Toolkits for Generating Wrappers

– A Survey of Software Toolkits for Automated Data Extraction from Websites –

Stefan Kuhlins and Ross Tredwell

University of Mannheim
Department of Information Systems III
D-68131 Mannheim
Germany
{kuhlins, tredwell}@uni-mannheim.de

Abstract. Various web applications in e-business, such as online price comparisons, competition monitoring and personalised newsletters require retrieval of distributed information from the Internet. This paper examines the suitability of software toolkits for the extraction of data from websites. The term *wrapper* is defined and an overview of presently available toolkits for generating wrappers is provided. In order to give a better insight into the workings of such toolkits, a detailed analysis of the non-commercial software program LAPIS is presented. An example application using this toolkit demonstrates how acceptable results can be achieved with relative ease. The functionality of the program is compared with the functionality of the commercial toolkit *RoboMaker* and the differences are highlighted. With the aim of providing improved ease-of-use and faster wrapper generation in mind, possible areas for further development of toolkits for automated web data extraction are discussed.

1 Introduction

The *World Wide Web* (WWW) has become one of the most important connections to various sources of information. The infrastructure of the WWW was originally developed for use by humans [10]. A large proportion of the data is embedded in *Hypertext Markup Language* (HTML) documents. This language serves the visual presentation of data in Internet browsers, but does not provide semantic information for the data presented [3]. This form of data presentation is, therefore, inappropriate for the demands of any automated, computer assisted information management system. In particular, if data from different sources needs to be combined, it is necessary to develop special and often complex programs to automate the data extraction [8]. In a large variety of scenarios, software routines (so-called *wrappers* [1, 10, 21]) for automated data extraction from various Internet sources are a necessity. For example, considering the vast number of online shops already offering their goods and services on the Internet, it is virtually impossible to manually retrieve all of the information necessary for a comprehensive and up-to-date price comparison. Automatic online monitoring of competitors is another potential e-business application ideally suited to the use of wrappers. For instance, it is a tedious task to manually check competitors' websites for any changes in product prices on a regular basis. Wrappers can help to automate such procedures.

As the manual coding of wrappers is a time-consuming and error-prone process, different methods have been proposed to automate the wrapper generation process. In recent work, efforts have been made to categorise toolkits according to the methods they apply [12]. We take a different approach that is more application oriented and aim to demonstrate the extent to which toolkits for generating wrappers can simplify and speed up the development of software applications for data extraction from heterogeneous Internet sources.

This paper presents an overview of toolkits for the generation of wrappers. Firstly, the meaning of the term *wrapper* and its role in the field of information management is explained. In the following section, the basic attributes of wrapper-generating toolkits are described. Thereafter, a number of currently available toolkits and their main features are presented. Special and unusual attributes are briefly highlighted and explained. Section 4 provides a closer examination of LAPIS (*Lightweight Architecture for Processing Information Structure*), a non-commercial toolkit. Details of the most important features of this application are presented and its simple and easy to learn scripting language is described. The ability to integrate external applications into this toolkit is highlighted. Additionally, the functionality of this program is demonstrated using a meta price comparison as a basic example of the wide range of possible applications for such toolkits. Section 5 highlights the strengths and weaknesses of the LAPIS toolkit in comparison with *RoboMaker*, a commercial wrapper-generating program. The paper concludes with an evaluation of the suitability of currently available toolkits for generating wrappers and a discussion of potential areas for further development.

2 Wrapper Definition

Wrappers are specialised program routines that automatically extract data from Internet websites and convert the information into a structured format. More specifically, wrappers have three main functions. Firstly, they must be able to download HTML pages from a website. Secondly, search for, recognise and extract specified data. Thirdly, save this data in a suitably structured format to enable further manipulation [6]. The data can then be imported into other applications for additional processing. According to [20], over 80% of the published information on the WWW is based on databases running in the background. When compiling this data into HTML documents the structure of the underlying databases is completely lost. Wrappers try to reverse this process by restoring the information to a structured format [21]. With the right programs, it is even possible to use the WWW as a large database. By using several wrappers to extract data from the various information sources of the WWW, the retrieved data can be made available in an appropriately structured format [4]. As a rule, a specially developed wrapper is required for each individual data source, because of the different and unique structures of websites. The WWW is also extremely dynamic and continually evolving, which results in frequent changes in the structures of websites. Consequently, it is often necessary to constantly update or even completely rewrite existing wrappers, in order to maintain the desired data extraction capabilities [1].

The *Extensible Markup Language* (XML) has the potential to alleviate such problems. Whereas HTML is presentation oriented, XML keeps the data structure separate from the presentation. However, it may take some time before all data is provided in the XML format, and it remains to be seen whether XML can establish itself in all areas of electronic information processing [11]. Taking into consideration that XML documents are based on varying *Document Type Definitions* (DTD) or *XML-Schemas*, the current problems regarding data extraction from HTML documents can be reduced, but not completely resolved. Wrappers will, therefore, retain an important role in the integration of data from WWW sources for some time to come.

