Extracting Product Data from E-Shops

Peter Gurský, Vladimír Chabaľ, Róbert Novotný, Michal Vaško, and Milan Vereščák

Institute of Computer Science
Univerzita Pavla Jozefa Šafárika
Jesenná 5,
040 01 Košice, Slovakia
{peter.gursky, robert.novotny, michal.vasko, milan.verescak}@upjs.sk,
v.chabal@gmail.com

Abstract: We present a method for extracting product data from e-shops based on annotation tool embedded within web-browser. This tool simplifies automatic detection of data presented in tabular and list form. The annotations serve as a basis for extraction rules for a particular web page, which are subsequently used in the product data extraction method.

1 Introduction and motivation

Since beginnings, web pages are invented for presentation of information to human readers. Unfortunately, even advent of semantic web, which stays with us for more than ten years, did not successfully solve the problem of structured web data extraction from web pages. Currently, there are various approaches to web extraction methods for information that was not indented for machine processing.

The scope of *Kapsa.sk* project is to retrieve information contained within e-shop products by crawling, extracting and presenting data in unified form which simplifies user's choice of preferred product.

The result of crawling is a set of web pages that contain product details. As a subproblem, the crawler identifies pages that positively contain product details, and ignore other kind of pages.

A typical e-shop contains various kinds of product. Our goal is to retrieve as much structured data about product as possible. More specifically, this means retrieving their properties or attributes including their values. We have observed that each kind of product, called *domain* has different set of attributes. For example, a domain of *television set* has such attributes as display size, or refresh rate. On the other hand, these attributes will not appear in the domains of *washing machines* or *bicycles*. However, we can see that there exist attributes which are common to all domains, such as *product name*, *price* or *quantity in stock*. We will call such attributes to be *domain independent*. Often, the names of domain independent attributes are implicit or omitted in the HTML code of web page (*price* being the most notorious example).

Since the number of product domains can be fairly large (tens, even hundreds), we develop an extraction system, in which it is not necessary to annotate each product domain separately. In this paper, we present a method which extracts product data from a particular e-shop and requires annotation of just single one page. Furthermore, many annotation aspects are automatized within this process. The whole annotation proceeds within web browser.

2 State of the Art

Web extraction systems area is well-researched. There are many surveys and comparisons of existing systems [1, 2, 3, 4]. The actual code that extracts relevant data from a web page and outputs it in the structured form is traditionally called *wrapper* [1]. Wrappers can be classified according to the process of creation and method of use into the following categories:

- manually constructed systems of web information extraction
- automatically constructed systems requiring user assistance
- automatically constructed systems with partial user assistance
- fully automatized systems without user assistance

2.1 Manually Constructed Web Information Extraction Systems

Manually constructed systems generally require use of programming language or define a domain-specific language (DSL). Wrapper construction is then equivalent to wrapper programming. The main advantage lays in easy customization for different domains, while the obvious drawback may the required programming skill (which may be made ease by lesser complexity of a particular DSL). The well-known systems are MINERVA [5], TSIMMIS [6] and WEBOQL [7]. The OXPATH [20] language is more recent extension of XPath language specifically targeted to information extraction, crawling and web browser automation. There are possibilities to fill the forms, follow the hyperlinks and create iterative rules.

2.2 Automatically Constructed Web Extraction Systems Requiring User Assistance

These systems are based on various methods for automatical wrapper generation (also known as wrapper induction),

mostly using machine learning. This approach usually requires an input set of manually annotated examples (i. e. web pages), where additional annotated pages are automatically induced. Then, a wrapper is created according to the presented pages. Such approaches do not require any programming skills. Very often, the actual annotation is realized within the GUI. On the other hand, annotation process can be heavily domain-dependent and web-page depended and may require increased effort. Traditional tools in this category are WIEN [8], SOFTMEALY [9] and STALKER [9].

2.3 Automatically Constructed Web Extraction Systems With Partial User Assistance

These tools use automated wrapper generation methods. They tend to be more automated, and do not require users to fully annotate sample web pages. Instead, they work well with partial or incomplete pages. One approach is to induce wrappers from these samples. User assistance is required only during the actual extraction rule creation process. The most well-known tools are IEPAD [11], OLERA [12] and THRESHER [13].

