Learning Rules to Pre-process Web Data for Automatic Integration

Kai Simon, Thomas Hornung, Georg Lausen
Institut für Informatik, Universität Freiburg
Georges-Köhler-Allee, Gebäude 51
79110 Freiburg i.Br., Germany
{ksimon,hornungt,lausen}@informatik.uni-freiburg.de

Abstract

Web pages such as product catalogues and web sites resulting from querying a search engine often follow a global layout template which facilitates the retrieval of information for a user. In this paper we present a technique which makes such content machine-processable by extracting and transforming it into tabular form. We achieve this goal via ViPER, our fully automatic wrapper system, while localizing and extracting structured data records from suchlike web pages following a sophisticated strategy based on the visual perception of a web page.
The first contribution of this paper is to give deep insight into the post-processing heuristics of ViPER, which become materialized by a set of rules. Once these rules are defined, the regular content of a web page can be abstracted into a relational view. Second, we show that new, unseen contents rendered with the same layout, only have to be extracted by ViPER, whereas the remaining transformation can be performed by applying the learned rules accordingly.

1. Introduction

The Internet has revolutionized the way we search for information and also the way we exchange information. However, due to the fact that a huge amount of information available on the Web is only accessible through presentation-oriented HTML pages, most of the interactions take place between humans. We present a sophisticated strategy based on the visual perception of web pages empowering machines to extract, represent and process structured information resources, on behalf of humans. Then we describe a federation of the fully automatic web data extraction system ViPER [9] and the F-Logic [2] inference engine Florid [4] supporting post-processing and rule based reasoning on extracted information. We focus on Web information resources which structure their content according to a global layout template and present multiple similarly structured data records, which is a typical characteristic of static web catalogs as well as dynamic web pages. Due to the fact that these resources are often filled with information from back-end databases, our extraction and post-processing process can be seen as reverse engineering on the basis of materialized database views published within HTML pages. First, we start with the description of our fully automatic Web data extraction system which has already been embedded into a wide-spread Web browser for simple daily use. We explain how ViPER localizes and extracts structured data records and rearranges them into a tabular data structure. Based on this new representation we present several data-driven methods to discover correlations inside the resulting table columns which facilitates the transformation of the information into a semantically meaningful format.

Each intermediate transformation step becomes materialized using F-Logic syntax resulting in a final set of learned rules. Moreover, a user is able to interactively manipulate or generate additional rules in an extra post-processing step if desired. Once these rules have been learned on an initial data set, further similar structured information from web pages following the same layout template can be abstracted into a relational view. Additionally we are able to mine simple integrity constraints which become verified by Florid after the data cleaning process. Each of these steps will be illustrated later on by means of a running example.
This discussion demonstrates, that according to the structure of the web pages, fairly different sets of rules are constructed. Besides this the system is able to adapt itself automatically to changing layouts of Web pages, because the rules are generated without human intervention.

The paper is structured as follows: First we start with an overview of related work in section 2. Next we introduce in section 3 our web data extraction system ViPER. In section 4 we give deep insight into the preprocessing techniques of ViPER, such that its heuristics can be materialized into a set of Florid rules. We exemplify each discussed transformation heuristic referencing a real world web page allowing simple layout modifications and show the generic schema for the respective Florid rule. In section 5 we show an excerpt of the specific rules for the running example and discuss the benefits of the materialization of the heuristics with Florid rules. Finally we conclude in section 6.

2 Related work

In the last couple of years, as part of the Semantic Web activity, a number of different systems whose objective is to transform structured HTML pages into machine-processable content have been built.
TARTAR [6, 7] is most related to our work. The system extracts proper tabular structures from arbitrary tables, e.g. HTML tables, with the technique developed by [12]. Next it generates semantic F-Logic frames from rich nested table-like structure. The initial structure of the table is of great importance to derive the semantic interpretations of the content and describe the relationship between table cells, blocks and headers via F-Logic rules. The whole system could be described with the words: Table structure understanding.
Due to the fact, that we do not only focus on resources which layout their products in a rich nested table structures we have to build our own tabular representation of initially unstructured but similar data records. On base of this aligned table we are able to mine different relationships and describe them via Florid rules. Beyond this we rely on rendering information and also mine integrity constraints for cleansing new unseen information from the same resource. Especially, we aim at formally describing new similar rendered information in more detail with our derived rules after their automatic extraction.
Another system called TANGO (Table ANalysis for Generation Ontologies) [10] realizes an approach to generate ontologies based on table analysis. Based on an extraction ontology, TANGO attempts to understand a table‘s structure and conceptual content. Besides this TANGO tries to discover the constraints that hold among concepts extracted from the table which is the most related part to our approach. But we do not rely on an extraction ontology and our extraction system wraps content from web pages fully automatically without any human interaction in contrast to TANGO. Here hand-made regular expressions are used to extract information form web sites.
DeLA (Data Extraction and Label Assignment) [11] is an automatic wrapper system with post-processing functionalities. Following a user request the resulting web page becomes analysed and different extraction rules are suggested. The user can choose one of the extraction rules for which the system generates the appropriate table and labels the columns with information initially given by the request form and some simple heuristics. The main idea of the label process is that "form elements will probably reappear in the corresponding fields of the data objects" [11]. Therefore they rely in part on a differentiated input structure of the web page, where multiple form fields show a glimpse of the underlying structure of the web page. Our system is totally oblivious to the input structure and relies solely on the web page in question. Regardless of this DeLA does not generate additional rules that describe the structure of the page but is focused on generating a labelled table from web pages with form fields.
Finally, we comment on our own related work. In [5] we have studied information integration in general, emphasizing language constructs of Florid. In [8] we have informally described the table extraction-process of ViPER and a manual post-processing by rules and integrity constraints. In contrast, in the current paper the central topic is an in-depth discussion of the *materialization* of the table-extraction strategies of ViPER by rules using Florid. Finally, in [1] we presented the OntoGather system which dynamically integrates up-to-date information from the Web into a given background ontology with the help of Florid and ViPER.

