The Lixto Project: Exploring New Frontiers of Web Data Extraction*

Julien Carme[1], Michal Ceresna[1], Oliver Frölich[1], Georg Gottlob[2], Tamir Hassan[1], Marcus Herzog[1], Wolfgang Holzinger[1], and Bernhard Krüpl[1]

[1] Vienna University of Technology, Database and Artificial Intelligence Group, Favoritenstraße 9-11, A-1040 Wien, Austria

[2] Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD, United Kingdom

Abstract. The Lixto project is an ongoing research effort in the area of Web data extraction. Whereas the project originally started out with the idea to develop a logic-based extraction language and a tool to visually define extraction programs from sample Web pages, the scope of the project has been extended over time. Today, new issues such as employing learning algorithms for the definition of extraction programs, automatically extracting data from Web pages featuring a table-centric visual appearance, and extracting from alternative document formats such as PDF are being investigated.

1 Introduction

Web data extraction is an active research field, dealing with the subject of extracting structured data from semi-structured Web sites. In order to extract structured data from a Web page, we need to generate an appropriate extraction program, called a wrapper. We can distinguish two main methodological approaches for constructing such extraction programs: the supervised and the unsupervised approach. In the supervised approach, an operator needs to define the wrapper program either by coding the program manually or by using a visual development environment to generate the program code. In the unsupervised approach, the system generates wrappers automatically from a given set of heuristics and domain knowledge.

Whilst the unsupervised approach is very scalable in terms of the number of input Web pages that can be processed, this comes at the cost of precision. Due to the quality control inherent in the supervised approach, this approach is best suited where highly accurate data with an almost zero failure rate is required.

The Lixto Visual Wrapper introduced in [3] employs a supervised approach to generate wrapper programs from a given sample Web page. The operator highlights relevant data items on the Web page and the system generates logic-based extraction rules to extract these data items from the sample Web page

* This work is funded in part by the Austrian Federal Ministry for Transport, Innovation and Technology under the FIT-IT Semantic Systems program.

D. Bell and J. Hong (Eds.): BNCOD 2006, LNCS 4042, pp. 1–15, 2006.
© Springer-Verlag Berlin Heidelberg 2006

and other web pages with a similar structure. These rules utilize the document structure and specific attributes of the user-selected data items to locate other relevant data instances to be retrieved. The logical foundation on Web data extraction, and on the complexity and expressive power on data extraction using the Lixto approach were studied in [10, 11, 9].

In the first part of this paper, we will report on a recent addition within this Web data extraction framework to employ a learning strategy to select an optimal set of attributes for identifying the data items on a Web page. In the next part, we will discuss our approach to unsupervised data extraction from Web pages. Another topic that we are currently investigating is the extraction from non-HTML formats such as PDF. We will use a use case, i.e., extracting data in the domain of digital cameras, to illustrate the techniques that we have developed in these various fields of Web data extraction. Furthermore, we will report on industrial applications of Web data extraction and highlight the benefits that these applications can generate in a real-world setting.

2 Supervised Wrapper Generation

We will use the digital camera domain to illustrate typical use cases for Web data extraction. Let us imagine the application of monitoring camera prices. Assume that we have several competitors and we want to *continuously* monitor their websites for price development of the listed goods. We store the prices collected from their websites into a local database. We can then use business intelligence tools to analyse the aggregated prices, and this will allow us to react to market changes with more effective pricing strategies and advertisement campaigns. Figure 1 shows a sample of Web pages from the Dell online shop that serves as an example for similar online shops from which price information could be extracted.

This use case favors the usage of a supervised approach due to the following requirements:

- if the wrapping algorithm does not work on the given website, we cannot go to another site and collect the prices there;
- accuracy (precision/recall) must be high, because business decisions rely on the gathered data, and the solution therefore must guarantee the quality of the results obtained with the wrapping service;
- deep Web navigation is required before the actual data can be wrapped, e.g., it is required to fill Web forms or handle JavaScript (AJAX) execution.

In the following example, we need to collect prices of digital cameras from the Web site `www.dell.com`. To obtain prices for some of the cameras we have to navigate to the detail pages of the shopping cart. Informally, the problem we are trying to solve is: given a Web site (or a set of Web pages) as input and a user knowing what should be extracted from the Web site, we need to construct a wrapper to extract exactly the required information items.