2.4 Fully Automatized Systems Without User Assistance

A typical tool in this group aims to fully automate the extraction process with none or minimal user assistance. It searches for repeating patterns and structures within a webpage or data records. Such structures are then uses as a basis for a wrapper. Usually, they are designed for web pages with fixed template format and the extracted information needs to be refined or more thoroughly processed. Example tools in this category are ROADRUNNER [14], EXALG [15] or approach used by Maruščák et al. [16].

3 Web Extraction System within Kapsa.sk Project

Our design focuses on an automatically constructed web information extractor system with partial user assistance. We have designed an annotation tool, which is used to annotate the relevant product attributes on a sample page from a single e-shop. Each annotated product attribute corresponds to an element within HTML tree structure of the product page, and can be uniquely addressed by XPath expression optionally enriched with regular expressions.

Then, we have observed that many web pages on a particular e-shop portal were generated from a server-side template engine. This means that in many cases, XPath expressions that address relevant product attributes remain the same. Generally, this allows us to annotate the data only once, on a suitable web-page. (See Figure 1). To ease an effort of annotation, we discover the repeating data

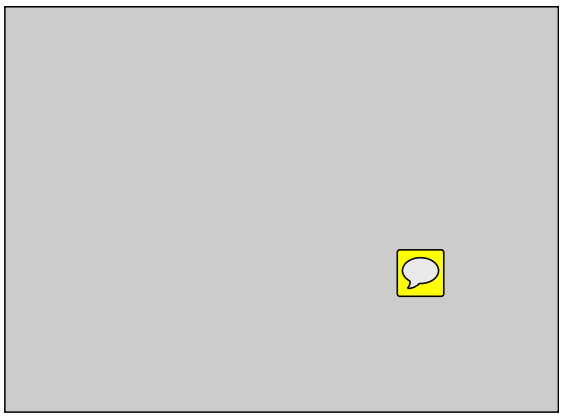

Figure 1: Extracting data from template-based pages

regions with the modified MDR algorithm [18] (more on that in section 5.1).

The result of annotation process is an *extractor* (corresponding to the notion of a wrapper) represented as a set of extraction rules. In the implementation, we represent these rules in JSON, thus making them independent from the annotation tool. (see section 4 for more information).

This way, we are able to enrich the manual annotation approach with degree of automation. Further improvements on ideas from other solutions are based on addressing HTML elements with product data not only with XPath (an approach used in OXPATH [20]), but also with regular expression. It is known, that some product attributes may occur in a single HTML element in a semi-structured form (for example as a comma-delimited list). Since XPath expressions are unable to address such non-atomic values, we use the regular expressions to reach below this level of coarseness. Although a similar approach is used in the W4F [21], we enhance that with our web-browser-based annotation tool and allow the use of modified MDR algorithm to detect the repeating regions.

4 Extractors – The Fundament of Annotation

4.1 Extraction Rules

In the first page of annotation we construct an extractor which is composed from one or multiple extraction rules, each corresponding to a object attributes. All extraction rules have two common properties:

1. addressed a single HTML element on a webpage that contains the extracted value. The addressing is represented by an XPath expression.

2. the default representation of an extraction rule in both annotation and extraction tools is JSON.

```
{
  "site": "http://www.kaktusbike.sk/terrano-lady-12097",
  "extractor": {
    "type": "complex",
    "items": [
      {
        /* -- Rule with fixed attribute name */
        "type": "label-and-manual-value",
        "xpath": "//*[contains(@class,\"name\")]/h1",
        "label": "Name"
      },
      {
         /* -- list (table rows) -- */
        "type": "list",
        "xpath": '//*[contains(@class,\"columns\")]/table/tbody/tr',
        "label": "N/A"
        "items": [
          {
            "type": "label-and-value",
            "labelXPath": "td[1]",
            "xpath": "td[2]"
          }]}]}}
```

Figure 2: Defining an extractor along with various extraction rules

4.2 Types of Extraction Rules

Generally, we define the following types of extraction rules:

Value with fixed name rule is used to extract one (atomic) value along with predefined attribute name. Usually, this attribute is specified via graphic user interface of the annotation tool. Alternatively, this is specified as a name of well-known domain-independent attribute. See Figure 2, rule `label-and-manual-value` for an example.