3 Web Data Extraction and Alignment

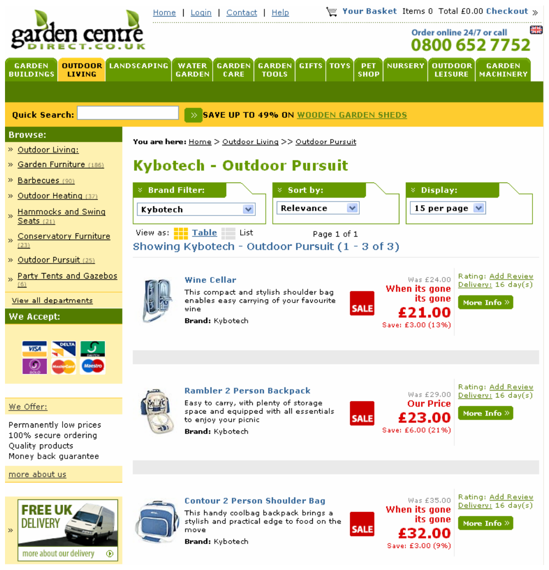
Wrapper tools for extracting information from HTML pages started attracting major research interest during the mid-nineties. One significant characteristic is their degree of automation, ranging from specially designed standalone wrapper programming languages, for manual wrapper generation, machine learning, and interactive approaches with more or less human interaction to fully-automatic wrapper tools. We refer the interested reader to [3] for a brief survey of different wrapping techniques. Whereas the majority of the approaches rely on user interaction to extract information, more recently, interest in automatically generated wrappers without human involvement has grown substantially.

3.1 Automatic Extraction

We developed a fully-automatic wrapper extraction tool named ViPER (**Vi**sual **P**erception-based **E**xtraction of **R**ecords). The original prototype controls the Gecko Engine of Mozilla over the XPCOM interface via the Java Framework JREX[1] which enables us to use the correction and rendering power of Gecko within Java. The principle assumption made is that a Web page contains at least two multiple spatially consecutive *data records* building a *data region* which exhibits some kind of structural and visible similarity. ViPER is then able to extract and discriminate the relevance of different repetitive information contents with respect to the user's *visual* perception of the Web page. Having identified the most relevant data region the tool extracts similar data records by declaring one data record contained in that data region as *pattern* and aligns matching data records utilizing a fast and robust *multiple sequence alignment* (MSA) technique. In [9] we already showed the accuracy of the extraction process by comparing our system with the results reported by existing fully automatic state-of-the-art wrapping tools.

Meanwhile, we have developed a slim plugin realization of ViPER for the Mozilla Firefox Web browser with additional interactive post-processing functionalities. Most of these functionalities pertain annotation modifications of the resulting table such as labeling of the aligned columns. On the other side the user has also the possibility to manually select different patterns from a ranked list of reported patterns after the extraction process. Some of these patterns could for instance match structured advertisements or links pointing to following pages.

Figure 1 depicts our running example on basis of which we like to illustrate both the extraction and rule mining process. More examples and their corresponding outputs can be found under the following link[2]. The web page in figure 1 contains three similar structured data records in the center of the page where each of them consists of the following *data items*: a linked picture of the product, a link labeled with the product name, a short product description, the brand name of the product, different price information with rating and shipping information. The screen shot of the web page shows the data records arranged in a vertical list. Additionally the web page provides a feature to change the layout of the data records and view them in a so-called "table view". By selecting this view a user is able to see more results at once without scrolling but with



Wine Cellar
This compact and stylish shoulder bag enables easy carrying of your favourite wine
Brand: Kybotech
Was £24.00
When its gone its gone
£21.00
Save: £3.00 (13%)
Rating: Add Review
Delivery: 16 day(s)

Rambler 2 Person Backpack
Easy to carry, with plenty of storage space and equipped with all essentials to enjoy your picnic
Brand: Kybotech
Was £29.00
Our Price
£23.00
Save: £6.00 (21%)
Rating: Add Review
Delivery: 16 day(s)

Contour 2 Person Shoulder Bag
This handy coolbag backpack brings a stylish and practical edge to food on the move
Brand: Kybotech
Was £35.00
When its gone its gone
£32.00
Save: £3.00 (9%)
Rating: Add Review
Delivery: 16 day(s)

Figure 1. Product catalog following a global layout template. The page is divided into several information regions such as navigation, banner, advertisement and product. We are focusing on the product records in the center of the page.

less information. The tabular layout is depicted in the upper left corner and the "list view" layout is located on the right side of figure 2. We have applied ViPER to extract information from both views resulting in a extraction pattern of 60 HTML elements for the list layout and 28 HTML elements in case of the "table view" whereas text content becomes abstracted by an artificial element depicted as <TEXT>. Within these elements only <A>, <IMG> and <TEXT> elements become displayed in the resulting extraction table described next, which finally sum up to at least 22 data items (list view) and 8 data items (table view). We have to mention that in the majority of cases the number of data items in the pattern does not determine the final number of columns because of additional data items belonging to data records matching the pattern with respect to the modified edit-distance computation as described in [9].