3 Wrapper-Generating Toolkits

Every wrapper can be manually developed from scratch, for example, in an established programming language using *regular expressions*. For smaller applications, this can prove to be a sensible approach. However, if the use of a larger number of wrappers is required, this inevitably leads to the use of so-called *toolkits*, which can generate a complete wrapper based on user defined parameters for a given data source. One of the most important features of generated wrappers is the format in which the extracted data can be exported. If, for example, the extracted data is converted into an XML format, then it can be imported and processed by a large number of software applications.

Toolkits for generating wrappers can be differentiated in a number of ways. They can be categorised by their output methods, interface type, Web crawling capability, use of a graphical user interface (GUI) and several other characteristics. *Laender et al.* [12] categorise a number of toolkits based on the methods used for generating wrappers. These methods include specially designed wrapper development languages and algorithms based on HTML-awareness, induction, modelling, ontology and natural language processing. However, a detailed presentation of such technical details is beyond the scope of this survey paper. Therefore, the toolkits are simply divided into two basic categories based on commercial and non-commercial availability.

The wrapper generating programs within both of these categories offer several different means of user interaction. Some toolkits are solely based on command lines and require routines developed in a pre-determined unique scripting language, in order to generate an appropriate wrapper for a specified data source. These wrapper development scripting languages are used in standard text editors and can be seen as application specific alternatives to general-purpose languages such as Perl and Java. A large number of toolkits offer a GUI, whereby the relevant data within an HTML document is highlighted with a mouse, and the program then generates a wrapper based on the specified information. Several toolkits combine both of the features described above. Initially, the relevant data is highlighted with a mouse and the program generates a wrapper from this input. If the automatically generated result does not meet the specified requirements, the user has the additional possibility of implementing changes via an editor integrated within the toolkit. Whether frequent corrections are necessary or not depends, largely, on the underlying algorithms and the functional maturity of the toolkit.

The tables below provide, in alphabetical order, an extensive overview of currently available toolkits for generating wrappers (a more detailed overview is available online, also covering toolkits that are no longer accessible [9]). Table 1 lists non-commercial programs and tables 2 and 3 display commercial programs. This section concludes by emphasizing and briefly explaining some of the more sophisticated features of the various programs.

Table 1. Overview of Non-Commercial Toolkits

Toolkit	Output Data	Java API	Open Source	Source Code	Web Crawling	GUI	Editor	Scripting Language	Tool Support
Araneus	XML, Text	✓	only API	Java	—	—	—	EDITOR	—
BYU	Text, Tabular	✓	—	Java	—	—	—	ontologies	—
COMMIX	HTML, XML, Text	No explicit API	—	Java	✓	✓	—	—	(✓) Chinese toolkit
DEByE	XML, Text	No explicit API	—	Java	✓	✓	—	—	✓
Info Extractor	DOM type structure	✓ JDBC	—	Java	—	—	—	reg. expr., grammar rules	—
Jedi	XML, Text	✓	—	Java	✓	—	—	JEDI	—
LAPIS	HTML, XHTML, XML, Text	✓	✓	Java	✓	✓	✓	Text Constraints	✓
NoDoSE	XML, Tabular, ODL, OEM	✓	✓	Java	—	✓	—	—	—
Road Runner	XML, HTML, Text	✓	—	Java	✓	—	—	regular expressions	Toolkit not yet available
Scout	HTML, Text	✓	✓	Java	—	—	—	regular expressions	—
SoftMealy	HTML, Text	✓	✓	Java	✓	✓	—	—	—
TSIMMIS	HTML, Text	—	✓	C/C++	—	—	—	—	—
WebL	XML, HTML, Text	✓	✓	Java	✓	—	(✓) third party	WebL	(✓) News-groups
Web Sphinx	HTML, Text	No explicit API	✓	Java	✓	✓	✓	ORO Matcher reg. expr.	✓
WIEN	HTML, Text	No explicit API	✓	Java	—	✓	—	—	—
XWrap	XML (incl. DTD)	✓	(✓) only Wrapper	Java	✓	✓	—	—	✓