Value with extracted name expands upon the previous rule. The name of the extracted attribute is defined by second XPath expression that corresponds to HTML element that contains attribute name (e. g. string *Price*). The example in Figure 2 uses the `label-and-value` extraction rule.

Complex rule is a composition (nesting) of other rules. This allows to define extractor for multiple values, usually corresponding to attributes of particular product. Whenever the complex rule contains an XPath expression (addressing a single element), all nested rules use this element as a context node. In other words, nested rules can specify their XPath expression relatively to this element. The example uses the extraction rule declared as `complex`. Usually, a complex rule is a top-level rule in an extractor.

List rule is used to extract multiple values with common ancestor addressed by XPath expression. This expression then corresponds to multiple HTML subtrees. This rule must contain one or more nested extraction rules. A typical use is to extract cells in table rows (by nesting a rule for extracted name). In the example, we use the extraction rule declared as `list`.

An extractor defined with rule composition (i. e. with *complex* rule) is specifically suited to data extraction not only from a particular web page (as implemented in the user interface, see section 6), but also for any other product pages of a particular e-shop. In this case, no additional cooperation with annotation tool is required.

Annotation of domain-independent values is usually realized with the *value with fixed name* rule, since the attribute name is not explicitly available within a HTML source of the web page.

Domain-dependent attributes (which are more frequent than domain-independent ones) are usually occurring in visually structured "tabular" form. The *annotation automatization process* described in the next section, allows us to infer a list rule along with a nested *value-with-extracted-name* rule. This combination of rules is sufficient to extract product data from multiple detail on web pages. Furthermore, such combination supports attribute permutation or variation. Therefore, we can successfully identify and extract attributes that are swapped, or even omitted on some web pages. This feature allows us to create wrappers that are suitable for all product domains of an e-shop.

Moreover, this set of extraction rules may be further expanded. We may specify additional rules that support regular expressions along with XPath or extraction of attribute values from web page metadata, e. g. product identifier specified within an URL of the web page.

5 Automatizing Annotation Process

As we have mentioned in the previous section, we aim to make the annotation process easier and quicker. A product page often uses either tabular or list forms, which visually clarify complex information about many product properties (see Figure 4).

Within the annotation tool, we need to nest a *value-with-extracted-name* within a list rule. Unfortunately, it is quite difficult to address an element which contains list items by mouse click. Although it possible to directly create an XPath expression, this requires advanced skills in this language and knowledge of HTML tree structure of the annotated web page. Therefore, we identify such list elements automatically by using the modified MDR algorithm. To recall, the classic MDR algorithm [17] is based on Levenshtein string distance and on two observation on HTML element relationships within a web page. In this algorithm, a notion of *data record* used, which denotes a regularly structured object within a web page, containin product attributes, user comments etc.

- Data records are occurring next to each other or in the close vicinity. Moreover, they are presented with similar or identical formatting represented in HTML elements. Such data records constitute a *data region* (see Figure 3). Since HTML elements can be transformed into string representations, we can easily use the Levenshtein string distance.
- HTML elements are naturally tree-based. Therefore, two similar data records have similar level of nesting and share the same parent HTML element.

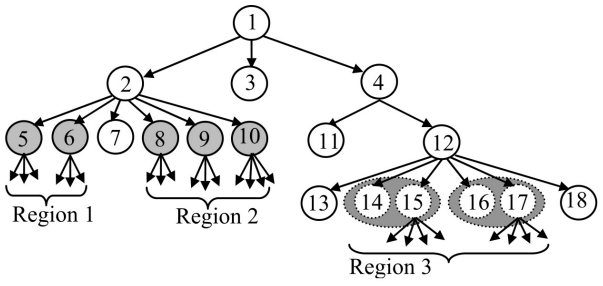

Figure 3: Data Regions

5.1 Modifying the MDR algorithm: attribute discovery

Bottom-up approach. The classical MDR algorithm traverses the HTML tree in the top-down direction and searches the data regions in the whole HTML document. Our modification uses bottom-up approach: we start from an element annotated by user and move upwards to the root element. This user annotation denotes the starting point for further comparisons, which removes the need for many computation steps.

