Automatized Information Retrieval from Heterogenous Web Sources*

Peter Sýkora, Andrej Janžo, Vojtech Szöcs, and György Frivolt

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 3, 842 16 Bratislava, Slovakia
TeamProject10@yahoogroups.com

Abstract. Intelligent information retrieval using automated tools is essential for extracting and keeping track of the changing contents on the Web. In this paper, we describe a web page wrapping system gathering contents of semi-structured web pages as its input and generating structured output of extracted data. Extracted data, due to its structured nature, can be thereafter easily deposited into XML file, relational database or ontology storage. We have designed a wrapping language for this purpose, which represents a compromise between simplicity and utilizable functionality. The wrapping program has tree-like structure, with actions chained together forming a continuous control flow. We also describe a GUI for effective wrapper creation and verification. Finally, typical wrapper behavior is demonstrated on a concrete example.

1 Automatized information retrieval: state of the art, problems and perspectives

It is an undisputed fact that the Web as we know it today offers nearly unlimited amount of information related to various topics. Information on the Web is exchanged in the form of web pages, which should follow a widely accepted *HyperText Markup Language (HTML)* format of structured data description and presentation. Unfortunately, many web page designers do not structure their HTML code properly or frequently change the page design and layout, making the task of information retrieval via HTML code more difficult and error-prone. Over years, the problem of finding specific and accurate information on the Web has evolved into a separate kind of problem aimed at automatized and intelligent information retrieval. This kind of information retrieval is also known as web wrapping and concerns methods, techniques and tools, which support programmable and automatized web navigation, data extraction and its deposition in various target environments (XML files, relational database, ontology storage). We also emphasize that this kind of information retrieval is only a part of

* This work was partially supported by the Slovak State Programme of Research and Development "Establishing of Information Society" under the contract No. 1025/04.

many other possible information retrieval techniques (e.g. IR via semantic web or using data mining techniques).

Generally speaking, there are two main types of web page wrappers: *static* and *adaptive* wrappers. Static wrappers are programmable and adjustable wrapping units, which commonly have a very close relationship to the web source they use as their input. In fact, they are programmed specifically for each web source and thus achieve excellent wrapping efficiency, but only as long as the web page structure remains unchanged. This static nature is thus also a drawback of static wrappers, meaning the need of their re-programming in the event of web source structural re-design (adding or removing banners, mixing of contents and presentation, etc.). However, static wrappers still dominate the area of commercial web page wrapping systems, mostly due to their simplicity, directness and a relatively good reliability of their results. There are many ways to express static wrappers, e.g. using logic programming as in *Elog* language of project Lixto [1]. The predicates of Elog programs operate on subtrees of the main HTML document tree. The programs are visualized using a tree consisting of patterns and filters [2].

Adaptive wrappers represent a higher level of abstraction in the context of wrapper program description. Adaptive wrappers are described more generally (e.g. using various patterns) and are therefore more usable for different web sources with similar structure, in comparison to static wrappers. The higher adaptivity is often reached by building on different representation of the HTML document. Instead of representation of the document as a DOM tree [3] visual properties of the document are represented. That representation builds on the fact that although web portals providing similar service may generate different type of HTML code, the visual look of the web page reflects the look the users got used to, so it should be present on different web pages. For wrapper creation various techniques, such as machine learning, are applied.

The main aspects concerning web page wrapping process are following: *navigation* in the web hyperspace, with respect to the common client-server communication issues such as cookie and session handling, user authentication via HTML forms, etc.; *extraction* of desired data from web pages using various techniques, e.g. structured data query languages like XPath or string data query languages like regular expressions; *deposition* of extracted data into various target environments, with an effort to retain the structured nature of extracted information; *verification* of wrapper program in order to evaluate its functionality and effectiveness for given web source; *integration* of the results of various wrappers for heterogeneous web sources.

This paper presents results and conclusions from a development of static wrapper system, which was verified via extraction of numerous job opportunities in terms of cooperation with the NAZOU research and development project [4]. The main objective of this research and development project is to design and verify by pilot applications new ways of information and knowledge processing in heterogeneous environment, in particular acquisition, representation, organization and maintenance of actual knowledge, and to develop tools for supporting new model of heterogeneous environment. Our tool stands at the first line of processing and aims to fill the knowledge repository by data gathered from the Web. We support other tools by collecting and annotating job offers published on other job offer sites. Principles of our static wrapping system are also described in more details in [5].

2 Wrapper program representation and interpretation

Main components of an internal wrapper program representation are actions, which operate on the context. Context thus reflects the actual state of a single wrapper program, consisting of many aspects (extracted documents, output objects, variables, cookies etc.). *Actions* are elementary executive units (nodes), which are connected together into a tree structure. The tree root is the wrapper program's first action and defines the starting point of its execution. Connections between individual actions thereafter define the control flow of the wrapper program.

