A Brief Survey of Web Data Extraction Tools

Alberto H. F. Laender  Berthier A. Ribeiro-Neto
Altigran S. da Silva  Juliana S. Teixeira
Department of Computer Science
Federal University of Minas Gerais
31270-901 Belo Horizonte MG Brazil
{laender,berthier,alti,juliana}@dcc.ufmg.br

ABSTRACT
In the last few years, several works in the literature have addressed the problem of data extraction from Web pages. The importance of this problem derives from the fact that, once extracted, the data can be handled in a way similar to instances of a traditional database. The approaches proposed in the literature to address the problem of Web data extraction use techniques borrowed from areas such as natural language processing, languages and grammars, machine learning, information retrieval, databases, and ontologies. As a consequence, they present very distinct features and capabilities which make a direct comparison difficult to be done. In this paper, we propose a taxonomy for characterizing Web data extraction tools, briefly survey major Web data extraction tools described in the literature, and provide a qualitative analysis of them. Hopefully, this work will stimulate other studies aimed at a more comprehensive analysis of data extraction approaches and tools for Web data.

1. INTRODUCTION
With the explosion of the World Wide Web, a wealth of data on many different subjects has become available online. This has opened the opportunity for users to benefit from the available data in many interesting ways [7]. Usually, users retrieve Web data by browsing and keyword searching, which are intuitive forms of accessing data on the Web. However, these search strategies present several limitations. Browsing is not suitable for locating particular items of data, because following links is tedious and it is easy to get lost. Keyword searching is sometimes more efficient than browsing, but often returns vast amounts of data, far beyond what the user can handle. As a result, in spite of being publicly and readily available, Web data can hardly be properly queried or manipulated as done, for instance, in traditional databases.

To manipulate Web data more efficiently, some researchers have resorted to ideas taken from the database area. Databases, however, require structured data. Yet, most Web data is unstructured or semistructured [8] and cannot be manipulated using traditional databases techniques. To address this problem, a possible strategy is to extract data from Web sources to populate databases for further handling [14].

The traditional approach for extracting data from Web sources is to write specialized programs, called wrappers, that identify data of interest and map them to some suitable format. Developing wrappers manually has many well known shortcomings, mainly due to the difficulty in writing and maintaining them. Recently, many tools have been proposed to better address the issue of generating wrappers for Web data extraction [3, 4, 9, 10, 11, 12, 15, 17, 19, 22, 25, 29, 31, 32]. Such tools are based on several distinct techniques such as declarative languages [4, 10, 17], HTML structure analysis [11, 25, 32], natural language processing [15, 29, 33], machine learning [9, 19, 22], data modeling [3, 31], and ontologies [12].

The problem of generating a wrapper for Web data extraction can be stated as follows. Given a Web page $S$ containing a set of implicit objects, determine a mapping $W$ that populates a data repository $R$ with the objects in $S$. The mapping $W$ must also be capable of recognizing and extracting data from any other page $S'$ similar to $S$. We use the term similar in a very empirical sense, meaning pages provided by the same site or Web service, such as pages of a same Web bookstore. In this context, a wrapper is a program that executes the mapping $W$. A common goal of all wrapper generation tools is to generate wrappers that are highly accurate and robust, while demanding as little effort as possible from the wrapper developers. As we shall see, in practice, this imposes an important trade-off between the degree of automation of a tool and the flexibility of the wrappers generated by it.

As more and more tools for Web data extraction continue to appear, the need for the analysis of their capabilities and features arises. In this paper we briefly survey some of the tools proposed in the literature, also discussing how some features, the ones we regard as most important for Web data extraction, are supported by each tool.

A pioneer initiative for comparing tools and techniques for Web data extraction is the RISE Web site [21]. In this
site, experimental results performed using STALKER [29], WHISK [33], SRV [13], WIEN [22], and RAPIER [9], which are tools based on either machine learning or natural language processing, have been made available. These quantitative results were submitted by the authors of the tools themselves. Unfortunately, the rules generated by distinct tools are directly comparable. In [28], Muslea compares and contrasts various types of extraction patterns that are generated by different types of machine learning algorithms. However, due to the difficulty to establish a protocol for such a comparison. The present paper is a first attempt in this direction.

The paper is organized as follows. In Section 2 we introduce a taxonomy for characterizing Web data extraction tools. Section 3 presents an overview of major data extraction tools found in the literature. Following, in Section 4, we present a qualitative analysis of these tools. Finally, Section 5 presents our conclusions.

