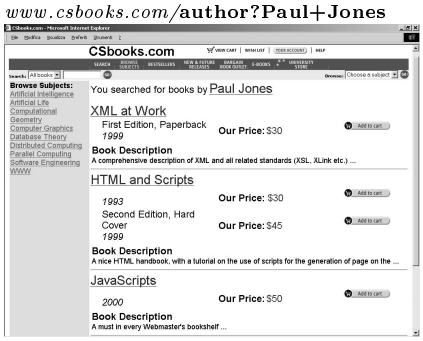
Name	*Books*				
	Title	*Descr.*	*Editions*		
			Details	*Year*	*Price*
John Smith	Database Primer	This book...	First Edition, Paperback	1998	20$
			Second Edition, Hard Cover	2000	30$
	Computer Systems	An undergraduate...	First Edition, Paperback	1995	40$
Paul Jones	XML at Work	A comprehensive...	First Edition, Paperback	1999	30$
	HTML and Scripts	A useful HTML...	*null*	1993	30$
			Second Edition, Hard Cover	1999	45$
	JavaScript	A must in...	*null*	2000	50$
...	...	...	...	...	...

b. HTML Pages

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



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



c. Data Extraction Output

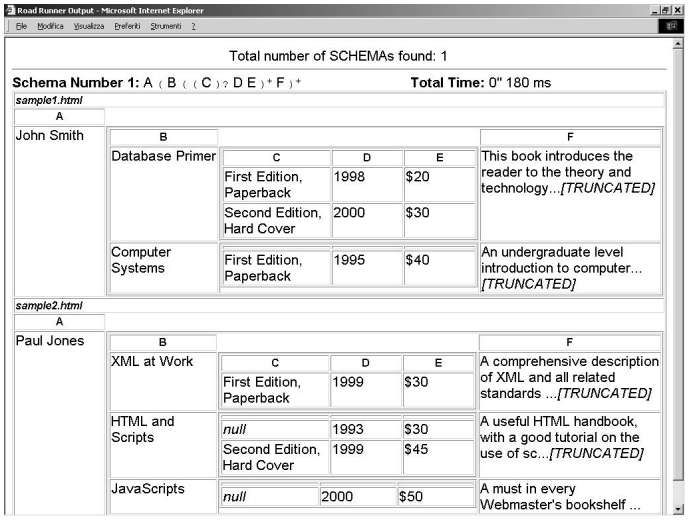


Fig. 1. Examples of HTML Code Generation

that HTML pages coming from the site have been somehow clustered into 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.

Second, 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; the Home Page of a site usually falls in this category. These pages mainly serve the purpose of offering browsable access paths to data in the site; the Aligner, 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 these pages also;

Finally, as it can be seen from the example in Figure 1.c, 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). In order to enhance the semantics of the schema, each field should be associated with a meaningful name. This step could be done manually after the dataset has been extracted. Nevertheless, since our ultimate goal is that automatizing the whole data extraction process, one intriguing alternative is to develop some form of post-processing of the wrapper to automatically discover attribute names.

The software architecture of the ROADRUNNER system, discussed in the next section, is aimed at providing solutions to all of these problems; it is centered around the *Aligner*, i.e., the module that implements the wrapper generation algorithm, and complements it with a number of additional modules.

4 The ROADRUNNER System Architecture

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

We identify four main modules, each of which addresses a specific problem, as follows:

- the *classifier* analyzes pages from the target site and 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 the *Aligner*, which implements the matching 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 inferred by the Aligner;