3.2 Automatic Alignment

After the extraction process the system attempts to align the data records resulting in a tabular representation. In [9] we described our partial data alignment approach based on global sequence alignment techniques, tree structure information and string similar-

[1][http://jrex.mozdev.org/]
[2][http://dbis.informatik.uni-freiburg.de/ViPER/Rex/Rules.html]

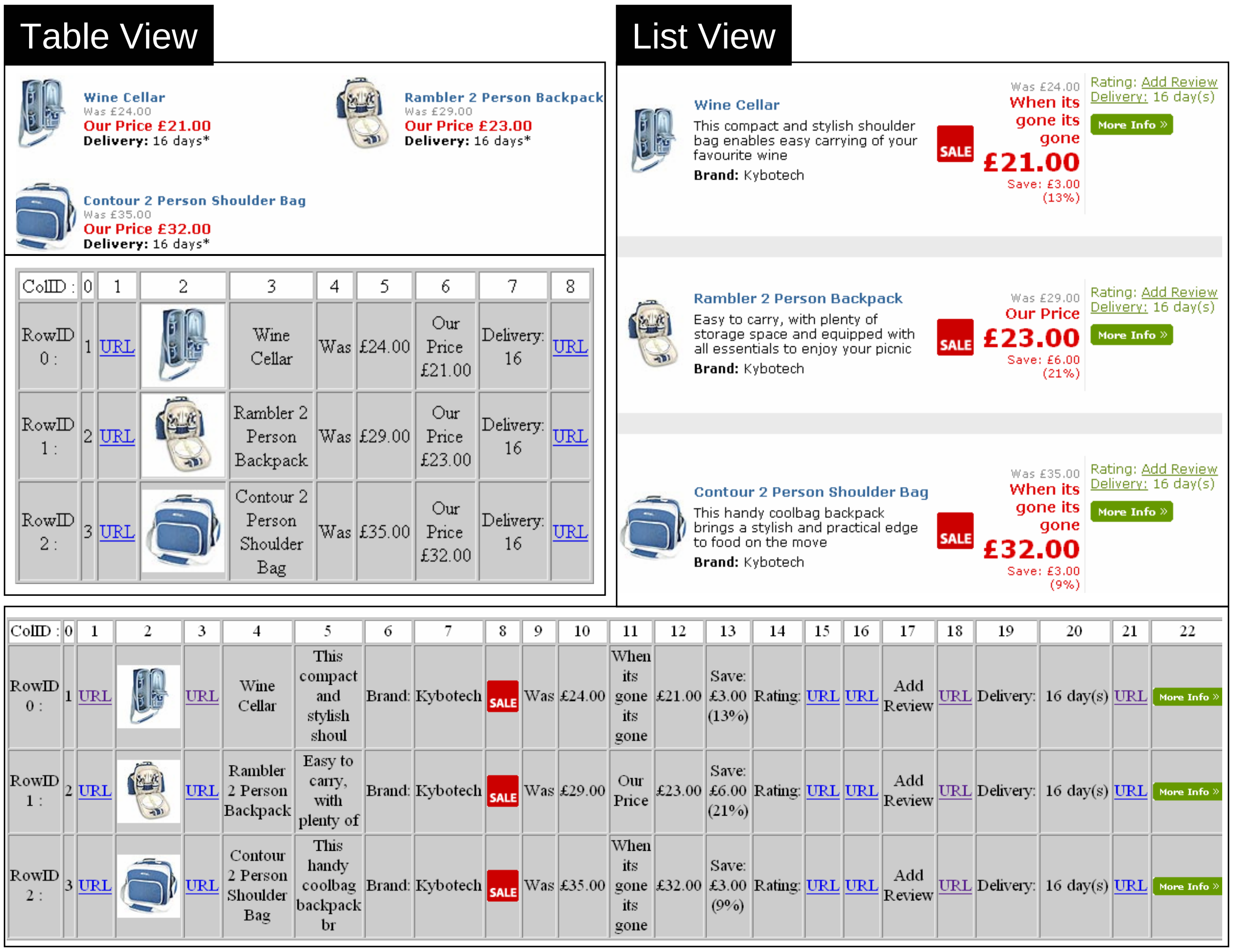


Table View

Wine Cellar
Was £24.00
Our Price £21.00
Delivery: 16 days*

Rambler 2 Person Backpack
Was £29.00
Our Price £23.00
Delivery: 16 days*

Contour 2 Person Shoulder Bag
Was £35.00
Our Price £32.00
Delivery: 16 days*

ColID :	0	1	2	3	4	5	6	7	8
RowID 0 :	1	URL		Wine Cellar	Was	£24.00	Our Price £21.00	Delivery: 16	URL
RowID 1 :	2	URL		Rambler 2 Person Backpack	Was	£29.00	Our Price £23.00	Delivery: 16	URL
RowID 2 :	3	URL		Contour 2 Person Shoulder Bag	Was	£35.00	Our Price £32.00	Delivery: 16	URL

List View

Wine Cellar
This compact and stylish shoulder bag enables easy carrying of your favourite wine
Brand: Kybotech
SALE
Was £24.00
When its gone its gone
£21.00
Save: £3.00 (13%)
Rating: Add Review
Delivery: 16 day(s)
More Info »

Rambler 2 Person Backpack
Easy to carry, with plenty of storage space and equipped with all essentials to enjoy your picnic
Brand: Kybotech
SALE
Was £29.00
Our Price
£23.00
Save: £6.00 (21%)
Rating: Add Review
Delivery: 16 day(s)
More Info »

Contour 2 Person Shoulder Bag
This handy coolbag backpack brings a stylish and practical edge to food on the move
Brand: Kybotech
SALE
Was £35.00
When its gone its gone
£32.00
Save: £3.00 (9%)
Rating: Add Review
Delivery: 16 day(s)
More Info »

ColID :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
RowID 0 :	1	URL		URL	Wine Cellar	This compact and stylish shoul	Brand:	Kybotech	SALE	Was	£24.00	When its gone its gone	£21.00	Save: £3.00 (13%)	Rating:	URL	URL	Add Review	URL	Delivery:	16 day(s)	URL	More Info »
RowID 1 :	2	URL		URL	Rambler 2 Person Backpack	Easy to carry, with plenty of	Brand:	Kybotech	SALE	Was	£29.00	Our Price	£23.00	Save: £6.00 (21%)	Rating:	URL	URL	Add Review	URL	Delivery:	16 day(s)	URL	More Info »
RowID 2 :	3	URL		URL	Contour 2 Person Shoulder Bag	This handy coolbag backpack br	Brand:	Kybotech	SALE	Was	£35.00	When its gone its gone	£32.00	Save: £3.00 (9%)	Rating:	URL	URL	Add Review	URL	Delivery:	16 day(s)	URL	More Info »

Figure 2. Two different layouts of almost the same data records of a shopping web page and their appropriate aligned raw tabular representation after the extraction process.

ity. The reported benchmarking results performed on a labeled third party data set underpin the accuracy and efficiency of our data alignment technique. With the aligned data records the storage in a relational database or export to XML is likewise easy. Within this paper we use the tabular arrangement of the results to mine specifics of the data records and derive logical rules.