Table 2. Overview of Commercial Toolkits

Company	Toolkit	Demo Version	Output Data	Web Crawling	Interface for External Programs	GUI	Editor	Scripting Language
Caesius Software	WebQL (Web Query Language)	—	Text, HTML	✓	✓	✓	✓	WebQL
Connotate Technologies	vTag	—	XML	✓	✓	✓	—	—
Crystal Software	TextPipe	✓	XML, Text	—	✓	✓	✓	Perl, VBScript, JScript, Rexx, Python
Data Junction	Content Extractor	✓	XML, Text	✓	✓	✓	✓	CXL
Extradata Technologies	Unwwwrap	✓	HTML, Text, Table	(✓) based on "Bookmarks"	Plug-In for MS Internet Explorer	✓	—	—
Fetch Technologies	Agent Builder	—	XML, Text	✓	ODBC Database	✓	—	—
Fire Spout	ETL Engine	✓	HTML, XML, Text	✓	✓	✓	—	—
firstRain	firstRain Studio	—	HTML, XML, Text	✓	✓	✓	—	—
IBM	Intelligent Miner	✓	HTML, Text	✓	✓	✓	—	—
ItemField	ParserStudio	—	XML	✓	✓	✓	✓	Parser Script Language
Kapow Technologies	RoboSuite	✓	HTML, XML, Text, CSV, SQL Database	✓	XML/SOAP, Import of Java Beans	✓	✓	reg. expr., Text Expressions
Knowmadic	KM Studio	—	XML, Visual Basic Object	✓	✓	✓	—	—
Lencom Software	Visual Web Task	✓	HTML, Text	✓	✓	✓	—	—
Lixto	Lixto Wrapper Toolkit, InfoPipes	—	HTML, XML, Text, WML	✓	✓ Java API	✓	✓	ELOG
Loton Tech	WebDataKit	✓	Table	✓	✓ Java API	✓	✓	SQL for HTML and XML
NQL Inc.	Network Query Language	✓	HTML, XML, Text	✓	✓	✓	✓	Network Query Language
Orsus Solution	UnoStudio	—	HTML, XML, WAP, Text	(✓) only prede-fined URLs	✓	✓	—	—

Table 3. Overview of Commercial Toolkits (continued)

Company	Toolkit	Demo Version	Output Data	Web Crawling	Interface for External Programs	GUI	Editor	Scripting Language
Republica	X-Fetch Wrapper	✓	XML, WML, XHTML, ...	—	✓ Java API.	✓	✓	DEL
Temis Group	Online Miner, Insight Discoverer Extractor	—	HTML, XML, Text	✓	✓	✓	—	—
Thunderstone	Webinator	✓	HTML, Text	✓	✓	✓	✓	Texis Webscript, reg. expr.
Tropea Inc.	W4F Wrapper Factory (not available anymore)	—	HTML, XML, Text	✓	✓ Java API	✓	✓	HEL
WhizBang!	WhizBang! Extraction Framework	—	HTML, Text	✓	✓	✓	—	—
Wisosoft	Info Scanner	✓	XML	(✓) User predefined Macro	HTTP, COM, Java API, ODBC	✓	✓	VBScript, JScript, reg. expr.

Many of the non-commercial toolkits are purely command-line based and offer user assistance through predefined modules. These assist, for example, in establishing a network connection or the completion and submission of HTML forms. The user merely has to modify them to suit his specific requirements. The toolkits provide relatively simple scripting languages for creating extraction rules. However, due to the rapid pace of development of modern programming languages and the existence of very extensive, freely available libraries with predefined classes, the use of these toolkits hardly speeds up or simplifies the generation of wrappers. This also explains the lack of further development of these non-graphical programs.

The freely available toolkits stem, almost without exception, from university development projects or from other academic research institutions. WebL is the only toolkit that is provided by a commercial company free of charge. Whilst, for the most part, the projects have been completed, except for DEByE (*Data Extraction by Example*), LAPIS, *Road Runner* and *XWrap*, there is a lack of technical backup and further development of the programs. As several of these toolkits do not provide currently popular output formats, they are limited in their use and often require specially implemented interfaces for the interaction with external applications.

XWrap is an exception in the category of non-commercial toolkits. Firstly, due to licensing regulations, it can only be used online and just provides the Java source code of the generated wrappers for use by local applications. Secondly, the output format of the extracted data is in XML [14]. DEByE is also able to export the extracted data in XML. Using only a small number of user provided data examples, DEByE tries to automatically recognize similarly structured data within the HTML document [13]. Section 4 provides a more detailed examination of the LAPIS toolkit, which is currently still being supported and enhanced.

Almost all commercial toolkits provide the ability to output the extracted data in XML. All commercial toolkits, without exception, include a GUI and are mouse driven. One of the first commercial toolkits, W4F from *Tropea Inc.*, uses its own scripting language and only offers so-called "Wizards" (small graphical interfaces) as an additional support. The program's robust extraction rules are based on path definitions within the document structure of an HTML page. These path definitions lead to the relevant data locations [21]. However, W4F was unable to establish itself in the market and is no longer available.

