Practical Issues of Integrating Advertising Data from the World Wide Web

Aivars Irmejs[1], Guntis Arnicans[2]

[1] CityPremises.com, Riga, Latvia
{aivars@serveris.lv}
[2] University of Latvia, Raina blvd. 19, Riga, Latvia
{Guntis.Arnicans@lu.lv}

Abstract. The internet provides us with a large amount of information but the necessary information is often scattered among several sites even for the same type of data, which complicates search and analysis. This problem can be solved by creating information systems which acquire and aggregate data from the Web. The process of acquiring data can be difficult due to differences between various internet sites and the way they structure data, as well as the fact that these structures keep changing. The paper analyses the most important practical issues that have to be dealt with when creating information systems for acquiring data from the Web. The paper presents a system that aggregates UK commercial real estate advertisements from various internet sites and has been in use for several years. A demonstration of the methods used to resolve the issues is included.

Keywords: web data extraction, web data integration, web semi-structured data, data acquiring automation.

1 Introduction

The World Wide Web contains a large amount of useful information but unfortunately it is distributed among numerous heterogeneous data sources that contain structured, semi-structured and unstructured data. There are many ways to solve this problem by data extraction and integration and different tools to use (e.g., [1, 2]). However the problem still remains in various business fields and in many countries – there are data that are not integrated but are demanded by consumers. One of such business fields is advertising data integration.

A lot of research has been conducted since the time that information useful to end-users began to appear on the internet. Most of the papers concentrate on a particular sub-problem or present reports about some specific business domain. It is rare to come across a report about the development and maintenance of a real working system that takes into consideration the skill of developers (developers of small to medium-size projects are not able to implement many of the techniques described in research papers) and the costs of keeping the system running (price/performance issues – which tasks are better automated and which – performed manually, for example).

In this paper we offer the lessons learned by developing and maintaining the data integration system named “CR system”. The system was developed in 2008 and has been maintained ever since. It collects and integrates UK commercial real estate advertisements from several internet sites. The advertising market is very scattered and there are approximately 1000-3000 advertisement agents that participate in it. Each agent has their own web page where the advertisements are published.

There were systems providing the integration of advertisement data before introducing “CR system” in UK. The main problem was that these systems required the advertisement agents to enter information and maintain it for each data-integrating system. Agents lacked the incentive and resources to cooperate with all such systems. Instead they concentrated on maintaining their own web pages.

“CR system” does not require the agents to do any additional work. The system collects data from their web pages, integrates it and provides the end user with a search engine capable of searching the integrated data. “CR system” has integrated approximately 800 actual data sources from the most significant agents in this market that do not prohibit data extraction from their sites.

The goal of this paper is to describe the main problems from the viewpoint of developers and maintainers of this successful implementation and to sketch practical solutions for development of an advertising data integration system or a similar integration system.

The paper is organized as follows. Section 2 describes the “CR system” architecture and lists the most important sub-problems that have to be solved. Section 3 details the issues regarding extracting data from a web page. Section 4 outlines the possibilities to automate the data-extraction process. Section 5 describes other practical issues, and Section 6 presents the conclusions.

2 Overall Data Extraction and Integration Process

“CR system” architecture is given in Fig. 1. The main parts of the system are the following:

CR Crawler reads the URLs of agents’ web pages and the corresponding processing scripts, makes connections to data sources, downloads page contents, executes the data-scanning scripts, extracts the advertisement data blocks and related items including pictures and geographic information, and saves the extracted data.

CR Backend stores the data, provides the data and control information to other components, performs primary auditing of data and processes, accumulates historical information, and prepares the information for publishing on the Web.

CR Script Editor is a tool that provides creating, editing and testing of scripts needed for data extraction from web pages.

CR Web is a Web information system for the end user that provides access to the integrated data, including searching, filtering, sorting, creating of visual presentation.

CR Manager is a Web information system that supports the maintenance of the whole “CR system”: monitoring and controlling the scanning processes, viewing all advertisements including the unpublished ones, performing the auditing procedures, enabling manual and visual validation of addresses and pictures, managing the list of advertisement agents and their services on the Web.

There are the following main steps of the integration workflow (see the numbers assigned to arrows in Fig 1):

1. *Programmer* creates scripts with *CR Script Editor*;
2. *CR Script Editor* sends the scripts or script modifications to *CR Backend*;
3. *CR Crawler* reads the list of data sources, the corresponding scripts and scanning meta-data from *CR Backend*;
4. *CR Crawler* downloads the agents' web pages for data extraction;
5. *CR Crawler* sends to *CR Backend* information about real estate and other advertising details it has obtained and the extracted URLs of pages to scan next;
6. *CR Backend* saves the relevant information in a database *CR Database*;
7. *CR Manager* reads the information about real estate, history, scanning sessions, audit reports and active processes, sends the validation and auditing results to the administrator;
8. *Administrator* controls the data-scanning processes and performs data validation;
9. *CR Backend* validates and sends the integrated real estate data to the public database *Published Advertisements*;
10. *CR Web* provides access to the published real estate advertisements.
11. End-user *Visitor* works with the integrated database.