2. A TAXONOMY FOR CHARACTERIZING WEB DATA EXTRACTION TOOLS

This section presents a taxonomy for grouping the various tools we have studied. This taxonomy is based on the main technique used by each tool to generate a wrapper, what led us to the following groups of tools: Languages for Wrapper Development, HTML-aware Tools, NLP-based Tools, Modeling-based Tools, and Ontology-based Tools. While such a taxonomy is useful for didactic purposes, it must not be taken as the only possibility. In fact, there are cases when a same tool could fit in two or more of the identified groups. In what follows, we describe the main characteristics of the tools belonging to each of the groups.

Languages for Wrapper Development.

One of the first initiatives for addressing the problem of wrapper generation was the development of languages specially designed to assist users in constructing wrappers. These languages were proposed as alternatives to general purpose languages such as Perl and Java, which were prevalent so far for this task. Some of the best known tools that adopt this approach are Minerva [10], TSMMIS [18], Web-QQL [4]. Other tools representative of this approach are FLORID [26] and Jedi [20], but we will not cover them here due space limitations.

HTML-aware Tools.

We group here tools that rely on inherent structural features of HTML documents for accomplishing data extraction. Before performing the extraction process, these tools turn the document into a parsing tree, a representation that reflects its HTML tag hierarchy. Following, extraction rules are generated either semi-automatically or automatically and applied to the tree. Some representative tools based on such an approach are WAF [32], XWRAP [25], RoadRunner [11]. Another tool that can be regard as HTML-aware is Lixto [5] but, due to space limitations, we will not discuss it in this paper.

NLP-based Tools.

Natural language processing (NLP) techniques have been used by several tools to learn extraction rules for extracting relevant data existing in natural language documents. These tools usually apply techniques such as part-of-speech tagging and lexical semantic tagging to build relationships between phrases and sentences elements, so that extraction rules can be derived. Such rules are based on syntactic and semantic constraints that help to identify the relevant information within a document. The NLP-based tools are usually more suitable for Web pages consisting of grammatical text, possibly in telegraphic style, such as job listings, apartment rental advertisements, seminar announcements, etc. Representative tools based on such an approach are RAPIER [9], SRV [13], and WHISK [33].

Wrapper Induction Tools.

The wrapper induction tools generate delimiter-based extraction rules derived from a given set of training examples. The main distinction between these tools and those based on NLP is that they do not rely on linguistic constraints, but rather in formatting features that implicitly delineate the structure of the pieces of data found. This makes such tools more suitable for HTML documents than the previous ones. Tools such as WIEN [22], SoftMelly [19], and STALKER [29], are representative of this approach.

Modeling-based Tools.

This category includes tools that, given a target structure for objects of interest, try to locate in Web pages portions of data that implicitly conform to that structure. The structure is provided according to a set of modeling primitives (e.g. tuples, lists, etc.) that conform to an underlying data model. Following, algorithms similar to those used by the wrapper induction tools identify objects with the given structure in the target pages. Tools that adopt this approach are NoDoSE [3] and DEByE [23, 31].

Ontology-based Tools.

All approaches described previously rely on the structure of presentation features of the data within a document to generate rules or patterns to perform extraction. However, extraction can be accomplished by relying directly on the data. Given a specific domain application, an ontology can be used to locate constants present in the page and to construct objects with them. The most representative tool of this approach is the one developed by the Brigham Young University Data Extraction Group [12].

3. OVERVIEW OF WEB DATA EXTRACTION TOOLS

In this section we overview the Web data extraction tools we have studied. We notice that the list of tools covered here must not be regarded as complete. Although we have tried to cover the most representative tools that have appeared in the recent literature, our is study is not exhaustive. The presentation of the tools follows the taxonomy introduced in Section 2.

3.1 Languages for Wrapper Development

Minerva – An important component of the Araneus system [27] is Minerva [10], a formalism for the development of
wrappers. Minerva combines a declarative grammar-based approach with features typical of procedural programming languages. The grammar used by Minerva is defined in the EBNF style: for each document, a set of productions is defined; each production defines the structure of a non-terminal symbol of the grammar, in terms of terminal symbols and other non-terminals. Minerva is complemented by a language for searching and restructuring documents called Editor, which provides basic operations found in text editors. For each production of the grammar, it is possible to add an exception clause, containing a piece of Editor code. Whenever the parsing of that production fails, an exception is raised and the corresponding exception code is executed.