More sophisticated toolkits, such as the ones offered by *Caesius Software*, *Crystal Software*, *Fetch Technologies*, *Item Field*, *Kapow Technologies*, *Lixto* and *Wisosoft* do not require the highlighting of all the data that is to be extracted. A small number of examples are sufficient for the programs to generate suitable extraction rules. *Lixto*, for example, can be operated by users without any HTML programming knowledge, as it does not require the user to work with the HTML source code [2]. Thus, the user can obtain a fully functional wrapper quickly and efficiently. *Fetch Technologies* goes one step further and generates wrappers that are capable of coping with structural changes of websites, in as much as they check their own functionality and automatically implement corrections if necessary [5]. A similarly comprehensive toolkit, *RoboSuite* from *Kapow Technologies*, is discussed in Section 5.

The stability and reliability of wrappers is highly dependent on the data extraction methods that the toolkit applies. For example, toolkits that only rely on HTML structures to identify relevant data are very vulnerable to the slightest website changes and frequent repairs to the wrappers may be necessary. Method combinations provide greater robustness, such as the combination of HTML path structures and pattern recognition methods. The reliability of a generated wrapper does not necessarily depend on the way a user interacts with the toolkit. If, for instance, the toolkit generates wrappers based on pattern or natural language recognition, the highlighting of example data in a browser can lead to very robust wrappers, without the user being forced to work with the HTML source code. This is particularly true if the user is not completely dependant on the automatically applied statistical methods of the toolkit and is able to adjust and fine-tune the resulting extraction rules.

Most toolkits can generate wrappers that have single page retrieval capabilities. Some have the ability to automatically crawl web pages or follow a restricted number of links. This, for example, enables the wrappers to retrieve all pages in a search request, even if the desired data is contained in several HTML documents. In general, the ability to automatically trace links is used to retrieve all requested data contained in web pages from one particular website and is not necessarily intended for the implementation of web robots that also follow links to unknown websites. This survey focuses primarily on the data extraction phase and only briefly highlights the document retrieval phase.

4 Wrapper-Generation with LAPIS

Following the overview provided above, we now take a closer look at one of the non-commercial toolkits to give a better insight into one of several possible approaches for

generating wrappers. This toolkit was chosen as an example, because, at the time this research began, it was the only non-commercial, open source toolkit available that was still being developed and supported. As Miller explains in [15], the syntax of regular expressions and grammars for automatic text processing is difficult to read and understand. Therefore, these methods have not gained general acceptance with the majority of normal users. For this reason, it became necessary to develop programs that provided intuitive procedures for automatic text manipulation. As a result of this research, the LAPIS toolkit was developed, opening up a range of possibilities for text manipulation. HTML pages are nothing other than simple text documents, enabling LAPIS to extract data and ultimately be used as a toolkit for generating wrappers.

4.1 LAPIS Program Features

LAPIS is essentially a web browser with an integrated text editor that can be programmed to process and manipulate text documents loaded within the program [17]. The individual manual manipulation steps are contained in an executable script in a proprietary scripting language. To start the automatic data manipulation process, the script created by LAPIS is simply activated within the toolkit. Used as a web browser, LAPIS is able to download all files that have a valid and unrestricted *Uniform Resource Locator* (URL) using the *File Transfer Protocol* (FTP) or the *Hypertext Transfer Protocol* (HTTP). HTML pages can either be displayed as a text document showing the complete source code, or the page can be presented as a standard web page, as in every regular web browser [16]. LAPIS is equipped with the pattern recognition language "*text constraints*", which allows the user to precisely specify and restrict the required data in a given document. In comparison with the often cryptic and difficult to read *regular expressions*, this scripting language has a syntax that is extremely intuitive and easy to understand.

With the aid of additional features incorporated within the program, it is possible to manipulate previously highlighted regions within a document. The data can be deleted, filtered, sorted or extracted [16]. There are several possibilities for defining relevant data in a document. The user can manually apply the *text constraints* language to define a specific pattern for all the required data, or the user can highlight only a few relevant text regions and LAPIS will try to automatically generate a suitable pattern [18]. For complex HTML documents a combination of both methods has proved successful. Firstly, LAPIS tries to generate a suitable pattern, which the user can then manually adapt if necessary. The toolkit provides further assistance by testing for mistakes in developed patterns. LAPIS achieves this by separately highlighting the text areas that lie within the valid definition of a given pattern, but differ in their structure from those regions that most commonly occur [19]. Furthermore, LAPIS has a built-in *command shell*, which allows the program to process Tcl-script commands. This feature enables the invocation of external command-line programs from within LAPIS. It is possible, for example, to transfer extracted data directly to an external user program for further manipulation [17].