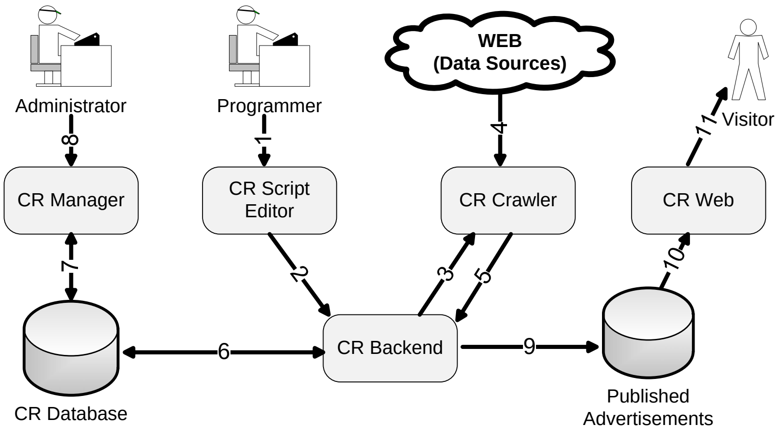

Fig. 1. CR system architecture

We now list the tasks that are very important for getting good results. Some of these tasks are not trivial to realize.

Discarding unwanted data. Web pages usually contain a lot of useless data (advertising that does not interest us, menu, headers, footers, etc.).

Finding data fields. We need to find places in HTML code or in DOM structure where the necessary data is stored (e.g., price, area, address).

Extracting values from data fields. Sometimes data fields contain unstructured text and we need to find keywords, make semantic analysis of the text, understand the context, and extract values with a high probability of being correct.

Value conversion. Numeric data are sometimes expressed in words (e.g., “one thousand”). HTML character entities may be encoded in different ways (e.g., „£” might be coded as „&#163;” (assuming the ISO 8859-1 character set) or as „&pound;”). Currency conversion may be necessary, etc.

Creating new values. For instance, price per month is calculated by dividing the yearly price by 12.

Generating identifiers. Artificial identifiers or checksums for data objects may need to be introduced to maintain them.

Eliminating data redundancy. One or more data sources may contain several records describing a particular object, sometimes with different attribute sets. We need to eliminate duplicates and consolidate attributes, sometimes different ones.

Data aggregation. For instance, we can calculate the rent price for the whole building by finding and summing the prices for each apartment.

Sorting data instances. Sorting may be necessary if the processing algorithm requires a specific data order.

Making data more accurate. For example, by using additional services and databases we can get complete a partial address or create it from geographical coordinates.

Obtaining additional data. For instance, we can download object photos, or convert them to a more appropriate format.

Data validation and correction. Depending on data correctness and completeness we might take different actions (e.g., publish, reject or submit for manual validation).

3 Data Extraction from an HTML Page

Generally we can assume that the more data sources there are the more different data coding and page organization principles would be applied. The principles can vary even within the same data source. This forces developers to analyze the structure of each data source (or even parts of it) and to form a set of methods for extracting data as correctly as possible.

For data sources that are similar to some previously processed data source format and structure, we can sometimes reuse proven scripts and methods.

To extract data fields from a page, one can

1. use the page source and process it as text, searching data by regular expressions or by text that wraps the data;
2. use the DOM structure of the page and carry out various DOM traversals and apply search functions;
3. use expression languages such as XPath, which combines node search by content, attribute values and the relative location to other nodes.

Several methods may need to be combined. For example, at first we find a node using XPath expressions, and then we extract text by matching the node content against a regular expression.

Sometimes it is useful to manipulate the page structure before extracting text by deleting unnecessary elements or inserting new tags, and then re-parsing the page. It can also be beneficial to use tools or libraries to clean and fix broken HTML structure before parsing.

It is common for data to be published in various formats, even for pages from the same site. There are several universal data normalization methods implemented for the project that can be reused in other similar projects:

number normalization (e.g., all these would typically denote the same value: “1200”, “1,200”, “1 200”, “1.2k”, etc.);

area normalization and conversion to a common system of units;

address normalization, using *Geocoding API* or searching by post code if the address text contains it;

price and rent normalization (e.g., converting the specified rent period to rent per year and converting rent per area unit to full rent).