TSIMMIS – Among other components for semi-structured data management, the TSIMMIS [18] includes wrappers that can be configured through specification files written by the user [17]. Specification files are composed by a sequence of commands that define extraction steps. Each command is of the form [variables, source, pattern] where variables represents a set of variables that hold the extraction results, source specifies the input document to be considered (e.g., a Web page), and pattern allows matching the data of interest within the source. The data stored in the variables can be used as input for subsequent commands. An extractor based on a specification file parses an HTML page to locate the interesting data and extract them. After the last command is executed, the set of variables holds the extracted data. Although there is no language for wrapper development formally defined for TSIMMIS, we included it in this survey because of its historical importance and pioneering.

Web-OQL – Originally aimed at performing SQL-like queries over the Web, Web-OQL [4] is a declarative query language that is capable of locating selected pieces of data in HTML pages. For this, a generic HTML wrapper parses a page given as input and produces as result an abstract HTML syntax tree, called a hypertext, representing the document. Using the syntax of the language, it is possible to write queries that locate data of interest in the hypertext and then output these data in a suitable format (e.g., tables). This is how Web data extraction can be accomplished using Web-OQL. Navigation through hypertexts is also supported.

3.2 HTML-aware Tools

W4F (World Wide Web Wrapper Factory) – W4F [32] is a toolkit for building wrappers. W4F divides the wrapper development process in three phases: first, the user describes how to access the document, second, he describes what pieces of data to extract, and third, he declares what target structure to use for storing the data extracted. A document is first retrieved from the Web according to one or more retrieval rules. Once retrieved, it is fed to an HTML parser that constructs a parsing tree following the Document Object Model (DOM) [55]. Following, the users can write extraction rules for locating data into the parsing tree. The extracted data is stored using the W4F internal format, called NSL (Nested String List). Finally, NSL structures can be exported to upper-level applications, according to specific mapping rules. The language used by W4F to define extraction rules is called HEL (HTML Extraction Language). An extraction rule is an assignment between a variable name and a path-expression. W4F offers a wizard to assist the user in writing extraction rules that are applied to tree nodes to extract data. For a given Web document, the user is presented with the same document annotated with additional information. The user clicks on the pieces of information of interest and the wizard returns a corresponding extraction rule. The wizard cannot deal with collection of items, so if the user is interested in various items of the same type of that one clicked on, conditions must be attached to the path expression to write robust extraction rules.

XWRAP – Another important HTML-aware tool for semi-automatic construction of wrappers is XWRAP [25]. The tool features a component library that provides basic building blocks for wrappers, and a user-friendly interface to ease the task of wrapper development. Before accomplishing the extraction process, the tool “cleans up” bad HTML tags and syntactical errors and turns the document into a parsing tree. The tool operates by leading the user through a number of steps, selecting in each step proper components of its library. At the end, XWRAP outputs a wrapper (coded in Java) for a specific source. In the object extraction step, the tool deploys a pre-defined set of data extraction heuristics tailored for HTML pages. The user may try one of six heuristics available to locate data objects of interest. If the user is satisfied with the extraction results, the extraction process may go on. The user can also refine the extraction by restricting or relaxing the number of components per object or by specifying data types for the elements. When the extraction result is satisfactory, the user may enter a tag name for each of the elements extracted and proceed to the wrapper code generation step.

RoadRunner – A recent tool that further explores the inherent features of HTML documents to automatically generate wrappers is RoadRunner [11]. It works by comparing the HTML structure of two (or more) given sample pages belonging to a same “page class”, generating as a result a schema for the data contained in the pages. From this schema, a grammar is inferred which is capable of recognizing instances of the attributes identified for this schema in the sample pages (or in pages of the same “class”). To accurately capture all possible structural variations occurring on pages of a same page class, it is possible to provide more than two sample pages. All the extraction process is based on an algorithm that compares the tag structure of the sample pages and generates regular expressions that handle structural mismatches found between the two structures. In this way, the algorithm discovers structural features such as tuples, lists, and variations. It should be noted that the process is fully automatic and no user intervention is requested, a feature that is unique to RoadRunner.

