Generating Wrappers for Semi-Structured Web Pages

Ştefan Udrişţoiu*, Anca Ion*

**University of Craiova, Faculty of Control, Computers and Electronics,*
Software Engineering Department, Bvd. Decebal, Nr. 107, 200440, Craiova, Dolj, ROMANIA
(e-mail:sudristoiu@software.ucv.ro)

Abstract: This paper presents a tool suit used to improve the activity of wrapper development for a meta-search engine. When a new type of source of information is to be added in the system a concrete wrapper is to be written. The Wrappers' Editor is a visual tool that helps editing wrapper representation files. The wrapper representation is visualized as a tree and on the hard disk is stored as an xml file, using Wrapper Description Language (WDL). The tools consist from a compiler which generates Java classes, a specialized editor for the WDL, an interpreter used also as a debugger and as a platform for the last tool of the suite, which generates extraction rules starting from user annotations.

Keywords: Compilers, Finite automata, Formal specification, Information retrieval, Information integration, Problem-oriented languages, Rapid programming, Rule-based systems, Software productivity, Software tools.

1. INTRODUCTION

A web meta-search engine tries to make things uniform for the user with respect to a wide range from Web search engines and standardized Z39.50 databases to proprietary SQL servers or Integrated Library Systems. The end user formulates the query in a single format, irrespective of any data source or schemas, and obtains results in a uniform way, tailored to its needs.

When a new type of source of information is to be added in the system a concrete wrapper is to be written. This module deals with native issues regarding the source: both query and retrieval. In order to be inserted into the system it has to be wrapped in a module that handles the translation to and from uniform items to native items.

Implementing a wrapper can be complicated and time consuming, but some of the coding involved in wrappers can be automated. Hence, one important goal is to automatically or semi-automatically generate wrappers from high level descriptions of the information processing they need to do.

Using any automated process to develop and maintain wrappers will reduce the level of programming expertise that is required for the persons involved in these tasks and also will reduce the time needed to finalize them.

Another objective for an automated process is to eliminate the programming errors associated especially with monotonous development.

This paper describes a suite of tools developed to assist creation and maintenance of wrappers for HTTP sources. The tools were developed incrementally. In the first stage we developed a language, called WDL, flexible enough to represent all wrappers and to make easier the task of creating new wrappers. In the second stage a compiler was implemented, which transforms the WDL representation into a Java class. In the next step, a specialized editor was built to facilitate programming in the WDL. In this stage a good understanding of the HTTP protocol along with the capacity of interpreting the sniffer's output for the retrieved pages is still required as a level of expertise. The time-consuming task of identifying and parsing the elements will still be made manually by human analyzes. For debugging purposes, in the fourth stage we implemented an interpreter capable of executing accurately the wrappers in WDL form. The last stage consists in automating as much as possible the WDL creation and maintenance, including automated detection of faulty wrappers.

In the ideal case scenario, a representation of the wrapper in WDL is created via an interactive process supervised by a human designer and then the result is compiled into a Java module. The reason for this is that a real meta-search engine is built for speed, without any compromise regarding the recall and precision. The value for both those measures has to be 100%. The recall and precision are defined as in (Zhao *et al*., 1998):

$$recall = \frac{E_c}{N_t} \tag{1}$$

and

$$precision = \frac{E_c}{E_t} \tag{2}$$

where E_c is the total number of correctly extracted records, N_t is the total number of records and Et is the total number of records extracted.

This last step, compiling the wrapper to java form, is what distinguish this tools from other similar commercial

products, Kapow RoboSuite (Kapow, 2006), Lixto (Baumgartner *et al.*, 2001; Gottlob *et al.*, 2002) or non-commercial, LAPIS (Miler, 2002), Jedi (Huck *et al.*, 1998).

Is different even from XRAP (Liu *et al*., 2000) because our generated wrappers have the same aspect with those written by a human programmer and the performance of the XRAP system is unacceptable for an industrial strength application. For a comprehensive survey of the toolkits for generating wrappers see (Kuhlins and Tredwell, 2003; Laender *et al*., 2002).