It is also common for data to be included in nodes containing descriptive text, which typically implies that the required data lacks any markup that would help to identify it. In this case it is necessary to use the methods that can extract information from plain text (e.g., [3]). The project uses the following methods:

1. find the area, using a special “area format” (number and units);
2. determine the property type and tenure by keywords in text;
3. determine the property price and rent, using a format to find amount of money and filtering the values found by adjacent keywords.

When working with text, it is important that the text is converted to a common language encoding, for example, UTF-8. Not doing so can cause problems with special symbols, even when only English is used.

If text is saved to output structure, it needs to be formatted to a common format. Unnecessary tags need to be removed or replaced, leaving only the appropriate ones; plain text needs to be correctly formatted, keeping the intended paragraph structure.

4 Automation of the Most Important Tasks

Data extractors and wrappers can be generated automatically (e.g., [4]) or semi-automatically (e.g., [5]). For small-to-medium-sized projects it is probably wise to start with the semi-automatic approach. Since using scripts for data extraction requires creating a script for each site, it affects the scaling ability of the project with respect to data sources. The problem can be mitigated by making the script-writing process more efficient, for example by providing a special-purpose tool that (at least partially) automates the script-writing steps.

The scripts used in this project are written in a domain specific language (DSL) designed by the project developers. Programming is supported by a visual tool. Any person with basic IT knowledge is capable of creating scripts after a brief training. The scripts are formed as trees, with tree nodes containing commands. Each command receives one input parameter from its parent command and sends one output parameter to its children commands. Input/output parameters consist of text or DOM nodes. The root input parameter is the document node or the full page source code, depending on the expected input type for the command. This makes the script structure simple and allows for a clear view of the data flow.

Using the loaded web page as a reference, the visual tool allows previewing command output by selecting the commands. There are several groups of commands used – *data processing commands* (text and node search and manipulating them), *output commands* (sending found data to the output structure) and *control commands*.

For those cases when the script language capabilities are insufficient, it is possible to use the Pascal programming language to write parts of the script. Before running, the script (written in a DSL) is translated into a Pascal program.

Most of the web pages use an HTML structure that divides data fields into separate DOM nodes. In these cases it is usually possible to find the nodes using XPath expressions. The script editor has built-in functionality that allows automatic XPath expression generation for a selected DOM node or several nodes. The generated XPath expressions use attributes that are associated with the web page structure, such as *class*, *id* and *width*. From all resulting expressions the shortest one having the smallest number of matched nodes while including the selected nodes gets chosen (see more details about finding shortest paths in [6].

The script editor allows assisted script generation as well. It allows loading the reference web page in a web browser control, selecting a DOM node in the page with the mouse, choosing a field type and automatically creating a script-tree that finds the selected DOM node using an automatically generated XPath expression and sends it to the output structure. In cases when the type of the data field in a web page is determined by a label, the script editor allows to specify the label for the selected node and includes it in the generated XPath expression so that only the correctly-labeled fields get found.

The basic techniques for automatic data extraction from data sources are described in [7]. The data extraction tool WIEN was created to validate the expressed ideas. It has several problems, for instance, in dealing with a list of items. This problem is solved by a tool named STALKER [8]. STALKER requires fewer pages for learning than WIEN. Both approaches need an oracle for the learning process. A technique described in [9] works without an oracle and is based on analyzing many web pages, processing both the static and dynamic parts of them (a tool named RoadRunner).

To separate the main content from “garbage”, the methods described in [10, 11, 12] can be used. The web pages created in recent years exploit various visual presentation methods. Similar objects can be found in such pages by visual similarity [13].

Taking into account our practical experience, we declare that some problems remain too hard for automatic solving:

1. The data source allows selecting data only by filling out a request form (usually by specifying a filtering/searching condition);
2. The recognition of some data fields and their types sometimes is a hard problem even for human experts;
3. It is not easy to find a strategy for downloading large amounts of web-pages for analyzing if we do not know the structure of the data source. Bad strategy might behave like denial-of-service attack.

5 Other Practical Issues

Let us look at the other practical issues that developers may have to take into account. One of the most important conclusions is that it is too hard to create a universal tool which can extract data from any web data source. The complexity of the task is very high. It is more economical to create a data-extracting script for each data source. The key point to success is to develop supporting tools that automate all standard activities with a minimal level of errors. The maintainers of the system need supporting tools that allow easy visual result validation in unclear situations and that separate the wrong results from the good ones. The humans must act as result-validation oracles while the artificial intelligence scientists keep searching for an automatic solution.

The system needs to have a data source control module. This module checks the data source for changes in data and formats, and sends messages to the data-extracting module and/or warnings to system administrators.