3.3 NLP-based Tools

1In this paper, we cover only the XWRAP Elite version, which is available for use at the URL http://www.cc.gatech.edu/projects/disl/XWRAPElite/
3.4 Wrapper Induction Tools

WIEN – A pioneer wrapper induction tool is WIEN [22], which takes as input a set of pages where data of interest is labeled to serve as examples, and returns, as a result, a wrapper that is consistent with each labeled page. The pages are assumed to have a pre-defined structure and specific induction heuristics are used to generate specific wrappers. For instance, if the pages have an HLT structure (i.e., pages have a head, a body containing flat tuples of data delineated by a left and a right component to be extracted, and then a tail), an HLT wrapper is generated. Wrappers generated by WIEN do not deal with nested structures or with variations typical of semistructured data.

SoftMealy – Similar to WIEN, SoftMealy [19] is a wrapper induction tool that generates extraction rules expressed using a special kind of automata called finite-state transducers (FST). An FST consists of input/output alphabets, states, and edges. To deal with structural variations, each state of the FST may have multiple outgoing edges. Before extracting data from a document, the wrapper segments an input HTML string into tokens, then the algorithm tries to induce extraction rules based on the context formed by the separators (tokens) of adjacent attributes present on given training examples. The resulting FST takes a sequence of tokens as input and matches the context separators with contextual rules to determine state transitions. An FST can be constructed for one tuple type. If there can be many types of tuples in a document, an FST can be built for each type.

STALKER – The wrapper induction techniques used in WIEN and SoftMealy are further developed in STALKER [29], a tool that can deal with hierarchical data extraction. The inputs to STALKER are (1) a set of training examples in the form of a sequence of tokens representing the surrounding of the data to be extracted; (2) a description of the pages structure, called an Embedded Catalog Tree (ECT). STALKER generates an extraction rule that covers as many as possible of the given examples. While uncovered examples exist, it generates a new disjunctive rule. When all positive examples are covered, STALKER returns the solution, that consists of a set of disjunctive rules. Using the ECT, STALKER can deal with nesting hierarchical objects.

3.5 Modeling-based Tools

NoDoSE (Northwestern Document Structure Extractor) – NoDoSE [3] is an interactive tool for semi-automatically determining the structure of documents that contain semistructured information and then extracting their data. Using a graphical user interface, the user hierarchically decomposes the document, outlining its interesting regions and describing their semantics. The decomposition process of a document occurs in levels. For each level of decomposition, the user builds an object with a complex structure, and then decomposes it in other objects with a more simple structure. After the user has “taught” the tool how to construct some objects, he can let NoDoSE to learn how to identify other objects in the document. This is accomplished by a mining component that attempts to infer the grammar of the document from objects constructed by the user. In its current version, NoDoSE features mining components for plain text and for HTML pages.

DEByE (Data Extraction By Example) – DEByE [23, 31] is an interactive tool that receives as input a set of example objects taken from a sample Web page and generates extraction patterns that allow extracting new objects from other similar pages (e.g., pages from a same Web Site). DEByE features a GUI that allows the user to assemble nested tables (with possible variations in structure) using pieces of data taken from the sample page. The tables assembled are examples of the objects to be identified on the target pages. From these examples, DEByE generates object extraction patterns (OEP) that indicate the structure and the textual surroundings of the objects to be extracted. These OEP are then fed to an bottom-up extraction algorithm that takes a target page as input, identifies on it atomic values in this page, and assembles complex objects using the structure of
3.6 Ontology-based Tools

This approach is mainly represented by the work of the Data Extraction Group [12] at Brigham Young University (BYU). In their tool, ontologies are previously constructed to describe the data of interest, including relationships, lexical appearance, and context keywords. By parsing this ontology, the tool can automatically produce a database by recognizing and extracting data present in documents or pages given as input. Prior to the application of the ontology, the tool requires the application of an automatic procedure to extract chunks of text containing data “items” (or records) of interest [13].

To work properly, this tool requires a careful construction of an ontology, a task that must be done manually by an expert in the domain of the ontology. On the positive side, if the ontology is representative enough, the extraction is fully automated. Furthermore, wrappers generated according to such an approach are inherently resilient (i.e., they continue to work properly even if the formatting features of the source pages change) and adaptable (i.e., they work for pages from many distinct sources belonging to the same application domain). Indeed, these features are unique to this approach. For convenience, in the remainder of the paper we will refer to this tool as the BYU tool.

As another example of an ontology-based tool for data extraction, we could cite Xtract [1], a tool for extracting data from floristic morphological description. However, as this tool applies only to a very specific domain, we will not discuss it further in this paper.