4.2 Text Constraints

The development of wrappers with LAPIS is based on the definition of text regions within a text document. The patterns inferred by the highlighting of the relevant text or data are defined using the implemented *text constraints* language. The syntax of this scripting language can be learned quickly and is easy to read and understand. This is due to the fact that simple English terms are used and the actual functionality of the language is hidden in the background, so that the user does not have to concern himself with complex programming procedures. The following example demonstrates the region highlighting effect of the simple *text constraints* pattern `Number.ScientificNotation just before " Euro"` on an HTML extract of a price comparison web page for a specific digital camera (see Fig. 1). The comparison itself is based on extracted data from several online shops.

```
<TD>Sony Cybershot DSC-P 50<BR>Incl. Memory Stick</TD>
<TD align=right><B>429.99 Euro</B></TD>
<TD align=middle>Primustronix.de</TD>
                    ...
<TD>Sony DSC-P50 PC/MAC<BR>2.1 Megapixel, 3x Zoom</TD>
<TD align=right><B>452.55 Euro</B></TD>
<TD align=middle>avitos.com</TD>
```

Fig. 1. Text Constraints region highlighting

Various parsers are already installed in LAPIS. Amongst others, an HTML parser with which one can directly access elements in an HTML document. It is also possible to specify restrictions within the *text constraints* patterns. Thus, the relevant data regions can be restricted by using terms such as `contains`, `first`, `last`, `just before` and `after`. For example, the first column of the first table in an HTML page would be highlighted using the pattern `text in first [td]`. In this way new patterns can be developed and added to the LAPIS library with a user defined name [19]. It makes no difference to LAPIS whether parsers, *regular expressions*, highlighting with a mouse or a combination of all three methods are used for determining relevant text regions. All that is needed to extract or manipulate data is a valid *text constraints* pattern.

4.3 Incorporating External Programs

Utilising Tcl-commands, it is also possible to store extracted data in a file or to invoke external command-line programs. The output of any external command-line program can be displayed in the LAPIS browser window and then be processed by the toolkit. It is also possible to integrate LAPIS into the program routines of other applications. This enables the specification of certain processes to be allocated to the toolkit and allows the results to be processed by the main program [17]. As an example, LAPIS can download various HTML documents from the WWW, then extract data using previously defined *text constraints* patterns and finally invoke a further application, which reads in the extracted data and stores it in a database.

4.4 Automatic Data Extraction

Once a suitable pattern has been created for an HTML document using *text constraints* and the individual text or data manipulation steps have been defined, it is then necessary to combine these steps into an automatic procedure. LAPIS offers an optional tool for this step. Similar to a cassette-recorder, LAPIS can record the individual steps of a user whilst the data processing and manipulation is being performed in order to generate an executable script. A script of this sort can contain, among other things, URL addresses, instructions on completing and submitting HTML forms, *text constraints* patterns, external program invocations or text manipulation commands. By invoking such a script, the manipulation of one or more documents can be automatically transacted by LAPIS, which processes the instructions contained in the script sequentially [17].

4.5 Application Example

To demonstrate the possible application of LAPIS for the generation of wrappers, an example in the form of a meta price comparison is presented in this section. A meta price comparison extracts price information for a specific product from various online price comparers and combines the information into a new price comparison. The procedure is identical to that of a meta search engine. To this end, the result pages of three different online price comparers are downloaded, the relevant data is extracted, then combined again in a newly created and sorted HTML table and finally presented in a standard Internet browser. The patterns or wrappers for the individual result pages must first be created using *text constraints* and are subsequently stored in the LAPIS parser library (see Fig. 2).

```
prefix Wrapper
    #
    # WebDeShopping
    #

    prefix WebDeShopping
        PriceWebDe is Business.Number.ScientificNotation just before " Euro"
        ProductWebDe is Text in [td] just before [td]
                                                contains PriceWebDe
        ShopWebDe is Link contains "Haendler"
                      in row contains PriceWebDe
        WrapperWebDe is either PriceWebDe
                              or ProductWebDe
                              or ShopWebDe
```

Fig. 2. Wrapper for the price comparer WEB.DE

The script for the automatic processing is then developed using LAPIS (see Fig. 3). Each of the result pages of the online price comparers is downloaded into the LAPIS browser window. The previously developed wrappers for each price comparer are then applied and the extracted data is provisionally stored in separate temporary files. To further process the extracted data, a specific unique symbol (“ # ”) is used to distinguish the individual data sets in the temporary files.

```
doc -type text
[http://shopping.web.de/Suche/Fotografie+%26+Optik/Fotografie/Digitale
+Fotografie/Digitalkameras/?pricef=&pricet=&sort=&search=sony+dsc-
p50&id=-L0Sm10**sga700]
extract -separatedby " # " {Wrapper.WebDeShopping.WrapperWebDe}
save C:/LapisFiles/ExtractedData/WebDe_1.ref
...
doc -type html [java -cp c:/LapisFiles/JavaFiles/ Mediator]
sort row -by {4th cell in row} -order numeric
browse
```