Actions are divided into these four main categories:

- *Navigation actions* are used for navigating on the Web and parsing web documents into DOM structure such as loading pages, following hyperlinks, and other.
- *Extraction actions* are used for data extraction using XPath [6] and regular expressions into structured output objects or context variables.
- *Iterative actions* are used for execution of a given subtree multiple times, e.g., action `ForEachTag` executes its subtree for each document node matching given XPath and/or regular expression criteria.
- *Auxiliary actions* are used for branching and sending extracted output objects to the output module.

Context contains the current state of a wrapper program, including temporary output data. It consists of documents, cookies and authentication data, output objects and variables. *Documents* are stored in an associative array of DOM documents. Each document is identified by its unique name and represents parsed web page dowloaded by navigation actions. *Cookies and HTTP authentication data* are gathered by navigation actions or directly set by the user. They are used for communication with the web servers, from which given web pages are downloaded.

Output Objects are structured containers of data presented to the user as the result of the information extraction. Output objects are filled with data using extraction actions and delivered to output by `WriteObject` action. Their structure is DOM-specific and is created by the user during extraction with simplified XPath. Example of ouput object in Figure 1 is output object named `JobOffer`, that contains job offer. Paths to filled text nodes are `@date` (attribute of main node), `Description`, `Description/Company`, and `Description/Salary`.

Variables are designed to help creating more versatile wrapper programs and to enable external inputs. Using variables, the user can parametrize the

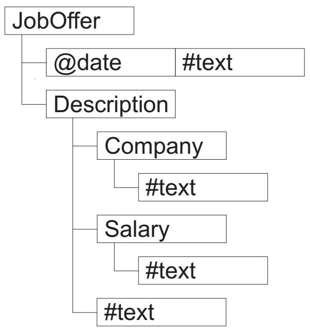

Fig. 1. Example of JobOffer output object

interpretation of the wrapper code. For instance, external inputs can be used in the parameter giving the URL of `LoadPage` action:

`http://searchengine.com/?q=${Product}`

There are two kinds of variables: *internal variables* and *external variables.* Internal variables evolve during the wrapper program execution, when a data extraction action (`ExtractData`) is programmed to place extracted data into a context as variable for later usage. This usually comprises data, which is not known at the start of a wrapper program and not relevant to output, but needed in some phases of the wrapping process. External variables are variables explicitly given in a wrapper program representation (XML) and reflect more general parameters vital for the whole wrapping process.

Extracted data deposition into different storages is done by a specific set of writers, which implement a single generic output writer interface `OutputWriter`. Their main purpose is transformation of generic, domain-independent output objects to a form suitable for deposition in a given target environment. Implementation of XML writer is simply done by serializing DOM structured output objects to XML. In the case of ontology data storages, output objects are transformed to RDF/XML code using templates: RDF/XML template has custom parameters, which are substituted with concrete values from output objects. Experiments were also performed in the field of ontology-object mapping, when a writer filled given domain objects and submitted them to object-to-ontology mapper for their deposition.

Before interpretation, the wrapper program must be initially loaded from an external representation - XML file describing the wrapper. XML representation contains actions with parameter values, list of mutual interconnections and internal variables. Program is then iterpreted by executing the first action (root of the action-tree structure). Type of action defines its behavior and the way how the direct succesors will be executed. Simple control flow can be shown on a navigation action of type `LoadPage`, which executes its direct successor after loading a page and exits. Iteration action `ForEachTag`, on the other hand, executes its succesor for each found node in the defined document.

During wrapper program execution various errors may occur (f.i., desired data field is not present on a web page and data extraction action fails). To avoid halting the whole wrapping process, one must be able to configure actions in a way that they can respond to possible errors. This error handling concept is realized by `errorHandler` parameter, which defines action's behavior after catching an error. Default error handler stops wrapping with an error result, other ignores the error and the last implemented error handler runs external command. Using the lastly mentioned error handler and some email sending script, it is possible to perform a simple email notification when a wrapper error occurs.

Wrapper program is designed and interpreted in way that allows simple debugging. Before the action's code execution, a breakpoint method is called which invokes a debugger, if any is set for the given wrapper program. Debugger recieves the current state of context as a parameter, which can be thereafter presented to the user. The whole debugging process is thus based on action stepping, meaning one can pause the wrapper program execution only between individual actions.