4. QUALITATIVE ANALYSIS

In this section, we analyze how the studied tools support some features that we regard as most important for data extraction. We address the following features: degree of automation, support for complex objects, page contents, availability of a graphical user interface, XML output, support for non-HTML sources, resilience and adaptiveness.

4.1 Degree of Automation

A very important feature of any data extraction tool is its degree of automation. This is related to the amount of work left to the user during the process of generating a wrapper for extracting Web data.

Regarding the degree of automation, the approaches based on languages for wrapper generation still require the writing of code, but provide some features, not available in general purpose languages, that ease this task. In tools such as Minerva, TSMIS, and Web-OQL, the user must examine the document and find the HTML tags that separate the objects of interest, and then write a program to separate the object regions. In other words, the process of discovering object boundaries is carried out manually.

HTML-aware tools usually provide a higher degree of automation. However, for this automation to be really effective, there must be a very consistent use of HTML tags in the target page. Unfortunately, this is not true for a great portion of Web pages available. In XWRAP, for example, the component library has a number of predefined heuristics to deal with several types of structuring HTML markups (e.g., tables, lists, etc.). By applying such heuristics and asking for feedback from the user, the tool can extract data very efficiently from certain type of pages. W4F uses the HTML Extraction Language (HEL) to define extraction rules. It features an extraction-wizard which can return a canonical path expressions for a piece of information selected by the user. As the wizard cannot deal with collection of items, the user who is interested in various items of the same type must manually write extraction rules that generalize the path expressions provided by the wizard. That is, the extraction process is semi-automatic. RoadRunner, as previously discussed, is fully automatic. In particular, the extraction procedure assumes that the target pages were generated from some data source (e.g., a database). Then, several heuristics are used to “reconstruct” the schema of such data source from the HTML tag hierarchy of the sample pages. This exempts users from supplying a target schema as well as examples of the data to be extracted.

The tools based on NLP, wrapper induction, and modeling, are said to be semi-automatic, because the user has only to provide examples that guide the generation of the wrapper.

As already discussed, the BYU tool requires the construction of an ontology to work properly, what should be done manually by an expert in the corresponding domain. After this, if the ontology is representative enough, the extraction is fully automated and can be used for other data sources in the same domain. Indeed, this feature is unique of such an approach. However, the ontology construction usually requires substantial effort for being validated.

As for the other tools studied, at the best of our knowledge, none one of them provides XML output.

4.2 Support for Objects with Complex Structure

Most of the data available on the Web implicitly presents a complex structure. Typically, this structure is loose, presenting degrees of variation typical of semi-structured data [2]. Further, in many situations, Web data is organized in hierarchies with multiple nesting levels. Thus, wrapper generation tools are expected to deal with such complex objects properly.

The exception mechanism and the Editor language featured in Minerva make it suitable to deal with the variations normally found on Web data. They are used to properly restructure the data of interest, whenever a production of the grammar fails. To represent complex objects, TSMIS adopts the Object Exchange Model (OEM) [30]. OEM is a flexible model very suitable for representing semi-structured data. Data represented in OEM constitute a graph, with a unique root object at the top and zero or more nested sub-objects. Web-OQL is capable of querying pages with irregular structure. The language, as well as the object model based on hypetrees (ordered arc-labeled trees) adopted by it, allows the representation of objects with varying structure and nested levels.
The language used by W4F to define extraction rules, HEL, provides some operators that allow constructing objects with complex structures. For instance, by using the fork operator [32], the user can group together in a single structure data that appear in several places. This operator can be used in cascade making it possible to build complex and irregular structures. In all of these cases, it is possible to handle complex objects by writing extraction code to deal with them. XWRAP, on the other hand, can only deal with nesting and variation if they are explicitly defined in the HTML formatting of the source page. It is able to determine the nesting hierarchy of the source page, by identifying top-level HTML structures (e.g., sections, tables) that form the page and internal structures (e.g., columns, rows, subsections). A similar approach is adopted by RoadRunner, that is based on the notion of nested types, which allows representing arbitrarily nested structures composed of lists and tuples.

In SoftMealy, the wrapper is represented as a FST where each state may have multiple outgoing edges. This allows the representation of structural variations in the code of the generated wrapper, making it capable of handling structural variations. SoftMealy, however, does not deal with nested structure. STALKER is more expressive than SoftMealy in this regard, since it uses an Embedded Catalog Tree formalism to describe the structure of the data contained in Web pages. This formalism represents the structure of the target page as a tree, where the internal nodes represent complex objects that can be decomposed and the external nodes (leaves) represent atomic data items to be extracted. This makes it able of dealing with nested structures. Structural variations are handled by generating disjunctive rules from the training examples provided by the user.