The results of the alignment process received from our example web site is shown in figure 2. Starting with the "table view" in the upper left corner of the picture we see that each of the three data records corresponds to a row in the table and 8 data items have been arranged in adequate columns. Additionally, each row starts with an artifical column which enumerates the records. On the right side we see the "list view" of the results and in the big table at the bottom of the figure the corresponding extraction table is displayed. In contrast to the "table view" the records in the "list view" provides more information resulting in additional columns. Comparing the two tables the following links exist: column 0, 1, 2 in both table representation are the same and the column pairs (3,4), (4,9), (5,10) and (8,18) directly correspond to each other, where the first index refers to the "table view" and the second to the "list view" result table. Some columns only partially match, e.g.the column pair (6,10) and (7,(19,20)).

We show that despite of the different representations of the same data records we are able to derive rules producing the same final representation of the information and thus are insensitive to changes in the visual rendering.

4 Table Mining

In this section we describe a table mining process with the objective to bring the raw tabular representation of the previously extracted data records into a semantically meaningful and machine-processable format. Our approach is totally data-driven and sample-based on account of that we can only identify properties from the presented web view and have no access to schema information or catalog statistics.
Initially, we mine different types of rules that describe the dependencies and properties of the content in the aligned table columns. Simultaneously, we show that these rules are invariant with respect to simple layout modifications. To illustrate these two aspects we opt for our single web site, see figure 1, containing three data records with the additional ability to lay out these data records both as table or list. Please notice that the number of records in this example have been consciously kept small due to space restriction but in general the more data records a web site contains the better is the statistical significance of the resulting rules. The new feature for changing the layout has become very prominent in most modern shopping systems to give the user different compact representations she is most familiar with. In our case it is a good possibility to point out the invariance of the rule generation process with respect to simple layout modifications. Before we start with the explanation of the different mining states and their resulting rules we like to define some statistical dimensions describing the data contained in the table in more detail.

4.1 Preliminary Definitions

Let t define the resulting table after the alignment process and $K \times N$ its dimension where K denotes the total number of rows and N the total number of columns. A particular data item in the i-th row and the j-th column is addressed by $t.s_{i,j}$ or shortly $s_{i,j}$ with $1 \leq i \leq K, 1 \leq j \leq N$. The short notation is practical because in our case we always refer to the same relation t. A common notation to refer to a special entry contained in a row or a column is $R_i := s_{i,\cdot}$ or $C_j := s_{\cdot,j}$, respectively.
This table can be represented in Florid as follows: each row, i.e. data record is modelled as an instance of the class *dataRecord*, which contains a *functional method* that identifies the corresponding web resource. These data record objects manage a set of data items, corresponding to the $s_{i,j}$ for some fix i and $1 \leq j \leq N$, realized in Florid as *multi-valued* method *dataItems*. Each entry in the set of data items is an instance of the class *dataItem* and initially has two *functional methods*: *colID* which identifies the column j and *val* which points to the actual value of the data item[3]. After the labeling step an additional functional method *label* is present that identifies the column label (see example (4.1) and (4.3) in table 2 for examples).

$$\begin{aligned} &item_{i,j} : dataItem[colID \rightarrow j; \\ &\qquad val \rightarrow'' value'']. \\ &dataRecord_i[webResource \rightarrow resourceID; \\ &\qquad dataItems \twoheadrightarrow \{item_{i,1}, \ldots, \\ &\qquad\qquad item_{i,j}, \ldots, \\ &\qquad\qquad item_{i,N}\}]. \end{aligned} \tag{4.1}$$

Next we define some statistical dimensions describing the table content in more details. If we focus on a specific column C_j with $D_j \leq N$ *distinct data items*, say $\mathcal{D}_j := \{1, 2, ..., D_j\}$, then $n_{k,j}$ denotes the *number of times* data item $k \in \mathcal{D}_j$ occurs in the j-th column. With *total sum* $n_j := \sum_{k=1}^{D_j} n_{k,j}$ where the *relative frequency* of data item k corresponds to $p_{k,j} := n_{k,j}/n$, with property $\sum_{k=1}^{D_j} p_{k,j} = 1$. Additionally we rank the items according to their frequency $n_{k,j}$ and denote by $v_{k,j}$ the k-th frequent data item in column j.
For each data item of type <TEXT> we analyse the item on the text level. In addition to this the text content of a data item becomes tokenized into an ordered set of tokens with attached data types. This is illustrated by the tree on the left side of figure 3 which belongs to the first data item of column 13 from the table at the bottom of figure 2. At the root of the tree we see the initial data item. The first level of the tree has been generated by tokenizing the initial text according to white spaces. If a token on the first level consists of mixed content with numeric subparts then this token will be tokenized until no more separations of numeric parts could be carried out. Each level of the resulting tree gives a finer granulated insight into the text content of the data item. Finally, we assign signature information as leaf nodes. At the moment of writing, the system recognizes simple and some specific data type signatures, like for instance currencies and date/time formats. Additionally we distinguish the type punctuation which becomes assigned to tokens having one character such as "-", ",", "(", and "/".

4.2 Splitting rules

In the first stage of our table mining process we identify columns which could be splitted into several sub-columns. The reason is that we would like to retrieve information that is hidden in structured text

[3]Note, that in F-Logic and Florid variables always start with a capital letter. Any identifier starting with a lower level letter denotes an obect, i.e. constant, method, class, etc.