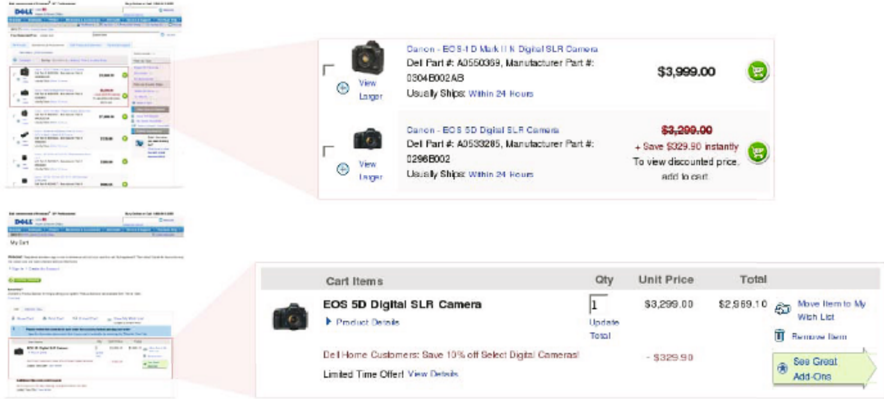

Fig. 1. Example of an information extraction scenario where a combination of wrapping and focused crawling is required

2.1 Wrapper Structure

The navigation sequence, together with the wrapper, is captured in a navigation language that we have described in one of our previous publications [4]. The navigation sequence is created by an algorithm that records interaction of the user with the Web browser and stores all mouse and key events that occur. An example of the navigation sequence with an embedded wrapper for the Dell shop scenario is shown in Algorithm 1.

The wrapping itself is embedded in the **extract** function. This function operates on DOM trees – the standard tree model of Web pages in modern Web browsers. It receives as input a list of DOM nodes and a definition of the extraction (called *pattern*), and outputs another list of DOM nodes. For example, in Algorithm 1, one of the extract functions receives a *camera* node and $pattern_price_1$ as input and returns a *price* node as output.

2.2 Learning Patterns

Patterns describe one specific extraction task, for example, the extraction of the $pattern_camera$ or $pattern_price_1$ from the wrapper in Algorithm 1. To define each pattern, we use a boolean combination of *basic conditions*, denoted as $C(p, n)$.

The basic condition $C(p, n)$ is a function that tests the correct position of the target DOM node n with respect to the context DOM node p, and local properties of the DOM node n. Examples of local properties of n are the presence of a given attribute with some value, or the existence of another sibling node s. Formally, $C(p, n)$ is a triple $(path_1, path_2, test)$ such that

$$C(p, n) \Leftrightarrow path_1(p, n) \wedge \exists s \; path_2(n, s) \wedge test(s)$$

Algorithm 1. Navigation sequence and wrapper for the Dell online shop

```
# load main page
load('http://www1.us.dell.com/content/...')
# wrap all camera entries
cameras = extract(#doc, pattern_camera)
for c in cameras do
  c.name = extract(c, pattern_name)
  c.image = extract(c, pattern_image)
  c.price = extract(c, pattern_price1)
  if c.price == None then
    # load next page
    cartImg = extract(c, pattern_cartImg)
    sendClick(cartImg)
    # wrap price from cart
    c.price = extract(#doc, pattern_price2)
    # return to main page
    goBack()
    goBack()
  end if
end for
```

The expression $path_1$ and $path_2$ are called XPath [19, 12] expressions. As we have shown earlier [7], there are boundaries of query-based learnability for XPath expressions. Therefore, here we use XPath expressions of simplified for, which contain only the child (/) and ancestor (//) steps, index test ([i]) and no wildcards (*) in tag names.

For example, for the wrapper in Algorithm 1, *pattern_camera* is defined as $(//table/tr, ., true)$, $pattern_price_1$ as $(./td/b, ., @bgcolor! = \text{`}red\text{`})$ and $pattern_price_2$ is defined as $(//table/tr/td[3], ., @bgcolor! = \text{`}red\text{`}) \wedge (., ./parent :: */parent :: */tr/td[3], text() == \text{`}Unit\ Price\text{`})$.