NoDoSE maintains a tree that maps the structural elements of the document to the text of the file. Each node of the tree represents one of the structural components of the document such as an element of a list or a field in a record. In DEByE, the underlying data model [23, 24] extends the usual notion of nested tables by allowing the representation of variations inside inner levels. Although such a model is not as powerful as XML or OEM, it is expressive enough to represent data presenting hierarchical structure and structural variations.

RAPIER, SRV, WHISK, and WIEN support neither nesting objects nor objects with structural variations.

4.3 Page Contents

With respect to page contents, there are basically two kinds of pages which wrapper generation tools apply to: those containing semistructured data and those containing semistructured text. To illustrate, consider the pages in Figures 1 and 2, which are examples of pages containing semistructured data and semistructured text, respectively. While pages of the first type feature data items (e.g., names of author, titles of papers, etc.) implicitly formatted to be recognized individually, pages of the second type bring free text from which data items can only be inferred.

Languages for wrapper development (Minerva and Web-OQL), HTML-aware tools (W4F, XWRAP, and RoadRunner), wrapper induction tools (WIEN, SoftMealy, and STALKER), and modeling-based tools (NoDoSE and DEByE) usually rely on delimiters surrounding data of interest to generate extraction rules. Thus they work better with pages of the first type.

Tools based on Natural Language Processing techniques, such as RAPIER, SRV, and WHISK, are generally more suitable to pages of the second type (e.g., job listings, apartment rental advertisement, etc.), but require that pages containing free text to be annotated by a syntactic analyzer and semantic tagger.

As the BYU tool rely on the presence of recognizable constants and keywords present in the target page, it can be applied to both types of pages. Indeed, the authors of this tool present experimental results that corroborate this [12]. Notice, however, that the accuracy of the wrapper generated, for both types of page, depends on how representative is the ontology for the domain to which the pages belong.

4.4 Availability of a Graphical User Interfaces

To help the user developing wrappers for Web data, some tools present a graphical interface (GUI) aiming at making this task easier. HTML-aware tools, NLP-based tools, wrapper induction tools, and modeling-based tools usually present a GUI. On the other hand, languages for wrapper development require the user to execute all the process manually. In the BYU tool, the ontology creation process must also be done manually by the user.

All NLP-based tools as well as the wrapper induction ones feature a GUI for the user to specify examples. In general, they allow the user to select pieces of data and to label these pieces of data properly to compose the examples.

In the case of the modeling-based tools, their GUI constitutes a crucial component in the whole extraction process. In NoDoSE, the user interacts with the GUI to select and decompose regions of interest in a page and also to associate with each region a proper structure (e.g., tuples, lists, etc.). The GUI also allows the user to test the generated wrapper against other pages and revise the previously generated extraction rule when needed. In DEByE, the user provides examples by assembling nested tables in such a way that each row of the outermost table corresponds to a distinct example. For this, the GUI provides several operations to build nested tables (e.g., column insertion and deletion, nesting and unnesting, etc.). The GUI also features an extraction feedback mechanism that allows users to select objects imperfectly extracted and build new examples from them, thus improving the extraction performance.

W4F offers some “wizards” to assist the user in the task of wrapper generation. For helping in the writing of extraction rules for a target page, the user can select pieces of data of interest and the extraction wizard returns a corresponding extraction rule in HEL. This rule can then be edited and modified to cover pieces of data similar to the ones initially
selected. In the case of XWRAP, the whole extraction process is guided by a GUI. It leads the user through a number of steps, implicitly selecting in each step a proper component of the library. At the end, XWRAP outputs a wrapper (coded in Java) for a specific source.

A graphical tool (ONTOS) is also provided by the BYU tool for helping the user in the process of editing an ontology.

The extraction process as performed by RoadRunner does not require user intervention, thus it need not a user interface.

4.5 XML Output

XML [6] is becoming the most important standard for data representation and exchange on the Web. Due to this fact, we consider an important feature of a data extraction tool whether it provides output in XML. In this section, we discuss the way how some of the analyzed tools provide output in XML.