The paper is structured as follows. We start by describing the parameters for the wrapper. In the next section we present the structure of the language. In the forth section we introduce the extraction rules used by the language constructs from the Section 5. In the Section 6 we present WDL instructions and in the Section 7, the compiler is detailed. The paper ends with the conclusions and future work.

2. CONFIGURATION PARAMETERS

The Java wrapper receives at startup a number of predefined parameters:

- start – represents the index of the first record that will be retrieved by the wrapper.
- perPage – the number of records that will be retrieved.
- query – this is the result of the XSLT translator which does the conversion from a common query model to the native source query.
- homeURL – this is usually the starting point for crawling.
- searchURL – this is usually the point where the actual search can start. It is used when the crawling can be improved by skipping some http requests.
- userName – this is the parameter used to fill the corresponding authentication field form. A similar purpose has userPassword and userPin.
- databaseName – a selector for multi-database search engines that can be used to designate more than one database.

Additional parameters can be loaded using specialized language constructs specifying the name of the parameter and the source.

3. LANGUAGE STRUCTURE

The WDL was designed with a dual purpose: to be used as a specialized wrapping programming language and to permit an intermediary representation on which to build other assisting tools.

The language is based on XML and has a structure inspired by the development process. When writing a new wrapper, a human programmer starts by downloading the first page, usually containing the authentication form. After downloading a page, she figures what information needs to extract in order to be able to send the query, select one or more specific databases, or a specific record format. That is the authentication and crawling part and could imply many http requests. For instance, to reach the page containing extracted records for the Biblioline site no less than 17 requests has to be made. So, the main language construct is described using a STATE element having a type attribute with value request. A request state may contain:

- A CONDITION that validates all the actions of the state.
- A HTTP_POST_REQUEST element node or a HTTP_GET_REQUEST element node describing, obviously, how to download a web page.
- A container for extractions.
- A container for markers, which will be described later.
- A WAIT_ONLY_FOR element. This is used for optimization, specifying that the download will stop after certain extractions are performed successfully. This is possible because the web page is red slice after slice, extractions being tried in between.
- A SAVE_PAGE_TO element, which contains the name of the variable assigned with the downloaded page.

Ideally, this should be enough but, often, different computations have to be made beside requests, which are contained by instruction states.

For debugging purposes, ASSERT nodes can be inserted in any place, containing logical expression that always must be true. Those conditions are used in detecting faulty wrappers in an automated fashion.

4. EXTRACTION RULES

We use three types of rules: string rules, regular expression rules and index rules. The rules can be mixed but the string rules are those used in the overwhelming majority of cases.

A string rule contains one or more strings to search for in the html page and has the following attributes:

- Direction – specifies in which direction to search for tags and can have, obviously, one of the values: FORWARD or BACKWARD. The starting point for the search is from where the previous rule has finished. The start point for the first rule is usually implicit but sometime can be specified, depending on extraction type.
- Action – specifies what to do with the search pointer after a tag was found. Can have one of the values: NOTHING, SKIP, INCREMENT or DECREMENT. Their meanings are: leave the search pointer at the beginning of the found tag, move the pointer at the end of the searched tag, move the pointer one character right or move the pointer one character left.
- Finds – indicates what the rules searches for: the beginning of the extracted region, the end or is an intermediary step. The defaults are that the last rule finds the end and the rule before that finds the start.
- Case – specifies if the string case is ignored or not.

- Type – if there are more than one tag to search for then it specifies which tag will be chosen: the first found, in the order that are added to the rule, the one with the minimum index or with the maximum index.

Regular expression rules have almost the same form, without the case attribute. There use is not recommended, the reason being again the cost, in computation time.

The last type, index rules, is used even more rarely than regular expression, usually to specify that the start or the end of the extracted region coincide with the start or the end of the searched area.

Other parameters that can be specified for a search, beside start position, are: the source for the search and the index limits. A rule is considered valid if it leaves the search pointer within specified limits. In majority of cases those parameters are implicit, for instance the source is considered to be currently downloading page.

At a superficial glance, one can say that the extraction is similar with STALKER (Muslea *et al.*, 2001; Muslea *et al*., 1998) but the structure of the WDL allows a lot more flexibility.

5. EXTRACTION CONSTRUCTS

A very specialized form of extraction is that used for extracting the estimate, i.e. an approximation for the total number of records. This extraction element contains usually only extraction rules, as described in the previous section, and very rarely an extraction source and an indicator to stop the wrapper if start parameter is greater than retrieved estimate.