When building our wrapper, no annotated data (Web page) exists in advance. Therefore, we interact with the user to query the required annotations, but with a goal to minimize the number of requested inputs. The patterns are then induced from the positive and negative examples received from the user during the interaction outline in Figure 2.

In each iteration cycle of the interaction, an exhaustive set of basic conditions is generated from the set of examples that have so far been received. Then, an optimal combination of some of these conditions is learned using a DNF-learning algorithm. Unfortunately, the number of variables and therefore basic conditions in the target formula is not bounded. This implies that the Vapnik-Chervonenkis dimension of the hypothesis space of all boolean combinations of the basic conditions is not bounded, and the problem is therefore not PAC learnable [5].

Also, to the best of our knowledge, PAC- and query-based learnability of the DNF itself is not known. Therefore, for our implementation to remain tractable, we limit ourselves to only learning the k-DNF. The learning algorithm works

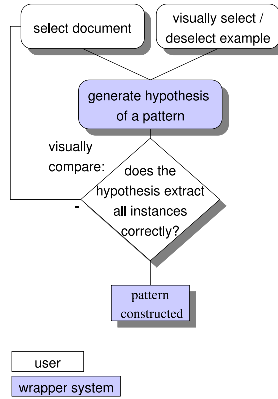

Fig. 2. Interaction of a user with the learning algorithm

in the following way: first all conjunctions of size smaller than k are generated, and then the minimum disjunction of these conjunctions, still consistent with the examples, is searched. Note that although finding the minimal disjunction is NP-complete, efficient heuristics are known.

2.3 Results

The main advantage of the learning algorithm is that it allows non-experienced users to build wrappers in an easy to understand way—point and select (or deselect) the wanted (or unwanted) instances with the mouse.

Our experiments show that the interactivity of the learning algorithm reduces the number of required examples to define a pattern. Usually, only 3–5 examples are required to build a pattern. This is due to the fact that redundant examples are not part of the annotation, e.g. receiving an image as a redundant positive example, if our hypothesis already extracts images, or receiving a hyperlink as an unrelated negative example if the pattern should extract only images.

Another advantage of the approach using boolean combination of basic conditions is that it generates wrappers that are understandable by human users. This allows the user to check or manually alter the learned pattern. The easiness of understanding is in contrast with other approaches used in information extraction, such as Hidden Markov Models or Conditional Random Fields, which learn a vector of real-valued parameters that is not intuitive for humans to understand.

3 Unsupervised Wrapper Generation

In contrast to the supervised approach, the unsupervised approach is feasible if the goal is to extract from a huge number of Web resources without the need to be able to successfully process every single Web resource that potentially holds relevant data items. For example, in order to harvest domain knowledge about cameras, it is sufficient to process a fraction of all potential descriptions

of digital camera models on the Web in order to generate a knowledge base on camera models describing all available camera models and their features.

3.1 Resource Discovery and Focused Crawling

In the unsupervised, approach the system needs to navigate and extract data from Web pages fully automatically. The basic idea here is to mimic the behaviour of an human expert. A person, given the task of collecting addresses of pages containing useable tabular information about digital cameras, will typically apply the following strategy: First, the person uses an internet search engine to find websites about cameras. To do this, the user will pose queries to the search engine that contain terms likely to appear on such a website. The user must have knowledge about the digital camera domain, i.e., the specific language and technical terms used. From the search engine's results, the user then sorts out the relevant items and repeats the process, this time using a slightly different query. After some iterations, some websites will stand out as exceptionally valuable sources of information—appearing in the search engine results in each of the queries—thus indicating that they are not only relevant to the exact phrase that was submitted during a single query but are indeed relevant to the whole domain.

At this point, the user will deviate from the initial strategy of using a search engine and begin to investigate these interesting websites directly. The user knows that information on the Web comes in a clustered form: a website that talks about some cameras is likely to talk about all cameras, or will at least contain links to such sites. The user will start *browsing* websites, having basic prior knowledge of how navigation on a website works (using fold-out menus, hyperlinks, forms, etc). The user will then need to learn the particular idiosyncracies that are used by each website to organize information. The user already knows some of the relevant pages on the site from the search engine results; now he has to find the path through the website's navigational structure that leads to these pages. Once this path is found, slight variations of it (i.e., go one step back and try alternatives) will uncover a wealth of relevant pages.