The system has to support various protocols (e.g., HTTP, HTTPS), technologies (e.g., Ajax), result formats (XML, JSON). Some home pages require that the users enter their requests via forms. This means that the system needs to be able to work with forms and cookies, for instance, it may have to create an HTTP POST request and send it to the server. Some web pages use JavaScript or Adobe Flash to provide all functionality. In such cases special tools are needed for working with page. In the worst case internet browsers with appropriate plug-ins can be used, but they do consume significantly more resources on the data-integration system’s servers.

HTML source code very often does not follow the applicable standards. Code cleaning may be needed. *Beautiful Soup* and *HTMLTidy* are products that may be integrated with the system.

There are no standards describing how the paging through many tied results-pages are to be organized (when a long list of items is displayed by portions). Sometimes the paging is cyclic or it generates an infinite amount of empty pages or pages with fixed or previously seen items. If we use third part services (for example, *Google Geocoding API*, *Yahoo! PlaceFinder*, *Bing Maps*), then we have to be ready for slow response times or service being not available at all.

Not all data sources owners like others extracting data or heavily loading their servers. Legal aspects have to be taken into account and working with others’ data sources should be done as carefully as possible.

Some web pages are developed in such an extraordinary manner or data is presented so atypically that one has to spend lots of resources creating scripts. In these cases it may make more sense from an economical standpoint to simply avoid these pages. One has to be ready for a total lack of data attributes or for data being presented as free unstructured text.

6 Conclusions

Extracting data from various heterogeneous data sources, particularly from the Web pages, and integrating for users’ needs is not a trivial task. We have described our experience that was obtained by creating and maintaining the “CR system” which aggregates UK commercial real estate advertisements.

There are many research papers that describe how a particular problem can be solved, but little information on working industrial systems can be found. We reveal for other developers the most significant practical issues that may arise, hopefully saving them time and other resources while developing similar systems.

For each system, the developers have to evaluate which tasks should be automated and which would be more cheaply and reliably performed if assigned to a human. To assist humans in tasks that require high levels of intellectual effort, the developers can create tools that help making decisions.

Acknowledgements

The second author is supported by the European Social Fund Project No. 2009/0216/1DP/1.1.1.2.0/09 /APIA/VIAA/044.

References

1. Laender, A. H. F., Ribeiro-Neto, B., DA Silva A. S., Teixeira J. S.: A brief survey of Web data extraction tools. SIGMOD Record 31(2): 84--93 (2002)
2. Chang C.-H., Kayed M., Girgis M. R., Shaalan K. F.: A survey of Web information extraction systems. IEEE Transactions on Knowledge and Data Engineering, 18(10):1411--1428 (2006)
3. Embley, D. W., Campbell, D. M., Jiang, Y. S., Liddle, S. W., Kai Ng, Y., Quass, D., Smith, R. D.: Conceptual-model-based data extraction from multiple-record Web pages. Data and Knowledge Engineering, 31(3): 227--251 (1999)
4. Arasu, A. Garcia-Molina, H.: Extracting structured data from Web pages. Proceedings of the ACM SIGMOD International Conference on Management of Data, San Diego, California, pp. 337--348 (2003)
5. Doan A., Domingos P., Halevy A.: Reconciling schemas of disparate data sources: a machine learning approach. In Proc. of SIGMOD, pp. 509--520 (2001)
6. Wood, P.T.: Minimizing simple XPath expressions, Proceedings of the Fourth International Workshop on the Web and Databases, Santa Barbara, California, USA, pp. 13--18 (2001)
7. Kushmerick, N. Weld, D. S. Doorenbos, R.: Wrapper Induction for Information Extraction. International Joint Conference on Artificial Intelligence, vol. 15; N 1, pp. 729--737 (1997)
8. Muslea, I. Minton, S. Knoblock, C.: A.Hierarchical Wrapper Induction for Semistructured Information Sources. Autonomous Agents and Multi-Agent Systems, vol. 4; part 1/2, pp. 93--114 (2001)
9. Crescenzi, V. Mecca, G. Merialdo, P.: RoadRunner: Towards Automatic Data Extraction from Large Web Sites. Proceedings of the International Conference on Very Large Databases, pp. 109--118 (2001)
10. Liu, B., Grossman, R., Zhai, Y.: Mining data records in Web pages. KDD, pp. 601--606 (2003)
11. Zhai, Y., Liu, B.: Web Data Extraction Based on Partial Tree Alignment. Proceedings of the 14th International Conference on World Wide Web (WWW), Japan, pp. 76--85 (2005)
12. Yi L., Liu B., Li X.: Eliminating noisy information in Web pages for data mining. Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, pp. 296--305 (2003).
13. Cai, D. Yu, S. Wen, J-R. Ma, W-Y.: Extracting Content Structure for Web Pages based on Visual Representation. Fifth Asia Pacific Web Conference (APWeb2003), pp. 406--417 (2003).