Another type of extraction is for “singleton” information, which is named internally “extraction of variables”, used mainly for crawling. This type of extraction contains, besides extraction rules, a name and a type parameter specifying what kind of computation should be performed after extraction. Type can be one of:

- TEXT – the extracted area is cleared of HTML tags and the entities are replaced.
- NUMBER – the extracted area is converted into a number.
- URL – an URL is constructed using as a base the current URL.
- UNTOUCHED – the extracted area is kept unmodified. This is usually needed when the area extracted will be used as source for other extractions, leading to hierarchical structure.

Optionally, a parsing source can be specified, along with a re-extraction indicator and a node containing rejection rules. If rejection rules are applied successfully inside extracted area then the extraction is invalidated. The indicator specifies that if the extraction is unsuccessful then the corresponding variable will remain with the previous value. In absence of the indicator, assigning a null value to the corresponding variable signals an unsuccessful extraction.

Another specialized form of extraction is that used to extract hidden fields. The source for extraction is specified using a variable name or using extraction rules that will be applied in the currently downloading page. The result of the extraction will have a “query” format and it will be saved into a variable specified by the name parameter. Optionally, an encoding and an indicator to extract also fields without value can be added.

For facilitating description for crawling that has a tree like structure, a special construct was added, named REPEATED_EXTRACTION, which may contain:

- An optionally source, specified as a variable name or using extraction rules. If missing, the entire page is assumed.
- A set of rules describing a block extraction. The next block is extracted from the region starting after the end of the previous block.
- Extraction of variables, which are tried inside every block.
- A set of instructions that are executed for every block, after variables extraction.

This is needed, for instance when records are partitioned using various criteria and can be accessed through a page containing a link for every set.

The purpose of the wrapper is records extraction. Record extraction is defined by the following elements:

- An optionally source for all the records that can be specified as a variable name or using extraction rules. If missing, the entire page is assumed.
- RECORD_BOUNDARIES – extraction rules that define a region containing exactly one record. The rules are applied starting from the end of the last extracted record. This is based on “the assumption that there exist (invisible) disjoint rectangular regions such that each region contains the attributes for one unique tuple” (Irmak *et al.*, 2006), assumption confirmed by our experience.
- REJECTION_RULES – if these rules are successfully applied inside record boundaries then the record is considered invalid.
- Zero or more FIELD nodes – a field node may contain extraction rules, variables extractions and instructions. The later are used when the field is composed. Its type can be URL or TEXT. The label of the field is given by its name. All the extractions are applied inside the record boundaries.
- Zero or more EXTENDED_FIELD nodes – their structure is similar with that of the regular field but the source is the page downloaded using the FIELD extracted under the name url.

Extraction of tables having a variable number of columns is specified using TABLE_HEADERS and TABLE_FIELDS elements. Each of them contains an optionally source and rules which extract a single cell.

Table headers can be global for all records from page or can be specific for every record. The association between headers and fields is done automatically, based on their relative order.

The language has also a specialized construct used to extract fields that have a visual aspect similar with nested tables, but without headers.

For optimization purposes, when the source for the repeated extraction and for records extraction is not the entire downloading page, the area containing the desired information can be designated using two sets of rules, called START_RULES and END_RULES. The block and record boundaries extraction is valid if start rules can be successfully applied before the beginning of the block (record) and the end rules cannot be applied in the region between the start of the page and the end of the block (record). The source can be specified using only start rules or only end rules. Specifying a variable name as a source has a different semantic: start the extraction of blocks (records) only if the variable is successfully extracted and that decision cannot be made until the entire area containing the variable is downloaded.

6. INSTRUCTIONS

The first instruction presented is maybe the most controversial: GOTO a state name. We preferred this to describe the flow of wrapper's actions because was the best choice for a visual representation. But there was a price to pay when implementing the generator, which transforms WDL in java.

An instruction that is essential for the language expressiveness is CALL, which has a single parameter, a filename containing a WDL description. The call is transformed in java as a function call for which the input-output parameters are determined automatically.