Following this two-step strategy that a human expert would employ, we constructed a software implementation that works in two stages: In stage 1 we select a small sample S from a collection C of phrases pertinent to our domain at random. Very common words appearing in C are given a smaller probability of being selected. S is converted to a (conjunctive) query and submitted to a search engine; the top rated results are grouped by website and stored for further processing. This query process is repeated until no significant new information shows up, typically after several dozen iterations.

After this, we use the table extraction algorithm explained later in this section to iterate through all the pages found and determine which pages contain extractable tables, and are therefore relevant for further processing. Eventually, we obtain a list of relevant websites p_i, each associated with a set of relevant pages that we call templates T_i.

In stage 2 we apply a Web crawler to all relevant websites, starting with the website that contains the most templates and therefore looks the most

promising. This crawler is focused on finding pages that match the templates T_i closely. For the matching algorithm we use a measure of *structural similarity* of pages as follows: A HTML page is reduced to its *skeleton code* by first removing all text nodes, tag attributes and closing tags. The remaining sequence of tags is then converted to a string by replacing each tag by a unique character. Once we have determined the skeleton codes s_1, s_2 of two pages we can measure their structural similarity $d(s_1, s_2)$ with an appropriate string distance function d; at the moment we use the Levenshtein distance [15]. The threshold distance d_T, above which we reject a page as being not sufficiently similar to our template pages, is determined by computing the pairwise distances d_{ij} among the T_i. We observed a normal distribution of these d_{ij} and use the 99% quantile of the observed distribution as the threshold d_T. Pages that fall below the threshold are called extraction candidate pages, or candidate pages for short.

Learning how to navigate a site in the same way as a human user turned out to be the most difficult part of the problem: to rely on structure and image pattern recognition—both of which are computationally expensive tasks—would not be feasible in a crawler that is expected to process thousands of pages in a short time. Instead, we turned to analysing the graph G spanned by the hyperlinks among the pages of a website. Our assumption is that the navigational pages have a special, central position in G that we can identify. After all, functional navigation should be available to the user at any time and can be used to reach even the remotest places on the website—the "trunk" of the website, so to speak. Therefore, a random surfer on the website would invariably visit those central pages more often—and we can use the PageRank algorithm [17] to identify these navigational pages.

The crawler keeps following outgoing links on pages with the highest page ranks until it hits upon one of the T_i or a page sufficiently similar to it. Once this happens, a new heuristic comes into play: *hubs*, pages that contain links to a large number of candidates, can be recognized. A page gets a *hub score* proportional to the number of extraction candidate pages it links to. The heuristic that decides which page to expand next is based on a weighted sum of both the page rank and the hub score of the page. This way, the crawler turns its attention from navigating the website to exploiting the nest of candidate pages it has stumbled upon.

3.2 Results

Figure 3 gives an indication of the crawler's performance when set to explore a typical digital camera review website. It shows the harvest rate; the ratio of candidate pages to visited pages, against the number of iterations. The crawler first begins by visiting pages leading out from the main navigation, then hits upon the first candidate at around iteration 50. It quickly focuses on the area of this result and stabilizes, turning out new candidate pages every four iterations. Due to irrelevant links present on the hub pages, the performance here never gets closer to 1.

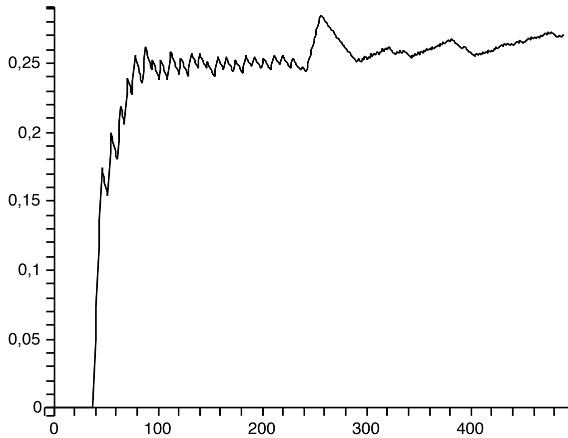

Fig. 3. Harvest rate over iteration

3.3 Automatic Table Extraction from Web Documents

There are situations where it is desirable to make the extraction process completely unsupervised. For example, many technical product descriptions on the Web are in the form of tables. A system that could locate, segment and analyse these tables automatically would therefore be of great value: it could assist the designer in the wrapper construction process by providing easier navigation within the document under consideration, or it could be used on its own to build a fully unsupervised extraction system to automatically extract data tables from a large amount of Web pages. We are currently implementing such a table location and analysis component.