3 Graphical User Interface

The graphical user interface (GUI) offers the support for end-users during the visual wrapper editing phase and during the processes of wrapper validation and debugging. The necessity of effective and convenient GUI for the wrapping system has been evident from the early time, because XML representation can be complex and overwhelming for new users, as every tag in XML representation can have several different attributes with different meaning and usage constraints. GUI was also designed to offer tools for validation and debugging of wrappers, which is essential for creating efficient and robust wrappers.

Editing the wrapper code. GUI offers an XML source code editor, where the wrapper code can be easy edited as in a typical XML editor with syntax highlighting. However, main focus was oriented towards a visual editor, which contains a set of defined actions (called Action Palette) and allows to create transition between those actions by creating and editing visual tree of actions. On the background of visual editor, the final XML file as wrapper program's external representation is created. Every action has defined set of attributes, together with some restrictions. This makes the process of creating wrapper code easier and prevents users from making faults. All changes in the wrapper's XML source code are automatically reflected into the visual representation. The goal is to make process of wrapper creation as simple and efficient as possible.

Extensibility. The entire architecture of our static wrapping system is designed to be very extensible and customizable. This also means that GUI must be able to reflect possible new actions, which can be easily designed and implemented using existing action model. This is done through special XML representation of each action. Each action is represented by a separate XML file, where attributes, their types and allowed values are described. Thanks to a simple structuring and easy meaningful construction, these XML descriptions of actions can be easily written in the future for each new action and integrated into the GUI.

Interpretation and debugging of the wrapper code. The next step after finishing the wrapper program design is to ensure that XML code is valid against defined XML Schema and fulfills all the restrictions. Although wrapper code can be valid against all the restrictions, it does not mean that it will work correctly and give desired outputs. It would be really hard for users to find the wrong part of code, eventually a bad XPath selection query or incorrect regular expression. For this reason, we have decided to enrich GUI with the possibility of interpretation and debugging designed wrapper code.

User can debug the wrapper during its execution in a step by step manner, action after action. The user can see input and output context of the last executed action in every step and can also modify it. With these four debugging techniques, the user should be able to filter possible mistakes out. By accurate and proper identification of wrapper problems, the user should be able to create a working wrapper more conveniently and in shorter time.

4 Example

While our wrapper program was experimentally verified by retrieval of information regarding job opportunities from job portals, our example will concern the *eurojobs.com* portal: European interactive job portal, that accumulates job vacancies from employers and CVs from candidates and offers it for interested persons. Portal makes possible to search in offers by defining queries on locality, sector and key words. All offers contain job title, posted date, company, contract type, industrial sector, job locality, salary and job description.

The root element of XML tree is named `Wrapper`. The `Wrapper` has child elements `Actions`, `Transitions` and `Variables`, which divide tree into three sections. First section contains actions and its attributes, next section contains transitions between actions and last section contains variables. Portal *eurojobs.com*'s XML tree is depicted on fig. 2.

The actions are ordered in the same sequence as they are executed. The first action is `LoadPage`, action that loads the first page, which contains list with the first 30 job offers and a “next link” to next page with another list of offers. Action `DoWhileLinkExists` performs iterations through next links. Action executes at first all the following actions and then opens next page on which executes the following actions again. Action `ForEachTag` performs iterations through selected part of HTML tree. Processing every single job offer is in one iteration. The following `ExtractData` action extracts posted date and the `FollowLink` action opens detail page of the job offer. Next seven `ExtractData` actions extract from detail page job title, company name and next details. The last action is `WriteObject`, which writes extracted data to defined target data storage (XML, Ontology, Database).

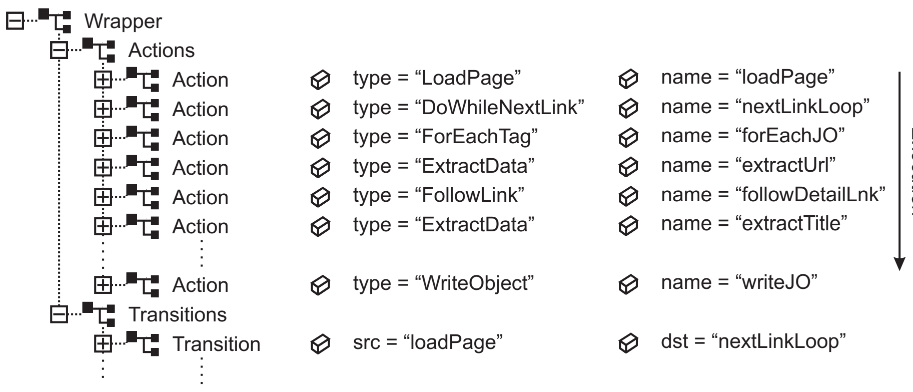
Wrapper
Actions
Action type = "LoadPage" name = "loadPage"
Action type = "DoWhileNextLink" name = "nextLinkLoop"
Action type = "ForEachTag" name = "forEachJO"
Action type = "ExtractData" name = "extractUrl"
Action type = "FollowLink" name = "followDetailLnk"
Action type = "ExtractData" name = "extractTitle"
Action type = "WriteObject" name = "writeJO"
Transitions
Transition src = "loadPage" dst = "nextLinkLoop"
Execution