For manipulating queries there are two instructions: REPLACE_IN_QUERY and GET_VALUE_FROM_ QUERY. First allows to change and the second to obtain the value for a certain parameter.

For cookies manipulation, the language possesses an instruction that clears all the cookies and an instruction that adds a specific cookie. The cookies are obtained automatically from http responses but sometimes the wrapper has to "forget" cookies or to add a cookie obtained in some other way.

For constructing a URL there is a SET_URL instruction and a SET instruction for assigning an expression to a variable.

The language has also a conditional instruction similar with if.

As an alternative for repeated extraction there is an instruction that allows simultaneous extraction of information from more than one source. After extracting one piece of information from each source, a set of instruction is executed. For each source, a start, a step and an end parameter can be used, specifying the index of the first extracted token, the index of next tokens and the index of the last tokens.

7. COMPILER

The compiler has two stages. In the first stage a rough Java representation is generated which is completed in the second stage with GOTO transformations, adding variable definitions and inserting protections against infinite loops.

The first stage labels code generated by each state in order to help transforming GOTOs in the next stage.

If the GOTO is a backward jump then it can define a DO WHILE or a CONTINUE inside a WHILE, depending of the nesting level at which was found GOTO. The level is determined for each GOTO in the first stage of the compiler. When is generated an opening bracket, "{", the level is incremented and when is generated a closing bracket, "}", the level is decremented. The level starts from zero. If the level is equal with zero then the generator tries to generate and DO WHILE. If the level is equal with one the generator tries first to generate a CONTINUE and if this is not successful then tries to generate a DO WHILE. If the level is higher than one then the compiler tries to generate a CONTINUE.

If the jump defined by the GOTO is forward then we have the following cases, depending again on the level value. If the level is zero then it can define only an ELSE block. If the level is equal with one then it can define a method call, a WHILE block, a BREAK, a CONTINUE, an IF or an ELSE. First it tries to generate a WHILE, next it tries o generate a CONTINUE inside a DO WHILE. If the previous attempts are not successful then it tries to generate a BREAK instruction then it tries first to detect if it is defining an ELSE construction and next a function call. If the level is greater than one then the options are CONTINUE inside a DO WHILE, a BREAK or a function call, in that order.

8. RECORDS EXTRACTION

This is a two step process. In the first step the records boundaries are established. The user will select the first record and will indicate the total number of records from the page.

The wizard will try to determine the rest of the records and will visualize the result. The user can correct the wizard by invalidating a supposed record or by selecting other records, until the visually selected records are correct. In the second step the field extractions will be generated after the user will label the fields from one or more records. Again, the process can be repeated until the correct result will be obtained.

Some special cases are: table fields, having record specific headers or global headers, loop fields and fields that are obtained by concatenating different pieces of information.

The main effort so far was devoted to record extraction and text field extraction. The current implementation

covers around 80% for records boundaries and 50% for text field extraction.

The records boundaries are found using the following algorithm: first a defining tag is searched, which can be found at the start, at the end or in the middle of the records, then the remaining boundaries are searched for starting from those defining tags.

A defining tag for records is a tag that can be found only once for every record and not in the rest of the page. If the defining tag is found in the middle of the record then at least one end must be found using a single rule with a single tag. The other end of the records can be found using more than one rule, but they use no more than two tags, only one searched for more than once, and all rules search for only one tag.

When searching for fields first a defining tag must be found that must indicate either the beginning or the end of the field. The defining tag is searched for starting from the beginning of the record or, if the later search was unsuccessful, the defining tag is searched for starting from the end. If such a tag can be found then the algorithm fails. After finding a defining tag which determines correctly all confirmed fields and the biggest number of unconfirmed fields the algorithm proceeds to search for rules which found the other end. First the wizard searches for a single tag that can be used to find the other end starting from the end already found. If such a tag cannot be found then algorithm tries to find a tag which determines correctly the end and then searches for a tag which used repeatedly, a constant number of times, will help to find the delimiting tag.

This implementation was chosen because it is faster and easier to implement and maintain.

Estimate and other singular extractions will be extracted by visually selecting the desired region and then label it. Thus specific information (such as estimated number of records returned by the native site) can be extracted and deposited in the right fields.