What makes tables different from free text is that they are inherently concise. By extracting information from tables, we can avoid dealing with most of the complexity of natural language, since tables usually contain information in a condensed style. Thus, analysing the content with extraction ontologies [8] is more promising than in the case of free text.

Several articles deal with the problem of classifying HTML tables as genuine or non-genuine. This comes from the fact that the HTML `<table>` element is often used just for the purpose of implementing a specific page layout. It is therefore crucial for methods that analyse the source code of a Web page to identify only those genuine table elements that are not for layout purposes. By operating directly on the visual rendition, such a classification becomes obsolete.

Traditional wrappers operate on HTML input either in the form of a sequential character string or a pre-parsed document tree. With the Lixto Visual Wrapper, the wrapper is specified visually by interactively clicking on the rendition of a page to annotate relevant content. The Lixto software then determines the node in the pre-parsed document tree that best matches the selected region and generates appropriate extraction statements. For automatic table extraction, we decided not to use the HTML source code at all: if it is possible to visually define

the relevant data area, all the data required to locate the extraction instance is clearly contained in the visual rendition. Going back to the document source code, whether pre-parsed or not, is therefore an unnecessary step. If the wrapper can be grounded on the same visual properties of a document that enable the user to mark the relevant parts of the page, it should also be more robust regarding future changes of the page.

3.4 Table Extraction Algorithm

We are currently implementing an automatic table location and analysis algorithm. This algorithm operates on the rendition of a Web page provided by a Web browser and thus avoids all the peculiarities and complications involved with the interpretation of HTML and associated CSS code. The goal of the algorithm is the identification of data-centric tables and the subsequent transformation of the data contained in the table into a structural form preserving the relationships between table cells. According to the literature [14], the steps involved are: table location, segmentation, functional analysis and structural analysis. So far, we have concentrated our work on a limited number of physical table models; essentially simple, unnested tables with the possibility for intermediate headings appearing as additional lines in the table.

Unlike most of the other implementations in the literature, our table algorithm works in a bottom-up fashion by starting from pixel positions of single words that have been determined with the help of the Web browser. These word bounding boxes are grouped into larger clusters of possible cells based on their adjacency, which is illustrated as step 1 in Figure 4. Since the pixel coordinates are derived from the Web browser layout engine, we can assume that there is no noise in the data and that there is a true adjacency of neighbouring cells with a pixel distance of zero.

As in the table definition given in [18], we do not consider line-art or other graphical properties of tables; we identify and segment a table just from the positions of its inherent word bounding boxes. Rather than looking for tables as a whole, we start by trying to identify possible table columns. A column candidate in our context is a collection of cells that, within a small tolerance, share a common coordinate on the horizontal axis. This can be either the left border, centre, or right border pixel coordinate of the cell to account for left aligned, centered, or right aligned columns. Figure 4 shows the column identification as step 2.

On the vertical axis, we allow up to one non-column cell between every two column candidate cells to account for the possible intermediate headings mentioned above; if this limit is exceeded, the column candidate is split into two column candidates. Then we investigate whether all the separating cells (in other words, the intermediate heading candidates) share a common coordinate. If they do, we have found a column candidate. Otherwise, we split our column candidates into column candidates that will be treated separately.

In the next stage, we try to find the best column candidate combination that could possibly form a data table. Here we follow a strategy that we call

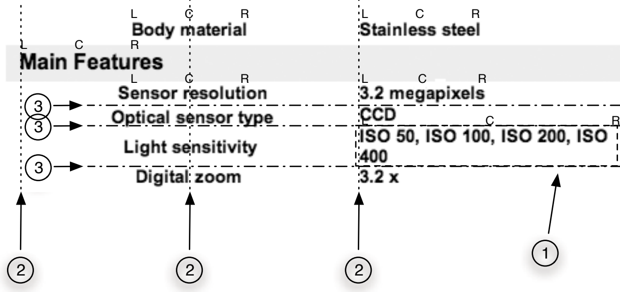

Fig. 4. Operation of the table extraction algorithm on part of a sample page

comb alignment of columns: we look for adjacent columns where we observe only 1:n or m:1 relationships between adjacent table cells. This means that we can handle cases where a cell in one column corresponds to several cells in the second column, usually representing a hierarchical relationship between these cells. Step 3 in Figure 4 illustrates the comb alignment between two columns. If we can establish such an alignment, we can derive a proper table segmentation.