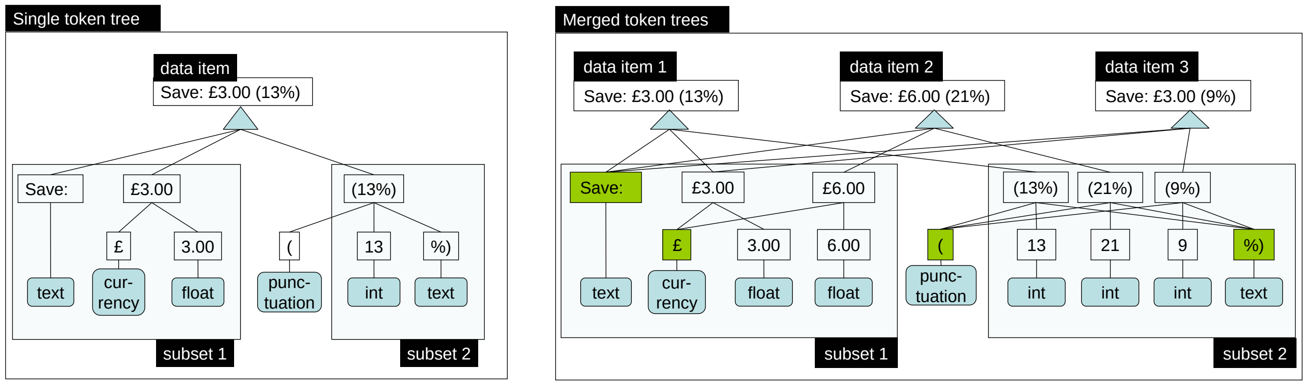


Figure 3. A single data item and its token tree. Each level in the tree offers a finer granulated insight into the text content. The leaf nodes of the tree describe the data type of its preceding parents. On the right side three data items taken from the same column of the resulting table with their aligned token trees are shown. Here the token of type punctuation splits the content of the data items into two new subsets resulting in two sub-columns. Fix elements contained in each of data items are the marked tokens: "Save:", "£", "(" and "%)".

like e.g. comma separated lists. With the tabular arrangement of the data records and the tokenized tree representation of each <TEXT> item we are able to explore every data item and compare these with other items belonging to the same column.

4.2.1 Heuristic

The heuristic iterates over each aligned column containing <TEXT> data items from the result table separately. If for instance the number of distinct data items D_j in column C_j is greater than one we check each token tree $T_{i,j}$ for punctation tokens and their frequency. Next we take the least common number of tokens appearing in all token trees having the same content (subset over all punctuations found in the column). If the intersection of shared punctuation tokens is not empty, then the token trees become aligned according to these matching tokens. Each subtree between the aligned tokens results in a new sub-column.
For instance in case of our example on the right side of figure 3 each data item contains the punctuation token '('. As a result the current column number 13 becomes splitted into two new sub-columns 13.1 and 13.2.

4.2.2 Florid rule

The heuristic can be materialized according to the generic schema depicted in example (4.2) in table 1. Here the variable *DataRecord* on the right side of the rule, i.e. after the $:-$, iterates over all data records of the table, where the variable *Item* specifies the j^{th} data item in each data record. This data item is splitted into 1 to Max new data items - specified by $Item.1$ to $Item.Max$ on the left hand side of the rule - with the help of a *regular expression*, that is automatically generated by ViPER.

4.3 Labeling rules

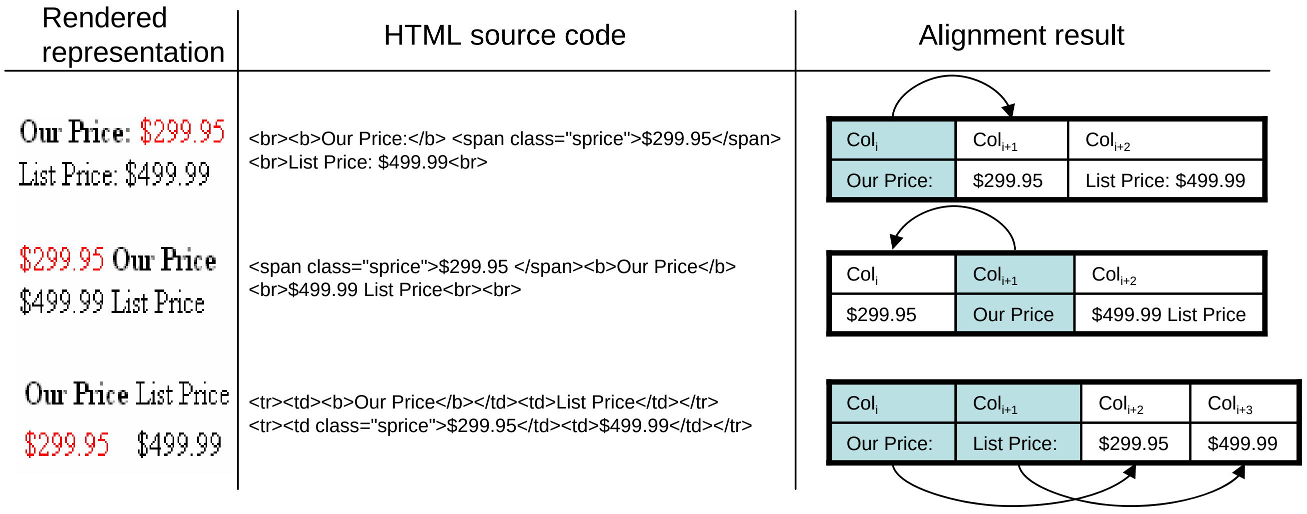
In the second stage we like to assign column labels to data items. Data records itself often contain labels which have been assigned "next to", with respect to the rendered HTML representation, variable data items. These typical characteristic reflects in columns where each row contains the same data item, which we will refer to as *fix columns*. The interpretation of such a fix column therefore depends on the layout structure. We handle this property with our *inter-column* label assignment heuristics. On the other hand with aid of the tabular data record rearrangement we are able to build statistics over tokens from the same column. Tokens which appear in each of the data items from the same row are of great importance and might stand for a label of the column. This property is handled by our *inner-column* label assignment heuristics introduced next.