Supporting shallow trees. The MDR algorithm has limited use for shallow tree data regions. (The original authors state minimal limit of four layers.) However, attributes or user comments are very often occurring in such shallow trees. For example, an user comment occurring in element `<div id="comment3787" class="hbox comment">` in the classical MDR is transformed into string `div`. It is obvious, that such string is too frequent in the HTML page, and therefore the Levenshtein distance cannot be use. We improve the string transformation by considering not only element name, but also some of its attributes. In our example, the element is transformed into the string `div.hbox.comment`. This vastly improves efficiency of comparison in shallow trees.

Slicing tree by levels. The classical MDR algorithm transforms elements into strings by depth-first traversal. This means, that a subtree of an element is represented as a string, which is used in the Levenshtein distance. In our approach, we slice the subtree into the levels, transform them into strings and subsequently calculate the distance for each such string. The total distance is calculated from the partial distances for each layer.

Example 1. *Consider the example on Figure 3. The classical MDR algorithm transforms the HTML tree into the following strings:* `[2, 5, 6, 7, 8, 9, 10]` *and* `[4, 11, 12, 13, 14, 15, 16, 17, 18]`*, which are then compared.*

In our modification, we slice the tree by levels and create the following strings: first layer transforms to `[2]` *and* `[4]`*, the second layer transforms to* `[5, 6, 7, 8, 9, 10]` *and* `[11, 12]`*, and final layer maps to* `[]` *and* `[13, 14, 15, 16, 17, 18]`*. Then, each pair is compared according to Algorithm 1.*

The slicing approach has positive effect on time and space complexity. This method needs not to compare elements in different layers (which usually do not have any relationship at all), and simultaneously does not decrease the distance between subtrees.

Removing non-structure elements. In the transformation process we intentionally ignore elements, which do not define a document structure. Mostly, we omit purely formatting elements, such as `b`, `i`, `tt`, `abbr` etc.

The algorithm for searching similar elements (see Algorithm 1) retrieves a set of elements A. Each element of this set is compared to an element E and return a set of similar elements, while using a similarity threshold P. For each element Y from set A, the algorithm computes a similarity of subtree of element Y with subtree of element E, level-by-level. This level-wise slicing computes two sets of element denoted as Z_x and Z_y. A similarity score is computed as a ratio of edit distance to number of compared elements.

The result score is a value between 0 and 1. A score of 0 means identical subtrees, while denotes completely different subtrees. This value is compared to pre-set threshold P.

Algorithm 1 Searching similar elements

Let A be an element set in which we search a similar element.
Let E be an element for which we search a similar element.
Let P be a similarity threshold.

```
function SEARCH_SIMILAR_ELEMENTS(A, E, P)
    similar ← []
    for Y in A do
        i ← 0
        score ← 0
        c ← 0        ▷ number of elements for comparison
        while exists i-th level in E or in Y do
            Z_x ← elements_in_level(i, E)
            Z_y ← elements_in_level(i, Y)
            score ← score + Levenshtein(Z_x, Z_y)
            c ← c + max(length(Z_x), length(Z_y))
            i ← i + 1
            score = score/c
        end while
        if score ≤ P then
            add element Y to result
        end if
    end for
    return result
end function
```

5.2 Annotation in the Annotation Tool

The process of annotation of tabular or list-based attributes is initiated by marking a single attribute value (by clicking on the particular highlighted element in the annotation tool). Then, the similar subtrees are discovered in the surroundings of this element. We start with the parent and run the algorithm for similar elements. If no similar elements are found on this level, we emerge on the parent level. Then, we rerun the algorithm for similar elements, until we find a level in which there exist elements that define all product attributes.

If we find similar elements on incorrect level (for example, it is necessary to create a list rule based on the parent or ancestor of discovered elements, or it is desired to dive one level deeper), we have a possibility to manually move above or below found elements. Effectively, this allows increasing or decreasing the level of discovered element, for which we create the list rule.

Beside the element with product value, we need to annotate an element with attribute name. Whenever any of subtrees generated by the list rule contains exactly two text elements, and one of these elements is annotated as an attribute name, the remaining element is considered as the attribute name element. Otherwise, a manual annotation assisted by user is required.