In the final stage, an analysis of the segmented table is performed. Here we try to recover the relation of cells in the table or, put differently, the reading order of the table. With the information about intermediate headings and the direction of the comb alignment, we can already make a judgement about the functional role of the respective cells. The analysis is finally finished by assigning subject, predicate or object roles to the table cells based on cell neighbourhoods and on our knowledge of table models.

Because our unsupervised table extraction algorithm generates a large number of triples, its results are well suited for a statistical analysis aimed at leveraging the great redundancy of information on the Web. This is in stark contrast to other approaches that rely just on a few sources. The aggregation and integration of all these information fragments is the objective of another research effort in the Lixto context.

4 Wrapping from PDF Files

In today's Web, the vast amount of HTML data is complemented by a significant number of documents published in Adobe's Portable Document Format (PDF). In general, these documents are primarily intended for printing, and many business-critical documents fall into this category. Examples of such documents include financial reports, newsletters and price lists, such as our example in figure 5 (left) from the digital camera domain. Clearly, the ability to semi-automatically extract information from these documents proves to be extremely useful for a number of business applications.

The success of PDF can be attributed to its roots as a page-description language. Any document can be converted to PDF as easily as sending it to the

printer, with the confidence that the formatting and layout will be preserved when it is viewed or printed across different computing platforms. This ease of publication has led to a lot of data on the Web being available only in PDF format, with no corresponding HTML alternative.

Unfortunately, this approach presents one major drawback: most PDFs have little or no explicit structural information, making automated machine processing and data extraction a difficult task. Although later versions of the PDF specification support the use of XML tags to denote logical elements, these are seldom found in business documents.

Our PDF extraction functionality within Lixto utilizes a variety of techniques from document understanding literature to attempt to rediscover the logical structure from the layout of the document. This structure can then be used in a similar way to the HTML parse tree to locate data instances for wrapping.

In this section we describe our recent advances in PDF wrapping within Lixto, and present an insight into our current work in this area, and what our future releases may offer.

4.1 The Wrapping Process

The PDF import filter within Lixto is automatically activated when the input document is detected as a PDF. Our algorithms detect structures on the page, such as columns, lists and tables, and represent them in XHTML, much like a web page. The wrapper designer is then able to interact with this representation in the same way as with a web page, as shown in the example in figure 5 (right).

The latest version of our PDF filter benefits from a several improvements to our document understanding algorithms, and can now produce good results even with relatively complex layouts. The remainder of this section details some of the techniques that we have used.

Document pre-processing: In general, the first step in understanding a document is to segment it into blocks that can be said to be *atomic*, i.e. to represent

Fig. 5. Example of wrapping from a PDF price list

one distinct logical entity in the document's structure. Many of the segmentation techniques in document understanding, such as those utilized in [2] and [1], have been developed by the OCR community, and take a scanned, binarized image of the page as input. Whilst we could make use of these techniques by rasterizing each page of the PDF, this process would throw away useful information, introduce noise, waste processing time and essentially take us backwards. Therefore, we choose to segment the page directly on the object data that is contained within the PDF.

A PDF file is little more than a collection of characters and graphic objects placed on a page. Referring again to our example in figure 5, we consider the title, the address of the store and the other single lines of text all to be distinct logical entities. Inside the table, each individual cell is a distinct logical entity. This definition gives us sufficient granularity for locating these data items later.

- **Line finding:** In a PDF file, text is stored in discrete blocks, usually with no more than 2–3 characters per block (although this can depend on the program used to generate the document). The first step is therefore to merge these text fragments into complete lines. Space characters are not always included in the original source, and therefore must be added to separate words if the distance between two neighbouring blocks is too large. Our algorithm examines the spacing between each character and, therefore, copes with a variety of different character spacings.
- **Clustering:** The next stage is to merge these lines into discrete blocks that are logically distinct. As text in a PDF can use a variety of different fonts, sizes and leadings, we make use of a *variable-threshold* clustering algorithm. This algorithm examines a variety of different possible groupings of paragraphs, and a consistency heuristic is used to determine the correct grouping from this set. Further heuristics are used to detect tabular structures and ensure that each cell is distinct.