Fig. 3. Excerpt of the LAPIS script used to automate the extraction process

When all the result pages have been processed the so-called *Mediator* [22], in this case an external Java program, is invoked by LAPIS. The *Mediator* has the task of combining all the extracted results into a table in a new HTML document (see Fig. 4). This table is returned to LAPIS via the standard output of the command-line program. The built-in HTML table-sorting feature enables LAPIS to rearrange the table rows in price order. This drastically reduces the time-consuming conversion of prices into a suitable data type for sorting within the *Mediator*. It is a good example of how efficiently various programs can be integrated into the data extraction process depending on which features are required. Finally, LAPIS invokes a standard Internet browser and transfers the resulting HTML document to it. By clicking on the links contained in the results page, the user can navigate to the price comparers or, in some cases, straight to the online shops with the most attractive offers.

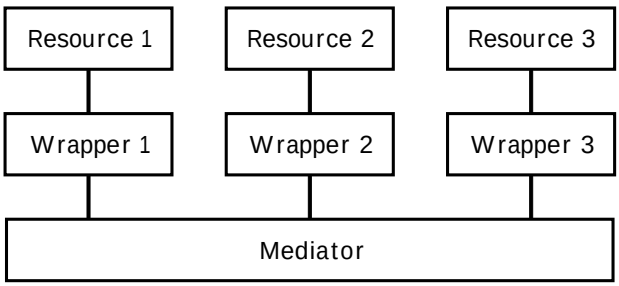


Fig. 4. Wrapper-Mediator Interaction

Only a short time is required to understand and comprehend the *text constraints* pattern language and to become familiar with the way LAPIS generates wrappers. Quick and acceptable results can be achieved using only a few lines of scripting code. With the possibility of integrating external programs into the data extraction process, LAPIS is able to efficiently utilise the strengths of these programs. This leads to an increase in overall performance and flexibility, which consequently makes LAPIS an extremely versatile toolkit.

5 Comparison of LAPIS and a Commercial Toolkit

LAPIS was originally developed to automate the editing and manipulation of text documents. With the use and combination of several of the implemented features, it is also possible to generate wrappers for HTML documents. It does, however, lack certain features that are already a standard implementation in commercial toolkits for generating wrappers. This includes, for instance, the processing of JavaScript commands, which are used in a large number of HTML documents nowadays. Due to the absence of this feature, these types of HTML documents are either incompletely or inadequately presented in the LAPIS browser window. Consequently, the user is often forced to work with the HTML source code in the LAPIS text-editing window, in order to generate suitable *text constraints* patterns. Moreover, the processing of *cookies* is not supported, which is often an essential requirement for downloading certain web pages. In addition, it is almost impossible for a user not possessing some basic programming skills to correctly integrate the developed wrappers and automation scripts into the LAPIS library.

In the following paragraphs, closer attention is paid to one of the commercial toolkits. This will demonstrate which features a modern wrapper-generating toolkit should provide. The *RoboMaker* toolkit, from *Kapow Technologies' RoboSuite*, acts as a very good example. The toolkit belongs to the category of programs that not only offer a GUI, but also provide a built-in editor. Functionality that is missing in LAPIS, such as the processing of *cookies* or the administration of generated wrappers, is already implemented and easy to use in *RoboMaker*.

The development of the wrappers occurs in several simple steps. Firstly, the procedures that lead to the relevant HTML pages are defined. In this respect, the toolkit offers standard features for automatically completing and submitting HTML forms, enabling, for instance, the use of different search terms in a search form. With further simple parameter definitions, the automatic tracing of links is also easy to master.

The next step is to highlight the structured data. Here, the extraction rules are based on the page structure of the *Document Object Model* (DOM) [23] and *regular expressions* [7]. This combination generates very robust wrappers, which do not need to be constantly updated when the structure of a website changes slightly. By highlighting the relevant data on example web pages using a mouse, these robust features can be applied by a user with very little HTML programming experience. The resulting complex extraction rules are automatically generated in the background by the toolkit. Should the user not be completely satisfied with the generated patterns, he still has the possibility of making manual corrections to the extraction rules. This is achieved by applying a specified scripting language using the provided editing window. It only takes a short time to become familiar with the scripting language. *RoboMaker* offers a full range of output formats. In addition to the structured XML output, the toolkit can provide extracted data for direct transfer into a SQL database.

A further extremely useful feature of *RoboMaker* is the fully integrated visual *debugger*. This enables the precise examination of a generated wrapper by running each individual step of the wrapper program and logging the results. By using this module, it is not only possible to find errors at their source, but it is also relatively easy to trace what is happening in each individual step within a wrapper program. This allows a rapid identification of the causes of an incorrect output.