4.3.1 Inter Column Label Assignment

Figure 4 illustrates an example of three typical types of fix columns and their interpretation handled by our heuristics. Scanning top-down through these examples

$$\begin{aligned}&DataRecord[dataItems \twoheadrightarrow \{Item.0 : dataItem[colID \rightarrow j_1; val \rightarrow Input_1], \ldots,\\&\qquad\qquad Item.Max : dataItem[colID \rightarrow j_{Max}; val \rightarrow Input_{Max}]\}] :-\\&DataRecord[webResource \rightarrow resourceID; dataItems \twoheadrightarrow \{Item[colID \rightarrow j; val \rightarrow Input]\}],\\&pmatch(Input, ''/regular\ expression/'', [''\$1'', \ldots, ''\$Max''], [Input_1, \ldots, Input_{Max}]).\end{aligned} \tag{4.2}$$

Table 1. Splitting rule with regular expression described in Florid syntax.


Rendered representation
HTML source code
Alignment result
Our Price: $299.95
List Price: $499.99
Col_i | Col_i+1 | Col_i+2
Our Price: | $299.95 | List Price: $499.99
$299.95 Our Price
$499.99 List Price
Col_i | Col_i+1 | Col_i+2
$299.95 | Our Price | $499.99 List Price
Our Price List Price
$299.95 $499.99
Col_i | Col_i+1 | Col_i+2 | Col_i+3
Our Price: | List Price: | $299.95 | $499.99

```
<br><b>Our Price:</b> <span class="sprice">$299.95</span>
<br>List Price: $499.99<br>
```

```
<span class="sprice">$299.95 </span><b>Our Price</b>
<br>$499.99 List Price<br><br>
```

```
<tr><td><b>Our Price</b></td><td>List Price</td></tr>
<tr><td class="sprice">$299.95</td><td>$499.99</td></tr>
```

Figure 4. Heuristics to label data items have to rely on render information. We capture three different kinds of data item arrangements. The above figure illustrates these by showing the rendered browser representation (left) of their corresponding HTML tag sequence (middle) and aligned table representation (right) for each of the three cases. A column with fix content (gray), is a potential label and becomes assigned (arrows) with respect to the render information to the spatially closest column with variable content.

we distinguish the following label assignment strategies: left-to-right, right-to-left and in case of vertical orientations up-to-down assignment. We do not consider the fourth orientation down-to-up because of its rareness.
The first orientation especially holds for documents encoded in (western) languages on account of the reading direction. In this case the aligned fix content (gray) can be easily assigned to the next column with presumable variable content (arrow).
Otherwise, if the label is positioned on the opposite side of the variable content then we also have to assign the aligned fix content (gray) to its predecessor column. These label notations are often used for units. An indication which of these two assignment directions have to be applied is realized by a decision tree.
First we test whether the surrounding data items have been rendered in the same line, with respect to the bounding box information. Provided that they are in the same horizontal line we next test if there exists a special separation character at the end of one of the strings, e.g., the colon in "Our Price:" from our first example. A final rule considers the complete row. Thereby the assignment direction which have been applied most times inside the row takes place. If non of these rules decide the assignment problem we leave the column untouched.
A typical characteristic of the up-to-down label assignment is that inside the aligned table a number of fix columns appear next to each other. To map the labels to the corresponding data items we search for the spatially closest located variable column in the render representation and thus assign the consecutive fix columns from left to right as depicted in the last example of figure 4.
The second table in figure 2 of our running example has fix <TEXT> columns at position 6, 7, 9, 14, 17, 19 and 20 which contain the data items "Brand:", "Kybotech", "Was", "Rating:", "Add Review", "Delivery:" and "16 day(s)". Some of them end with colons and others do not. With the additional render information the heuristic assigns the content of column 6, 14 and

19 ending with colons to their adequate right neighbor column as label. Next each remaining fix column not already having a label becomes assigned according to its render information. Thus, column 9 gets assigned as label to column 10 and column 17 which is the title of the link from column 16 becomes assigned to this column. Please note that despite of the fact that column 19 is the title of the link of the previous column its content gets assigned to column 20 due to the colon indicating that the content this data item describes follows to the right.

4.3.2 Inner Column Label Assignment

Next each remaining column becomes scanned for fix non-numerical sub-tokens appearing in each of their merged token trees. For instance in figure 2 the data items of column 10 and 12 contain the currency character "'£". Also the two new sub-column 13.1 and 13.2 of the splitted column 13 contain the fix sub-tokens "Save:", "'£" and the mixed text content "%)" which have been marked in subset1 and subset2 of figure 3. Additionally, column 20 contains the fix non numeric token "day(s)". Each fix token which is not of numeric type becomes removed in the individual rows and added as label.

4.3.3 Florid rule

The generic schema for an inter-column-label assignment rule ist depicted in table (4.3) and can be described as follows: the variable *Data Record* on the right hand side of the rule iterates over all data records of the table. The value of each data item in column j_2 is then assigned as label for the data item in column j_1.
The generic schema for an inner-column-labeling rule is depicted in example (4.4) in table 2. Again the variable *DataRecord* iterates over all data records and the value of each data item in column j gets overwritten with the matching result of a regular expression that removes the fix content. Two things should be mentioned here: first the label that is assigned to the respective column is obtained directly from the ViPER heuristics and second the real Florid rules are more complex, due to the fact that the result of a functional method is not allowed to change after its initialization. We circumvent this problem at the moment by introducing temporary variables until the final assignment takes place (see example (4.5) in table 3 for an example) but we envision to implement an update predicate in the Florid system to provide a more natural solution for this issue.

4.4 Data consistency constraints

In almost the same manner as schema definitions of relational databases allow the definition of integrity constraints we like to learn these kind of constraints from small instances. The more instances we have the better the quality of the constraints. One constraint which becomes feasible with the new representation of the raw data records is to find simple arithmetic constraints between columns containing only numeric data items.