Logical structure understanding: After page segmentation, the task is to identify higher-level logical relationships and detect substructures, such as lists and tables, within the page. Currently, we have a set of heuristics that detect multiple layers of headings, and cope with multiple column layouts. Our table understanding algorithm converts tabular structures to `<table>` elements in our XHTML representation, and can detect spanning columns or rows.

We are now investigating the use of an ontological framework to abstract these rules and heuristics from our code. This will enable the rules in future releases to be more easily adapted, and for domain-specific rules to be modularly "plugged in".

4.2 Future Developments

Currently, wrapping from PDF is a two-step process. First, the PDF document is imported, and the user then interacts with its representation in XHTML. To improve interaction with the user, we are also developing a method that will allow the user to select the desired wrapping instances directly on a rendition of the PDF.

Behind this graphical rendition, the document is represented as an attributed relational graph. Each block is represented as a vertex, and the vertices are interconnected with various logical and geometric relationships. Wrapping is then performed by the application of error-tolerant graph matching algorithms, such as those described in [16]. This approach is described in more detail in our forthcoming paper [13]. As well as the obvious benefits in user-friendliness, this method will also allow more powerful wrappers to be generated, for a wider variety of applications.

5 Application in Competitive Intelligence

In this chapter, we will give an example of a business case in the domain of competitive intelligence. This business case describes the process using Lixto for Web data extraction, transformation, and delivery to the data warehouse of the SAP Business Information Warehouse (SAP BW).

A company sells consumer electronics, such as digital cameras, computers and cellular phones, with a product catalogue of more than 1000 items (short: P1000). Before using the Lixto software, many employees of the company spent many hours a day searching the Web to collect information about their competitors' pricing for items from the P1000 catalogue. The price information retrieved was used for monthly price definitions. Product availability and regional price differences should also be included in the data analysis.

By using the Lixto suite, Web pages of online shops of several competitors are automatically searched on a daily basis. For a complete Web site, just one Lixto wrapper is necessary. For every product on an overview page in the online shop, the wrapper extracts all information (even from sub-pages with detailed information). By automatically clicking on the "next overview page" button at the bottom of the page and applying the same wrapping procedure to the succeeding overview Web page, all necessary information can be retrieved for all items sold in the online shop, from all overview pages and all sub-pages. Complete product information is retrieved, i.e. *price*, *manufacturer*, *model name*, *model description*, *availability*, *discount rates*, *combined offers*, etc. The wrapper generates a hierarchically organized XML data file in a defined standard data model. Highly nested structures representing the connections and interrelations between the information items, such as *price* and *combined offers*, are possible and allow for a detailed data analysis later.

Within the Lixto Transformation Server, the XML data from different wrappers is then aggregated, reformatted and normalized. For example, all price information (e.g. in £ Sterling or Swiss Francs) are normalized to the company group standard currency (Euros), and differences in taxation are accordingly considered to allow for a standardized price comparison. Finally, the data is reformatted within the Lixto Transformation Server into SOAP, so that the retrieved information can be integrated into the SAP BW using Web services.

The data is then automatically transferred to the SAP BW in an automatic ETL process. Within SAP BW, sophisticated pre-defined data analysis and work-

flow capabilities exist. Together with the Web data supplied by the Lixto Suite, it is now possible to automatically define prices on a weekly or even on a daily basis, taking into account short-term and regional market price fluctuations. This "intelligent pricing" can increase the company's revenue margins for their products and altogether increase the revenue per product. In practice, an increase between 1% and 4% can be achieved.

With Lixto, the whole process of defining wrappers and data flows is performed semi-automatically in a graphical user interface. Within the Lixto Transformation Server, graphical objects symbolize components, such as an integrator for the aggregation of data, or a deliverer for the transmission of information to other software systems. By drawing connecting arrows between these objects, the flow of data and the workflow are graphically defined. In our example, the time-consuming and mostly manual process of mapping items from the competitors' Web sites to equivalent items from the P1000 product list is accomplished in the GUI, allowing for fast and effective data mapping.