6 User Interface for Annotations

Extraction rules defined in section 4 and attribute discovery can be achieved in the annotation tool implemented as an add-on *Exago* suitable for Mozilla Firefox browser (see Figure 4). This open-source multiplatform tool (in comparison with commercial tools like MOZENDA, VISUAL WEB RIPPER, DEIXTO) represents simple and practical user interface. We support preview of extractors based on attribute discovery or manual annotations in JSON representation.

A usual workflow includes visiting a web page of a particular product, and annotating product attributes, thus declaring an extractor structure in the background (see example on Figure 2).

The declared extractor can be used in two ways: it can be immediately executed within browser context, thus extracting product values from the web page. This data can be consequently sent to server-side database to further processing. Otherwise, the JSON representation of the extractor can be sent to the server middleware. The extraction process can be performed independently on the server by one of more advanced extraction techniques [19].

7 Conclusion and Future Work

We have presented a tool for annotating attributes of product in e-shops. We have defined a language of extraction rules, which are created either with user assistance or are automatically inferred by modified MDR algorithm. Extraction rules along with algorithm were implemented as an add-on for Mozilla Firefox web browser.

The future research will be focused on further extension of the modified MDR algorithm, which will search for data regions in web pages. Beside that, we will aim to implement the product page identification on the server-side, and extend the extraction methods for support of pagination and tabs within a single web page.

This research was supported by the Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic for the Structural Funds of European Union, by project ITMS 26220220182.

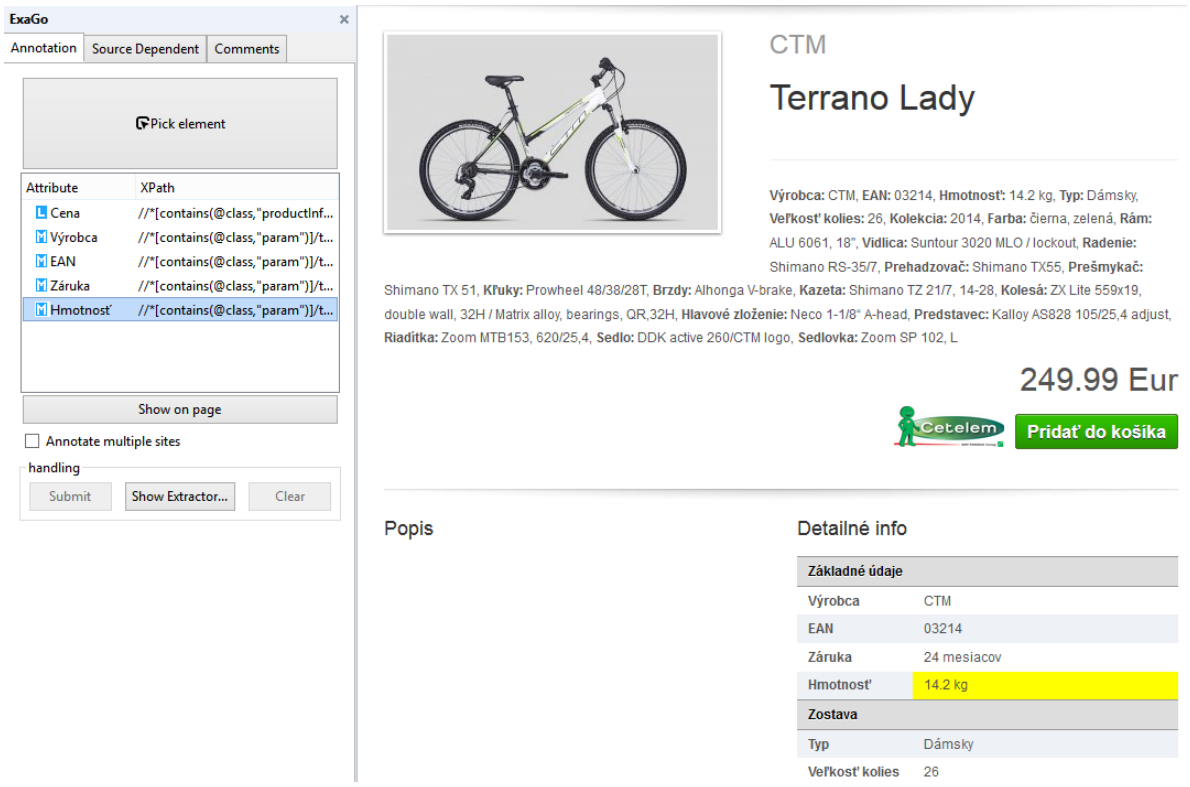

Figure 4: Extracting data from template-based pages

References

[1] C.-H. Chang, M. Kayed, M.R. Girgis, K.F. Shaalan: A Survey of Web Information Extraction Systems. IEEE Trans. Knowledge and Data Eng., vol.18, no. 10, pp. 1411–1428, Oct. 2006.