Fig. 2. Eurojobs.com wrapper XML tree

Every action contains attributes and parameters. Necessary attributes are name and type of the action. Action parameters are stored in child elements of the `Action` element and in `Attribute` elements, that contains attribute name, attribute type and value. For example, action `DoWhileLinkExists` contains parameters `InDocument` and `XPathExp` to identify next link on the page.

Launching the wrapper is also possible from the command line tool or from GUI. In both tools, debugging can take place, checking and verifying the results of each individual action.

5 Conclusion and future work

We have presented design of a static wrapping system. The current architecture enables further development of the system. We implemented framework for wrapping tasks, language for wrapper program definition and a GUI for designing the wrapper. Our wrapping system also features advanced debugging options like action holding, detailed error reporting and error handling for better control of the wrapper creation process. All core functions and the wrapper framework were already implemented and tested. This includes wrapper program creation via GUI and new action extensions for better wrapping problem description. The outputs can be currently extracted to XML and Sesame ontology format. However, the design of the system enables its extension to other outputs (database, RDF) as well. The outputing phase of the wrapping is independent from the extraction part of the wrapping, therefore the extraction can be simply changed to other output in any phase of the development.

The implemented tool was tested. We extracted 1937 job offers from three job portals [1].

Our aim is to extend the architecture to enable the possibility to wrap documents other than HTML (PDF, PostScript), design and implementation of more

[1] www.eurojobs.com, www.careerbuilder.com, www.linkedin.com

representations behalf the straightforward DOM representation. On the top of the different document representations, different patterns and filters has to be developed which should be combined regardless which representation they are applied on.

We plan to the describe wrapper programs by hierarchy of patterns. This development will consist of

- simplification of the context, which will become more and more specific - containing smaller and smaller parts of the wrapped document - as deeper parts of the action hierarchy are processed,
- uniting the interface of context variables and output objects - extraction action should not recognize the difference in extraction to inner or output variable,
- removal of the `WriteObject` action - the writing of the wrapped data to the target environment will be performed by output variables,
- the extraction actions will consist of several filters and one target variable.

The core task of the developer is to define filters. This task will be supported by visualisation of the filters and by induction of the filters by machine learning techniques. The creation of wrappers might be reduced to giving positive and negative examples what parts are wished or refused to be wrapper from the document.

Although the above described changes of the current architecture are radical, but doable and lead to a more extendable tool with extensible set of document representations and a better framework for wrapper pattern experiments.

Another very interesting direction of development is generic wrapping, which consists of automatic understanding of the wrapped document's structure. A generic wrapper after visiting of a site has to recognize the parts which are repeating, shortly it has to be able to extract the template of the document. For different document representations different template languages has to be developed requiring different generic wrapping approaches.

We believe that acquiring the template of a document or of a whole site should yield simpler wrapper design in the latter phase. Every position in the generated template should identify a filter. The designer would thus choose the parts of the generated template, exactly identifying the filter she wish to use in the pattern.

Clearly the outlined goals are broad. A well elaborated architecute need to be developed and implemented by close cooperation of the development team members. However, we believe that the described rough roadmap might yield a very effective wrapper designing tool.

Acknowledgements

This work was partially supported by the Slovak State Programme of Research and Development "Establishing of Information Society" under the contract No. 1025/04.

References

1. Baumgartner, R., Flesca, S., Gottlob, G.: The elog web extraction language. LPAR 2001 (2001)
2. Baumgartner, R., Flesca, S., Gottlob, G.: Visual web information extraction with lixto. The VLDB Journal (2001) 119–128
3. Consortium, W.: Document Object Model (DOM) Level 3 Core Specification (2004)
4. Návrat, P., Bieliková, M., Rozinajová, V.: Methods and tools for acquiring and presenting information and knowledge in the web. In Rachev, B., Smrikarov, A., eds.: CompSysTech 2005. (2005) IIIB.7.1–IIIB.7.6
5. Sýkora, P., Janžo, A., Jemala, P.K.M., Berta, I., Szöcs, V.: Automatized Information Retrieval from Heterogenous Web Sources. In Bieliková, M., ed.: IIT.SRC 2006: Student Research Conference, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava (2006) 137–144
6. Consortium, W.: XML Path Language (XPath) Version 1.0 (1999)