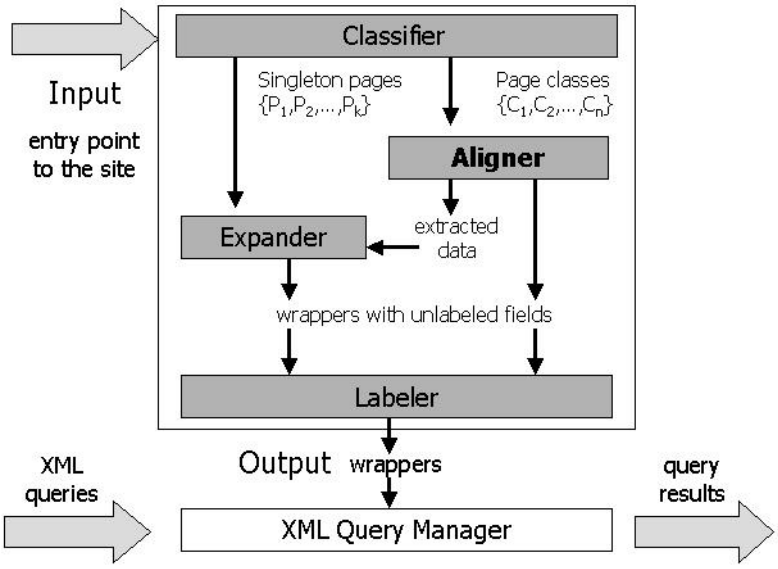


Fig. 2. Architecture of the System

– 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 various components.

4.1 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 [6], 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.

5 The Classifier

The goal of the Classifier [7] 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 in order to generate the corresponding wrappers. 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*; cluster are then generated trying to minimize the average distance between points in a cluster (see [14] 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 – this requires a deeper form of structural similarity; (ii) also, typically 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; consider for example two pages about books, each with the list of books by a certain author; if the first author has published few books (say 3) and the second one many more (say, 15), the sizes of the two HTML sources may be completely different; nevertheless, the two pages are very likely to belong to the same class.

We have identified two families of features that can give information about the similarity of pages as needed in our context. In the former family, there are features that aims at giving information about the internal structure of the pages; clearly these properties are somehow connected to the structure of the common grammar these pages should obey to. In the current implementation we have taken into account the following features:

- *Tag Probability* it is reasonable to assume that pages complying 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; in practice, we calculate for each sample the percentage of occurrences of each tag with respect to the total number of tags, and use these numbers as coordinates in the feature space.
- *Tag Periodicity* there are cases in which tag probabilities may be misleading, since they do not give information about the relative positions of tags. This means that two pages containing approximately the same tags will be considered similar even if the tags are completely rearranged with respect to each other, and therefore the pages are different in terms of the grammar. To complement tag probability with some other feature that gives us information on tag positions, we apply to HTML pages a variant of the classical frequency spectrum method [17].
- *Distance from the Home Page* 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* experience tells that, in most 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 use the classical notion of string similarity [2] in order to associate a number of numerical features with each URL string.

Interestingly, our experiments show that the four classes of features together, usually guarantee a good approximation of the correct classification.

6 The Expander

The Expander is responsible for extracting data from singleton pages, i.e. pages forming a separate class by themselves, so that each class is composed of a unique instance. It is worth noting that usually the role of these pages in a site is to collect access paths, i.e. links, to other pages; usually, 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.

On the basis of these observations, we do not need extract data from singleton pages; they works as indices in a database, and the data items they contain are redundant with those provided by other (richer) pages. Therefore, they do not carry any useful information.

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. Singleton pages providing access paths can efficiently support this approach. Therefore inferring a wrapper also for singleton pages may become a relevant issue. It is worth noting that the problem here has specific objectives: we are not interested in extracting data provided by these pages; our objective is to build 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 have at disposal the data offered in singleton pages: we have seen that they offer redundant data, that is data which can be extracted elsewhere in the site (applying the matching 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 [16]. Thus, we have planned to integrate one of these techniques into the ROADRUNNER architecture in order to support the Expander.

7 The Labeler

The Aligner is able to infer a wrapper and its associated schema for a class of pages; however, as we have said above, non-terminal symbols, which corresponds to attributes over the schema, are assigned anonymous names, as shown in the example of Figure 1.c.

The goal of the Labeler is to associate a meaningful name to each attribute of the extracted data set. 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 some meaning for the fields. However, it is worth observing that 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", pricing data would be completely misunderstood. Also, textual descriptions are often associated to data items organized into tables; usually, the first rows reports labels to describe the

contents of the columns (the usage of table headers is strongly suggested by the W3C itself for the sake of usability).