[2] Doan, A. Halevy: Semantic integration research in the database community: A brief survey. AI magazine, 2005, 26(1): p. 83.

[3] B. Liu: Web Data Mining: Exploring Hyperlinks, contents and Using Data, Second edition, Springer 2011. ISBN 978-3-642-19459-7

[4] E. Ferrara, G. Fiumara, R. Baumgartner. Web Data Extraction, Applications and Techniques: A Survey. Tech. Report, 2010.

[5] V. Crescenzi, G. Mecca: Grammars have exceptions. Information Systems, 23(8): 539–565, 1998.

[6] J. Hammer, J. McHugh, Garcia-Molina: Semi-structured data: the TSIMMIS experience. In Proceedings of the 1st East-European Symposium on Advances in Databases and Information Systems (ADBIS), St. Petersburg, Russia, pp. 1–8, 1997.

[7] G. O. Arocena, A. O. Mendelzon: WebOQL: Restructuring documents, databases, and Webs. Proceedings of the 14th IEEE International Conference on Data Engineering (ICDE), Orlando, Florida, pp. 24–33, 1998.

[8] N. Kushmerick: Wrapper induction for information extraction, PhD Thesis. 1997.

[9] C. Hsu, M. Dung: Generating finite-state transducers for semi-structured data extraction from the Web. Information Systems, 1998, 23(8): p. 521–538.

[10] Muslea, S. Minton, C. Knoblock: A hierarchical approach to wrapper induction. In Proceedings of Intl. Conf. on Autonomous Agents (AGENTS-1999) 1999.

[11] C.-H. Chang, S.-C. Lui: IEPAD: Information extraction based on pattern discovery. Proceedings of the Tenth International Conference on World Wide Web (WWW), Hong-Kong, pp. 223–231, 2001.

[12] C.-H. Chang, S.-C. Kuo: OLERA: A semi-supervised approach for Web data extraction with visual support. IEEE Intelligent Systems, 19(6):56-64, 2004.

[13] Hogue, D. Karger: Thresher: Automating the Unwrapping of Semantic Content from the World Wide. Proceedings of the 14th International Conference on World Wide Web (WWW), Japan, pp. 86–95, 2005.

[14] V. Crescenzi, G. Mecca, P. Merialdo: RoadRunner: towards automatic data extraction from large Web sites. Proceedings of the 26th International Conference on Very Large Database Systems (VLDB), Rome, Italy, pp. 109–118, 2001.

[15] Arasu, H. Garcia-Molina: Extracting structured data from Web pages. Proceedings of the ACM SIGMOD International Conference on Management of Data, San Diego, California, pp. 337–348, 2003.

[16] D. Maruščák, R. Novotný, P. Vojtáš: Unsupervised Structured Web Data and Attribute Value Extraction. Proceedings of Znalosti 2009.

[17] B. Liu, R. Grossman, Y. Zhai: Mining Data Records in Web Pages. In: Proc S IGKDD.03, August 24–27, 2003, Washington, DC, USA.

[18] V. Chabal': Poloautomatická extrakcia komentárov z pro-

duktových katalógov. Diploma Thesis. Defended on P. J. Šafárik University, Košice, 2014.

[19] P. Gurský, R. Novotný, M. Vaško, M. Vereščák: Obtaining product attributes by web crawling. WIKT '13: Proc. of the 8th Workshop on Intelligent and Know-ledge Oriented Technologies, pp. 29–34, 2013.

[20] T. Furche, G. Gottlob, G. Grasso, C. Schallhart, A. Sellers: OXPath: A language for scalable data extraction, automation, and crawling on the deep web. The VLDB Journal 22(1): 47–72, 2013.

[21] A. Saiiuguet, F. Azavant: Building intelligent Web applications using lightweight wrappers. Data and Knowledge Engineering 36(3): 283–316, 2001.