4.4.1 Heuristic

With respect to the computational complexity we limit the mining process to equations of the type

$$C_z = \lambda_1 C_x + \lambda_2 C_y \qquad (1)$$

$$C_z = \lambda_1 C_x \qquad (2)$$

where $\lambda_i \in \mathbb{R}$ with $i \in 1, 2$. To get information about the relations between numeric columns we check the homogeneous system of linear equations for non-trivial solutions. In case we have a solution and thus linear dependent columns we try to describe each of these columns by a linear combination of at most two other columns as described by the upper equations.

4.4.2 Florid rule

Since Florid is able to deal with integer and float numbers, we are able to verify numeric constraints between different columns. The example shown in table (4.6) and (4.7) checks if the arithmetic constraint $C_{j_1} = \lambda_1 C_{j_2} + \lambda_2 C_{j_3}$, with $\lambda_1 = -1$ and $\lambda_2 = 1$ holds for all data items. The principal schema of the rules is similar to the ones already discussed, with the exception that two similar rule heads are present. This means that it is sufficient that one of the bodies of the rules in example (4.6) and (4.7) in table 4 is true to satisfy the head, i.e. the arithmetic constraint. Both bodies are identical except the last line, that checks two distinct cases that can occur when evaluating the equation $C_{j_3} - C_{j_2} - C_{j_1}$, which is due to the underlying representation of floating point numbers in modern computers. Either the result is greater or equal than 0, then we require it to be smaller than a reasonable small value ϵ or it is smaller than 0, in which case we require it to be greater than the negative value of ϵ.

5 Application of learned rules

After an initial training step were we materialize the abovementioned post-processing heuristics in

$$\begin{aligned}
&Item_1[label \rightarrow Label] :- DataRecord[webResource \rightarrow resourceID; \\
&\qquad\qquad dataItems \twoheadrightarrow \{Item_1, Item_2\}], \\
&\qquad Item_1[colID \rightarrow j_1], \\
&\qquad Item_2[colID \rightarrow j_2, val \rightarrow Label].
\end{aligned} \tag{4.3}$$

$$\begin{aligned}
&Item[label \rightarrow'' Label''; val \rightarrow NewValue] :- DataRecord[dataItems \twoheadrightarrow \{Item\}], \\
&\qquad\qquad Item[colID \rightarrow j; val \rightarrow Value], \\
&\qquad pmatch(Value,'' /regular\ expression/'', \\
&\qquad\qquad [''\$1 \ldots \$Max''], [NewValue]).
\end{aligned} \tag{4.4}$$

Table 2. The two different types of labeling rules in Florid syntax. The first rule describes an inter-column-label assignment rule from column j_2 to j_1. The second rule is a inner-column-labeling rule where the fix non-numeric content of column j gets removed and added as a label to itself.

$$\begin{aligned}
&Item[tmp_1 \rightarrow NewValue_1] :- DataRecord[dataItems \twoheadrightarrow \{Item\}], \\
&\qquad\qquad Item[colID \rightarrow j; input \rightarrow Value], \\
&\qquad pmatch(\ldots, [NewValue_1]). \\
&\ldots \\
&Item[tmp_{k-1} \rightarrow NewValue_{k-1}] :- DataRecord[dataItems \twoheadrightarrow \{Item\}], \\
&\qquad\qquad Item[colID \rightarrow j; tmp_{k-2} \rightarrow Value], \\
&\qquad pmatch(\ldots, [NewValue_{k-1}]). \\
&Item[val \rightarrow NewValue_k] :- DataRecord[dataItems \twoheadrightarrow \{Item\}], \\
&\qquad\qquad Item[colID \rightarrow j; tmp_{k-1} \rightarrow Value], \\
&\qquad pmatch(\ldots, [NewValue_k]).
\end{aligned} \tag{4.5}$$

Table 3. Florid rules for k updates of the result of a functional method

$$\begin{aligned}
&arithmetic\_constraint :- DataRecord[dataItems \twoheadrightarrow \{Item_1, Item_2, Item_3\}], \\
&\qquad Item_1[colID \rightarrow j_1; value \rightarrow Value_1], \\
&\qquad Item_2[colID \rightarrow j_2; value \rightarrow Value_2], \\
&\qquad Item_3[colID \rightarrow j_3; value \rightarrow Value_3], \\
&\qquad Check = Value_3 - Value_2 - Value_1, Check < \epsilon, Check >= 0.
\end{aligned} \tag{4.6}$$

$$\begin{aligned}
&arithmetic\_constraint :- DataRecord[dataItems \twoheadrightarrow \{Item_1, Item_2, Item_3\}], \\
&\qquad Item_1[colID \rightarrow j_1; value \rightarrow Value_1], \\
&\qquad Item_2[colID \rightarrow j_2; value \rightarrow Value_2], \\
&\qquad Item_3[colID \rightarrow j_3; value \rightarrow Value_3], \\
&\qquad Check = Value_3 - Value_2 - Value_1, Check > -\epsilon, Check < 0.
\end{aligned} \tag{4.7}$$

Table 4. Arithmetic constraint for the case $C_{j_1} = \lambda_1 C_{j_2} + \lambda_2 C_{j_3}$, with $\lambda_1 = -1$ and $\lambda_2 = 1$.

Florid rules, we can directly utilize Florid for the post-processing step. This way, we have full control of the application of the heuristics and can decide which rules we want to apply for the Web resource in question. Another benefit is, that we can increase the accuracy of the extraction process by editing the rules appropriately where necessary. To make use of this capability, ViPER is able to transform the HTML page into an initial tabular Florid representation that makes it amenable to the application of the learned rules. Tables 5 and 6 show an excerpt of the list representation of our running example with the accompanying rules.

6 Conclusion

In this paper we discussed the materialization of the post-processing heuristics of ViPER via Florid rules. By making these heuristics explicit we are able to control our extraction system on a fine-grained level, e.g. change the automatically-assigned labels derived by the heuristics, while retaining the convenience that the initial rules are derived automatically. Another benefit of the materialization is that we are now able to verify data consistency constraints, that have been identified by ViPER, afterwards with Florid.
Right now we are investigating how to automatically derive additional data consistency constraints, like functional constraints known from classical databases and the integration of the materialized rules into the OntoGather prototype described in [1] which dynamically integrates up-to-date information from the Web into a given background ontology with the help of Florid and ViPER.