In order to improve record and text field extraction the following must be done. There can be some cases uncovered by the wizard when:

- The defining tag can be found also outside records region. In this case the wizard must search for start or/and end rules. The record extraction is valid only if the search rules are successful before the start of the record and the end rules are not successful before the end of the record.
- There is not a defining tag but only a defining structure, a succession of tags that can be found once for every record.
- The defining tag or defining structure is not the same for all records. Usually odd records can have a structure and even records can have another, for example different colors. Also, there can be a different structure for records that need to be emphasized for some particular reason in which case the pattern for finding the records will be difficult, even impossible, to find given that we know only the first record and the total number of records. In that case the user will be asked to label a particular record.

The cases uncovered by the wizard when determining text field extractions can be one of the following:

- Neither of the field ends can be found using a single rule with a unique tag. In this case, an approach similar with that used when searching for the other end with multiple rules can be employed.
- The structure is not the same for all the fields. This will be the most difficult case to tackle.
- Searching first for one end and, after finding the best choice, starting to search the other end may not lead to a solution.

One of the reasons for this can be that when searching only for the first end we can't detect that the field extracted will overlap another extracted field, for a record that doesn't contain an instance of the field searched for, because it is near information that won't be extracted, such as a label for the next field. For instance, the algorithm may detect when searching for start of description field the start of the text “Author: John Stuart Mill”. The fact that the extracted field overlaps "John Stuart Mill", assuming that the author was extracted early, can be detected only after detecting the other end and in that stage this can be too late. The user can overcome that by labelling first all the information extracted incorrectly, even if this information doesn't need to be extracted. The algorithm can be improved by searching simultaneously for both ends but this will increase the time of execution exponentially because for every potential tag found for one end the best tag for the other end must be found.

A case with a similar solution is when we have a following template. One record:

“

CRAWFORD, DANIEL J.CLEVELAND, JEFF I., IISTAIB, RICHARD O. (NASA, Langley Research Center, Hampton, VA)

AIAA-1988-4595

IN: Flight Simulation Technologies Conference, Atlanta, GA, Sept 7-9, 1988, Technical Papers (A88-53626 23-09). Washington, DC, American Institute of Aeronautics and Astronautics, 1988, p. 109-121.

<FORM ACTION="../store/MtgPaperPurchase.cfm" METHOD="Post">

„

Another record:

“

HAJELA, P. (Rensselaer Polytechnic Inst., Troy, NY) BERKE, L. (NASA,Lewis Research Center, Cleveland, OH)

AIAA-1992-4805

<FORM ACTION="../store/MtgPaperPurchase.cfm" METHOD="Post">

“

When extracting the field "IN: Flight Simulation Technologies Conference, Atlanta, GA, Sept 7-9, 1988, Technical Papers (A88-53626 23-09). Washington, DC, American Institute of Aeronautics and Astronautics, 1988, p. 109-121." a defining tag is found at the end of the field but when searching the other end there is no solution because the number of the
 tags is variable. But if a different approach will be used, searching first for the start of the field, the number of
 tags is constant, assuming that the previous fields are always present. If the later assumption is not true we have only the choice of selecting entire text as a citation field and we can have a similar problem. Another solution to this problem is to try first the fastest choices and go to the slower ones if they fail. But we still need to implement detection of the first end using multiple rules which is little difficult than searching for the other end because in the later we have more information.

9. CONCLUSIONS

In (Gottlob *et al.*, 2004) is stated that a suitable wrapping language over document trees is required to have the following properties:

(i) has a solid and well understood theoretical foundation,

(ii) provides a good trade-off between complexity and the number of practical wrappers that can be expressed,

(iii) is easy to use as a wrapper programming language,

and

(iv) is suitable for being incorporated into visual tools, since ideally all constructs of a wrapping language can be realized through corresponding visual primitives.

The structure of the language was designed to be easy to use as a wrapper programming language and is already incorporated into visual tools. Using WDL as a programming language increased the productivity of the wrappers department with 300% and allowed employment of people without programming skills.

Regarding the trade-off between complexity and the number of practical wrappers, we had an initial target of 90% coverage and we obtained 100%. Actually, the language was continually improved during implementation of the first two hundred wrappers and from that point we reached 1286 wrappers without the need to extend the language. This can be an indication that the language is complete, at least from the point of a meta-search application.