RoboSuite contains an extra module called *RoboManager*. This module enables all generated robots or wrappers to be administrated in a well-structured GUI. Therefore, it is easy to manage several hundred wrappers at one time and, should a wrapper no longer function properly, corrections can be effortlessly undertaken at any time. If, for example, network problems arise that result in data sources being temporarily inaccessible, it is very simple to deactivate certain wrappers with *RoboManager* for a specified period of time. Even predefined times for the activity of wrappers can be set up using this module. This is, for instance, very useful when relevant data on a specific website is only changed on a monthly basis. In order to extract updated data, the wrapper is automatically activated once a month and after it has extracted the requested information, it is deactivated again. As the wrapper is not continuously checking for changes within a website, the Internet traffic is kept to a minimum. To reduce the local resources required by *RoboSuite*, it is possible to activate the generated wrappers from the command line using a module devoid of a GUI (*RoboRunner*).

6 Conclusions

It can be concluded that, in comparison to manual programming, the use of wrapper-generating toolkits saves time in the development of wrappers for extracting data from HTML resources.

Toolkits based purely on command lines can generate robust wrappers, but are not easy to use and have little chance of being generally accepted. The possible applications of the other non-commercial toolkits are also limited due to the restricted number of features, as highlighted. Only two of the currently developed non-commercial toolkits are in a position to provide extracted data in an XML format. However, as is demonstrated using LAPIS, it is possible to achieve acceptable results with very little effort. If only a limited number of wrappers are required, non-commercial toolkits are a good alternative to manual programming from scratch.

The commercial toolkits provide the necessary and most advanced features for the professional development of a larger number of wrappers. Focusing on ease-of-use, the most important features are the simple administration of the generated wrappers as well as the support of current output formats. As with the non-commercial toolkits, the robustness of the wrappers with respect to structural changes of a website are highly dependant on the extraction rules employed. If a wrapper depends too heavily on a standardised document structure within a website and relies mainly on the DOM without combining it with other methods, then even the slightest structural changes can cause the wrapper to malfunction.

Several methods and approaches for the generation of Internet data extraction wrappers exist. This overview paper describes the methods used in two particular toolkits in more detail. The reader is referred to the references section for more detailed descriptions of the methods employed by other toolkits, as an in-depth review of these is beyond the scope of this paper.

Extracting data from XML documents is far easier than from HTML files, due to the separation of content and presentation. However, the conversion of all HTML documents into XML involves a tremendous amount of time and effort. Additionally,

it is not always in the interest of website owners if their data is extracted and used for third party purposes, especially taking into account the widespread commercial use of advertising banners that would be bypassed by these extraction procedures. It is, therefore, questionable whether all HTML pages will be replaced by XML documents in the near future.

Nowadays, a large amount of data is processed by web programming languages and is not contained within plain HTML code. Hypertext links, for example, are often dynamically generated by JavaScript code. Most of the current toolkits are unable to automatically overcome this type of hurdle and require user assistance in order to correctly navigate websites. Solving this problem would lead to major usability improvements for the toolkits.

Current and future advances in the field of *artificial intelligence* could also provide further opportunities for the development of toolkits. For example, the possibility of incorporating knowledge discovery applications and automated classification tools could greatly enhance the reliability and possibilities of automation for data extraction wrappers. Scenarios in which the user simply defines specific criteria for the desired information and then dispatches intelligent agents to wrap previously unknown websites and extract the requested data are becoming feasible and are no longer pure fiction. By these means, nearly all of the products and prices of various online shops could be collected to provide a fully automatic worldwide price comparison using relatively little development effort. Should the currently generated wrappers evolve into more sophisticated extraction agents, then more complex types of applications could be realised. For example, it would be possible to make research documents from various Internet sources searchable, whereby the user only defines a topic he is interested in and agents automatically collect *and* effectively summarise all of the available online information on the requested subject. This would drastically reduce the time currently spent on manual academic research.

7 Acknowledgements

The authors are grateful to Robert Baumgartner for helpful comments and his research support. Moreover, we thank three anonymous reviewers for their valuable suggestions on an earlier draft of this paper. Acknowledgements go to all software companies and non-commercial toolkit developers that have provided evaluation versions of their toolkits and offered additional information and support. Particular thanks go to *Kapow Technologies* for generously supporting this research by supplying a free, temporarily fully licensed version of their *RoboSuite* toolkit and to Rob Miller for developing and providing the open source LAPIS toolkit.