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 hook. Figure 3 illustrates this concept; note that the marked node represents the iteration.

```
... <TABLE> <TR> <TD> Name </TD>
<TD> Phone </TD> </TR>
(<TR> <TD> $A </TD> <TD> $B </TD> </TR> )+
</TABLE> ...
```

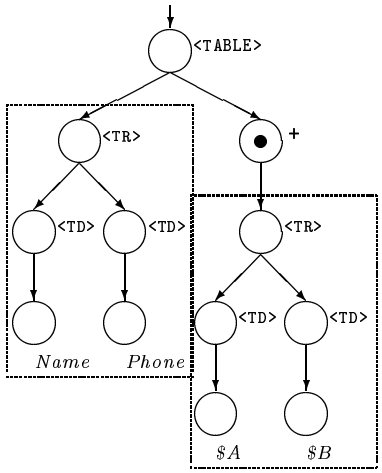


Fig. 3. Wrappers as DOM Trees

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.

Figure 3 gives an intuition of this strategy. The left dashed tree corresponds to a heading for the iterated pattern represented by the right dashed tree: since the two are isomorph the leaves of the former – namely, the strings "name" and "phone" – are candidate to be used as names for the non-terminals $A and $B respectively.

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 [4]. 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.

References

1. B. Adelberg. NoDoSE – a tool for semi-automatically extracting structured and semistructured data from text documents. In *ACM SIGMOD International Conf. on Management of Data (SIGMOD'98), Seattle, Washington*, 1998.
2. R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *International Conference of Foundations of Data Organization (FODO'93)*, pages 69–84, 1993.
3. A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. *ACM SIGMOD Record*, 29(1):68–79, 2000.
4. D. Brin. Extracting patterns and relations from the World Wide Web. In *Proceedings of the First Workshop on the Web and Databases (WebDB'98) (in conjunction with EDBT'98)*, pages 102–108, 1998.
5. D. Chamberlin et al. Xquery 1.0: An xml query language. W3C Working Draft, June 2001.
6. V. Crescenzi, G. Mecca, and P. Merialdo. ROADRUNNER: Towards automatic data extraction from large Web sites. In *International Conf. on Very Large Data Bases (VLDB'2001), Rome, Italy, September 11-14*, pages 109–119, 2001.
7. V. Crescenzi, G. Mecca, and P. Merialdo. Wrapping–oriented classification of Web pages. Submitted for publication, 2001.

8. D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, Y. Ng, D. Quass, and R. D. Smith. A conceptual-modeling approach to extracting data from the web. In *Proceedings of the 17th International Conference on Conceptual Modeling (ER'98)*, pages 78–91, 1998.
9. T. Goan, N. Benson, and O. Etzioni. A grammar inference algorithm for the world wide web. In *AAAI Spring Symposium on Machine Learning in Information Access*, 1996.
10. E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.
11. E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.
12. S. Grumbach and G. Mecca. In search of the lost schema. In *Seventh International Conference on Data Base Theory, (ICDT'99), Jerusalem (Israel), Lecture Notes in Computer Science, Springer-Verlag*, pages 314–331, 1999.
13. C. Hsu and M. Dung. Generating finite-state transducers for semistructured data extraction from the web. *Information Systems*, 23(8):521–538, 1998.
14. A. K. Jain, N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.
15. N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence (IJCAI'97)*, 1997.
16. I. Muslea, S. Minton, and C. A. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 190–197, 1999.
17. A. V. Oppenheim, R. W. Schafer, and J. R. Buck. *Discrete–Time Signal Processing*. Prentice Hall, second edition edition, 1999.
18. L. Pitt. Inductive inference, DFAs and computational complexity. In K. P. Jantke, editor, *Analogical and Inductive Inference, Lecture Notes in AI 397*, pages 18–44. Springer-Verlag, Berlin, 1989.
19. B. A. Ribeiro-Neto, A. H. F. Laender, and A. Soares da Silva. Extracting semistructured data through examples. In *Proceedings of the 1999 ACM International Conference on Information and Knowledge Management (CIKM'99)*, pages 94–101, 1999.
20. S. Soderland. Learning information extraction rules for semistructured and free text. *Machine Learning*, 34(1–3):233–272, 1999.