Currently, we have developed 1286 wrappers, which cover more than 4000 sources.

Regarding productivity improvements achieved using those tools our experiments showed that after third stage the development time was reduced with 20% and with an additional 30% after the fourth stage.

The automatic parsing generation might be improved from graphically markup to heuristic analyzes for automatic extraction and qualification. At this stage we’ll reduce the development time with 15%.

In this stage, the tools above will be extended to incorporate similar browser functionality and record the steps made by the user (connector developer) to perform the query. This suite tool will reduce the time needed to develop a wrapper with a total of 85%.

Other improvements that can be added are:

- Improve the usability when the wizard is used for fixes. Now the wizard starts and generates the extractions from scratch, whatever extractions already present are not taken into account.
- Add support for URL field extraction.
- Improve single URL extractions.
- Construct a database which will store information about every step executed when generating wrappers or (html pages, extracted information, chosen strategies). This is needed for immediate use for the polymorphic html pages and in the future for improving the extractions. Also this database can be used to automatically repair wrappers.
- Generate estimate extraction using user selection.
- Generate extraction for table fields, loop fields.
- Generate extractions for repeated parsing.
- Generate representation for actions such as selecting databases.
- Add support for JavaScript.
- Add support for AJAX.

A permanent task will be to reduce user interactions and maybe to improve the extraction mechanism used by connectors, based on information gathered in the database.

REFERENCES

Baumgartner, R., Flesca, S., and Gottlob, G. (2001). Visual Web information extraction with Lixto. In *VLDB Journal*, pp. 119–128.

Huck, G., Fankhauser, P., Aberer, K., and Neuhold, E.J. (1998). Jedi: Extracting and synthesizing information from the Web. In *Proceedings of Conference on Cooperative Information Systems*, pp. 32–43.

Gottlob, G., and Koch, C. (2002). Monadic Datalog and the Expressive Power of Languages for Web Information Extraction. In *Proceedings of Symposium on Principles of Database Systems*, pp. 17-28.

Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., and Flesca, S. (2004). The Lixto Data Extraction Project - Back and Forth between Theory and Practice. In *PODS '04: Proc. of 23rd ACM SIGMOD-SIGACT-SIGART Symposium. on Principles of Database Systems*, pp. 1-12.

Irmak, U., and Suel, T. (2006). Interactive Wrapper Generation with Minimal User Effort. In *Technical Report TR-CIS-2005-02*. CIS Dept., Polytechnic University, Brooklyn, NY.

Kapow Technologies. (2006). *RoboMaker User Guide*. http://kdc.kapowtech.com/documentation_6_0/robosuite/RoboMakerUsersGuide.pdf.

Kuhlins, S. and Tredwell, R. (2003). Toolkits for Generating Wrappers. A Survey of Software Toolkits for Automated Data Extraction from Websites. In *LNCS*, vol. 2591, pp. 184–198. Springer, Berlin.

Laender, A., Berthier, A., Ribeiro-Neto, Silva, A., and Teixeira, J.S. (2002). A Brief Survey of Web Data Extraction Tools. In *SIGMOD Record*, vol. 31(2), pp. 84-93.

Liu, L., Pu, C., and Han, W. (2000). XWRAP: An XML-enabled wrapper construction system for web information sources. In *Proceedings of International Conference on Data Engineering (ICDE)*, pp. 611-621.

Miler, R. (2002). *Lightweight Structured Text Processing*. PhD Thesis, Computer Science Department, Carnegie Mellon University.

Muslea, I., Minton, S., and Knoblock, C.A. (2001). Hierarchical Wrapper Induction for Semistructured Information Sources. In *Autonomous Agents and Multi-Agent Systems*, vol. 4(1), pp. 93-114.

Muslea, I., Minton, S., and Knoblock, C.A. (1998). Stalker: Learning extraction rules for semistructured, web-based information sources. In *AAAI Workshop on AI and Information Integration*.

Zhao, H., Meng, W., Wu, Z., Raghavan, V., and Yu, C. (2005). Fully Automatic Wrapper Generation for Search Engines. In *Proceedings of 14th International World Wide Web Conference (WWW14)*, pp.66-75.