In Minerva, the user has to explicitly write code to generate an output in XML. To perform this task, the user must refine the format of the extracted objects with appropriate language statements. In W4F, there is a “mapping wizard” that helps the user to create mapping rules to output the extract data in XML. XWRAP and DEByE natively provide output in XML. NoDoSE supports a variety of formats to output the data extracted from a document, among them XML and OEM.

As for the other tools studied, at the best of our knowledge, none of them provides XML output.

4.6 Support for Non-HTML Sources

A vast quantity of semistructured data stored in electronic form is not present in HTML pages, but in text files, such as e-mail messages, program code and documentation, configuration files, system logs, etc. Therefore it is very important that the data extraction tools might be able to handle such data sources.

The NLP-based tools and the BYU tool are specially suitable for non-HTML sources, since they do not depend on any kind of markup to work.

The wrapper induction tools and the modeling-based tools can be used to extract data from some non-HTML sources. These tools do not rely uniquely on HTML tags, so they are able to perform data extraction from other kinds of documents presenting some form of markup, be it implicit or explicit. The same can be said about Minerva and T3MMIS, where a skilled user can code extraction rules based on any existing markup.

On the other hand, as Web-OQL, W4F, XWRAP, and Road-
4. CONCLUSIONS

In this paper we presented a short survey of existing tools for generating a new wrapper for a Web page. We also discussed how these tools can be used in conjunction with other tools such as W3C, W3C and STARKON, which are also based on examples given by the users.

5. REFERENCES

The work described in this paper was partially supported by Project STAKON (NICT/CNP/PRONEX) grant number 0003000). and by the authors' individual grants from CNPq and CAPES.

ACKNOWLEDGMENTS

This paper is a revised version of the paper presented at the 3rd International Conference on Electronic Commerce and Web Technologies (ECWeb'99) held in Kobe, Japan, April 1999. The authors would like to thank all the participants for their valuable comments and suggestions.

6. CONCLUSIONS

In this paper we presented a short survey of existing tools for generating a new wrapper for a Web page. We also discussed how these tools can be used in conjunction with other tools such as W3C, W3C and STARKON, which are also based on examples given by the users.

5. REFERENCES

The work described in this paper was partially supported by Project STAKON (NICT/CNP/PRONEX) grant number 0003000), and by the authors' individual grants from CNPq and CAPES.

ACKNOWLEDGMENTS

This paper is a revised version of the paper presented at the 3rd International Conference on Electronic Commerce and Web Technologies (ECWeb'99) held in Kobe, Japan, April 1999. The authors would like to thank all the participants for their valuable comments and suggestions.
Table 1: Summary of the Qualitative Analysis

<table>
<thead>
<tr>
<th>Tools</th>
<th>Degree of Automation</th>
<th>Support for Complex Objects</th>
<th>GUI</th>
<th>XML Output</th>
<th>Support for Non-HTML Sources</th>
<th>Type of Page Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Languages</td>
<td>Minerva</td>
<td>Manual</td>
<td>Coding</td>
<td>No</td>
<td>Yes</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>TSIMMS</td>
<td>Manual</td>
<td>Coding</td>
<td>No</td>
<td>No</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>Web-DQL</td>
<td>Manual</td>
<td>Coding</td>
<td>No</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>HTML-aware</td>
<td>WIF</td>
<td>Semi-Automatic</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>XWRAP</td>
<td>Automatic</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>RoadRunner</td>
<td>Automatic</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None</td>
</tr>
<tr>
<td>NLP-based</td>
<td>WHISK</td>
<td>Semi-Automatic</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>RAPIER</td>
<td>Semi-Automatic</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>SRV</td>
<td>Semi-Automatic</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Full</td>
</tr>
<tr>
<td>Induction</td>
<td>W3RN</td>
<td>Semi-Automatic</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>SoftMealy</td>
<td>Semi-Automatic</td>
<td>Partial</td>
<td>Yes</td>
<td>No</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>STARKER</td>
<td>Semi-Automatic</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Partial</td>
</tr>
<tr>
<td>Modeling-based</td>
<td>NoJUSE</td>
<td>Semi-Automatic</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>UEBYSE</td>
<td>Semi-Automatic</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Partial</td>
</tr>
<tr>
<td>Ontology-based</td>
<td>BYU</td>
<td>Manual</td>
<td>Coding</td>
<td>Yes</td>
<td>No</td>
<td>Full</td>
</tr>
</tbody>
</table>

Figure 3: Graphical perspective of the Qualitative Analysis