References

[1] T. Hornung, K. Simon, and G. Lausen. Information Gathering in a Dynamic World. In *Principles and Practice of Semantic Web Reasoning 2006*, Budva, June 2006. (to appear).

[2] M. Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-based Languages. *Journal of the ACM JACM*, 42(4):741–843, 1995.

[3] A. H. F. Laender, B. Ribeiro-Neto, A. S. D. Silva, and J. S. Teixeira. A Brief Survey of Web Data Extraction Tools. *ACM SIGMOD Record*, 31(2):84–93, 2002.

[4] B. Ludascher, R. Himmeroder, G. Lausen, W. May, and C. Schlepphorst. Managing Semistructured Data with FLORID: A Deductive Object-Oriented Perspective. *Information Systems*, 23(8):589–613, 1998.

[5] W. May and G. Lausen. A Uniform Framework for Integration of Information from the Web. In *Information Systems*, volume 29, pages 59–91, 2004.

[6] A. Pivk, Y. Sure, P. Cimiano, M. Gams, V. Rajkovi, and R. Studer. From Tables to Frames. Technical report, AIFB, 2004.

[7] A. Pivk, Y. Sure, P. Cimiano, M. Gams, V. Rajkovi, and R. Studer. Transforming Arbitrary Tables into F-Logic Frames with TARTAR. In *Elsevier Science*, 2005.

[8] K. Simon, H. Boley, and G. Lausen. From HTML Documents to Tables and Rules. In *8th International Conference on Electronic Commerce (ICEC 2006)*, Fredericton, New Brunswick, Canada, August 2006. (to appear).

[9] K. Simon and G. Lausen. ViPER: Augmenting Automatic Information Extraction with Visual Perceptions. In *Proceedings of the 2005 ACM CIKM Conference on Information and Knowledge Management*, pages 381–388, Bremen, GERMANY, November 2005. ACM Press.

[10] Y. Tijerino, D. Embley, D. Lonsdale, Y. Ding, and G. Nagy. Towards Ontology Generation from Tables. *World Wide Web*, 8(3):261–285, September 2005.

[11] J. Wang and F. H. Lochovsky. Data Extraction and Label Assignment for Web Databases. In *World Wide Web Conference*, pages 187 – 196, May 20-24 2003.

[12] Y. Wang and J. Hu. A Machine Learning Based Approach for Table Detection on the Web. In *World Wide Web Conference*, pages 242–250, 2002.

```
...
item_1_10 :  dataItem[colID→''10''; input→''£24.00''].
...
item_1_12 :  dataItem[colID→''12''; input→''£21.00''].
item_1_13 :  dataItem[colID→''13''; input→''Save:  £3.00 (13%)''].
...
dataRecord_0[webResource→42; dataItems↠item_1_1, ..., item_1_22].
```

Table 5. Excerpt of list representation of the running example

```
% Splitting Rules
DataRecord[dataItems↠{Item.1:dataItem[colID→''13.1''; input→Value1],
                       Item.2:dataItem[colID→''13.2''; input→Value2]}] :-
DataRecord:[webResource→42;
            dataItems↠{Item[colID→''13''; input→Input]}],
pmatch(Input,''/(.*)\s*\(\s*(.*)/'', [''$1'',''$2''], [Value1,Value2]).
% Labeling Rules
...
Item[label→''Was £''; tmp →Value] :-
_:[webResource→42; dataItems↠{Item}], Item[colID→''10''; input →OldValue],
pmatch(OldValue,''/(.*)\s*£\s*(.*)/'', [''$1 $2''], [NewValue]).
Item[label→''£''; tmp →NewValue] :-
_:[webResource→42; dataItems↠{Item}], Item[colID→''12''; input→OldValue],
pmatch(OldValue,''/(.*)\s*£\s*(.*)/'', [''$1 $2''], [NewValue]).
Item[label→''Save:  £''; tmp →NewValue] :-
_:[webResource→42; dataItems↠{Item}], Item[colID→''13.1''; input→OldValue],
pmatch(NewValue,''/(.*)\s*Save:\s*(.*)\s*£\s*(.*)/'', [''$1 $2 $3''], [NewValue]).
Item[label→''%)''; tmp →NewValue] :-
_:[webResource→42; dataItems↠{Item}], Item[colID→''13.2''; input→OldValue],
pmatch(OldValue,''/(.*)\s*\%\)\s*(.*)/'', [''$1 $2''], [NewValue]).
...
% Signature Rules
...
Item[val⇒float] :- _:[webResource→42; dataItems↠Item], Item[colID→''10''].
Item[val⇒string] :- _:[webResource→42; dataItems↠Item], Item[colID→''11''].
Item[val⇒float] :- _:[webResource→42; dataItems↠Item], Item[colID→''12''].
Item[val⇒float] :- _:[webResource→42; dataItems↠Item], Item[colID→''13.1''].
Item[val⇒integer] :- _:[webResource→42; dataItems↠Item], Item[colID→''13.2''].
...
% Arithmetic dependency mining
constraint_1() :- _:[webResource→42; dataItems↠Item1,Item2,Item3],
                    Item1[colID→''12''; val→Val1], Item2[colID→''13.1''; val→Val2],
                    Item3[colID→''10''; val→Val3], Diff = Val3 - Val2 - Val1,
                    Diff <= #0.001, Diff >= 0.
constraint_1() :- _:[webResource→42; dataItems↠Item1,Item2,Item3],
                    Item1[colID→''12''; val→Val1], Item2[colID→''13.1''; val→Val2],
                    Item3[colID→''10''; val→Val3], Diff = Val3 - Val2 - Val1,
                    Diff >= -#0.001, Diff < 0.
```

Table 6. Excerpt of transformation rules for list representation of the running example