6 Conclusion

In this paper we have reported on the latest developments in the Lixto project. In the field of supervised data extraction from Web documents, we have highlighted the benefits of employing learning strategies to guide the user in selecting relevant information items to define patterns in wrappers. For unsupervised data extraction, we have revealed strategies for resource discovery and focused crawling, as well as automatic table extraction from Web documents. We have also discussed the related issue of extracting from non-HTML document formats such as PDF. Finally, we have given a brief description of a real-world business case that illustrates the applicability of Web data extraction technology in competitive intelligence solutions.

References

[1] Aiello, M., Monz, C., Todoran, L. and Worring, M: Document understanding for a broad class of documents. Int. J. of Document Anal. and Recog. **5(1)** (2002) 1–16

[2] Altamura, O., Esposito, F. and Malerba, D.: Transforming Paper Documents into XML Format with WISDOM++. Intl. J. of Doc. Anal. and Recog. **4(1)** (2001) 2–17

[3] Baumgartner, R., Flesca, S. and Gottlob, G.: Visual Web Information Extraction with Lixto. Proceedings of the 27th International Conference on Very Large Data Bases (VLDB 2001), Rome, Italy, (2001) 119–128

[4] Baumgartner, R., Ceresna, M. and Ledermüller G.: Automating Web Navigation in Web Data Extraction. Proceedings of International Conference on Intelligent Agents, Web Technology and Internet Commerce, Vienna, Austria (2005) (to appear)

[5] Blumer, A., Ehrenfeucht, A., Haussler, D. and Warmuth M. K.: Learnability and the Vapnik-Chervonenkis dimension. J. ACM **36(4)** (1989) 929–965

[6] Chakrabarti, S., van den Berg, M., Dom, B.: Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. Computer Networks. **31(11–16)** (1999) 1623–1640
[7] Ceresna, M. and Gottlob G.: Query Based Learning of XPath Fragments. Proceedings of Dagstuhl Seminar on Machine Learning for the Semantic Web (05071), Dagstuhl, Germany (2005)
[8] Embley, D. W.: Toward Semantic Understanding – An Approach Based on Information Extraction Ontologies. Proceedings of the Fifteenth Australasian Database Conference, Dunedin, New Zealand (2004) 3
[9] Gottlob, G., Koch, C.: A Formal Comparison of Visual Web Wrapper Generators. SOFSEM 2006: Theory and Practice of Computer Science, 32nd Conference on Current Trends in Theory and Practice of Computer Science, Merín, Czech Republic, (2006) 30–48
[10] Gottlob, G., Koch, C.: Monadic datalog and the expressive power of languages for Web information extraction. J. ACM **51(1)** (2004) 74–113
[11] Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., Flesca, S.: The Lixto Data Extraction Project - Back and Forth between Theory and Practice. Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGAR Symposium on Principles of Database Systems, Paris, France (2004) 1–12
[12] Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. ACM Trans. Database Syst. **30(2)** (2005) 444–491
[13] Hassan, T., Baumgartner, R.: Using Graph Matching Techniques to Wrap Data from PDF Documents. To appear in Proceedings of the 15th International World Wide Web Conference (Poster Track), Edinburgh, UK (2006)
[14] Hurst, M.: The Interpretation of Tables in Texts. PhD thesis, University of Edinburgh (2000)
[15] Levenshtein, V. I.: Binary Codes Capable of Correcting Spurious Insertions and Deletions of Ones. Russian Problemy Peredachi Informatsii. **1** (1965) 12–25
[16] Llados, J., Marti, E. and Villanueva, J. J.: Symbol Recognition by Error-Tolerant Subgraph Matching between Region Adjacency Graphs. IEEE Tran. on Pattern Anal. and Mach. Intel. **23(10)** (2001) 1137–1143
[17] Page, L., Brin, S.: The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks. **30(1–7)** (1998) 107–117
[18] Silva, A. C., Alipio, J., Torgo, L.: Automatic Selection of Table Areas in Documents for Information Extraction. 11th Protuguese Conference on Artificial Intelligence, EPIA (2003) 460–465
[19] XML Path Language (XPath), Version 1.0. http://www.w3.org/TR/xpath