8 References

1. Adelberg, B. and Denny, M.: Building Robust Wrappers for Text Sources, Technical Report 1999, http://www.cs.northwestern.edu/~adelberg/papers/www.pdf (Aug. 2002)

2. Baumgartner, R., Flesca, S. and Gottlob, G.: Visual Web Information Extraction with Lixto, Paper for the 27th International Conference on Very Large Data Bases (VLDB 2001), Rome, Italy, September 2001
3. Doorenbos, R., Etzioni, O. and Weld, S.: A Scalable Comparison-Shopping Agent for the World-Wide Web, Paper for the First International Conference on Autonomous Agents, February 1997, http://www.cs.washington.edu/homes/weld/papers/shopbot.pdf (Aug. 2002)
4. Eikvil, L.: Information Extraction from World Wide Web – A Survey, Report No. 945, ISBN 82-539-0429-0, July 1999
5. Fetch Technologies: Technology Overview – Reliably Extracting Web Data, White Paper, November 2001, http://www.fetch.com/whitepapers/FetchWhitePaper.doc (Aug. 2002)
6. Golgher, P., Laender, A., Silva, A. and Ribeiro-Neto, B.: An Example-Based Environment for Wrapper Generation, in: Proceedings of the 2nd International Workshop on The World Wide Web and Conceptual Modeling, pp. 152–164, Salt Lake City, Utah, USA, 2000
7. Kapow Technologies: RoboSuite Technical White Paper, November 2001, http://www.kapowtech.com/robosuitetechnicalwhitepaper.pdf (Aug. 2002)
8. Knoblock, C., Minton, S., Ambite, J., Ashish, N., Muslea, I., Philpot, A. and Tejada, S.: The ARIADNE Approach to Web-Based Information Integration, 2000, in: International Journal of Cooperative Information Systems 10(1-2): pp. 145–169, 2001
9. Kuhlins, S. and Tredwell, R.: Wrapper-Generating Toolkits, Online Overview, available since December 2001: http://www.wifo.uni-mannheim.de/~kuhlins/wrappertools/index.html
10. Kushmerick, N.: Wrapper Induction for Information Extraction, Dissertation 1997, Dept of Computer Science & Engineering, Univ. of Washington, Tech. Report UW-CSE-97-11-04, http://www.cs.ucd.ie/staff/nick/home/research/download/kushmerick-phd.ps (Aug. 2002)
11. Kushmerick, N.: Wrapper Verification, World Wide Web Journal 3(2): pp. 79–94, 2000
12. Laender, A., Ribeiro-Neto, B., Silva, A. and Teixeira, J.: A Brief Survey of Web Data Extraction Tools, in: SIGMOD Record, Volume 31, Number 2, June 2002
13. Laender, A., Ribeiro-Neto, B., Silva, A. and Silva, E.: Representing Web Data as Complex Objects, in: Proceedings of the First International Conference on Electronic Commerce and Web Technologies (EC-Web 2000), pp. 216–228, Greenwich, UK, 2000
14. Liu, L. Pu, C. and Han, W.: XWrap – An XML-enabled Wrapper Construction System for Web Information Sources, Proceedings of the 16th International Conference on Data Engineering (ICDE'2000), San Diego CA, 2000
15. Miller, R.: Lightweight Structured Text Processing, PhD Thesis Proposal, Computer Science Department, Carnegie Mellon University, USA, April 1999, http://www-2.cs.cmu.edu/~rcm/papers/proposal/proposal.html (Aug. 2002)
16. Miller, R. and Myers, B.: Lightweight Structured Text Processing, in: Proceedings of 1999 USENIX Annual Technical Conference, Monterey, CA, pp. 131–144, June 1999
17. Miller, R. and Myers, B.: Integrating a Command Shell Into a Web Browser, in: Proceedings of USENIX 2000 Annual Technical Conference, San Diego, pp. 171–182, June 2000
18. Miller, R. and Myers, B.: Outlier Finding: Focusing User Attention on Possible Errors, in: Proceedings of UIST 2001, Orlando, FL, pp. 81–90, November 2001
19. Miller, R. and Myers, A.: Multiple Selections in Smart Text Editing, in: Proceedings of IUI 2002, San Francisco, CA, pp. 103–110, January 2002
20. Sahuguet, A. and Azavant, F.: Web Ecology – Recycling HTML pages as XML documents using W4F, in: ACM International Workshop on the Web and Databases (WebDB'99), Philadelphia, Pennsylvania, USA, June 1999
21. Sahuguet, A. and Azavant, F.: Building Intelligent Web Applications Using Lightweight Wrappers, Paper, July 2000, http://db.cis.upenn.edu/DL/dke.pdf (Aug. 2002)
22. Wiederhold, G.: Mediators in the Architecture of Future Information Systems, IEEE Computer 25 (3), pp. 38-49, 1992
23. World Wide Web Consortium: The Document Object Model, http://www.w3.org/DOM