Interactive Learning of HTML Wrappers Using Attribute Classification*

Michal Ceresna

DBAI, TU Wien, Vienna, Austria
ceresna@dbai.tuwien.ac.at

Abstract. Reviewing the current HTML wrapping systems, it is possible to recognise two mainstream categories. The first category are systems based on various machine learning techniques with lower roll-out and maintenance costs, but reaching worse results and usually being specialised to particular domains. The second category are systems which allow to build more complicated wrapping solutions. But here a human wrapper designer is required to build and maintain the wrappers. Therefore, it increases costs for acquiring of the information.
In this paper we apply machine learning techniques to automate and simplify building of the HTML wrappers by the wrapper designer. We present a learning algorithm that creates wrappers from interaction with an human designer. She only marks positive and negative example instances inside of the currently rendered Web and an HTML wrapper is generated from this interaction. The learning algorithm presented in this paper is based on clustering of the example instances with respect to the similarity of their tree shape and on classification of HTML attributes inside of each cluster.

1 Introduction

Commonly adopted strategies for accessing of the data located on Web pages are technologies known as Web information extraction or HTML wrapping. They turn interesting data in Web pages into formats suitable for further machine processing such as XML [1], data models of integrated applications [13] or relational databases [10].

Over the time various approaches of building the HTML wrappers have been researched. Chronologically, the following mainstreams can be recognised: hand coded wrapping programs [7], inductive learning of string based automata [8], semi-automatic visually guided construction [1] and machine learning techniques using support vector machines [10] or Markov models [9].

In this paper we focus to approaches that represent Web pages in form of a DOM tree [11]. DOM trees are constructed using parsers from modern Web browsers such as Mozilla and Internet Explorer. The extraction process is then

* This work is partially supported by the GAMES Network of Excellence of the European Union.

Vojtěch Svátek, Václav Snášel (Eds.): RAWS 2005, pp. 25–32, ISBN 80–248–0864–1.

executed on the DOM tree instead of the HTML source code. Having a DOM tree, the HTML wrapping can be viewed as identification of the relevant parts, for example subtrees in the input DOM tree. The interesting data chunks to be extracted from a DOM tree are called *instances*. The usually extracted types of instances are: sets of tree nodes in the given input DOM tree, substrings from text content nodes or values of tree element attributes. Examples of the various instance types are in the Figure 1 highlighted with green background.

Fig. 1. Different types of instances extracted from Web pages

Operating on the DOM-tree, various techniques are used to express how to extract the interested instances from the given Web page. We have identified the following existing approaches: Datalog-like programs evaluated over tree domains [1], finite (tree) automata [2,4], XPath and XQuery node selection queries [5,3]. None of these approaches is fully automated. Either the user has to construct the wrapper manually with a visual assistance of the wrapper system [1,5] or only tree structures are learned, but with difficulty of handling various kinds of HTML attributes and text contents [2,4].

This paper is structured as follows. In Section 2 we give an overview of the interactive wrapper generation. Section 3 is devoted to the wrapper induction and extraction using the induced wrapper. Section 4 discusses experimental results and highlights future research directions.

2 Interactive Wrapper Generation

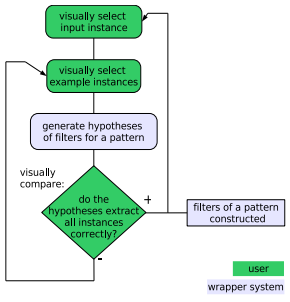
Human beings tend to assign semantic meaning to parts of a Web page. Human designer does not think of HTML *table* as of a tree with some text values, but rather as of a list with book entries that contain book title, author name and ISBN number. Therefore, the basic building block of our wrappers are a so-called patterns, containers for pieces of information (instances) with the same meaning. Patterns are hierarchically organized into a tree structure. That is, except of the top-level pattern, has each pattern exactly one parent pattern. An example of the hierarchical pattern structure is illustrated in the Figure 2.

Each pattern contains from one or more filters that define how to extract the relevant instances for this pattern. Filter is an unary function that returns a set of output instances extracted from a given input instance. A set of instances extracted for a pattern is then an union of instances extracted by all its filters. Inputs of these filters are instances of the parent pattern or the whole example HTML document in case of the top-level pattern.

Fig. 2. Pattern structure of a wrapper

We present a learning algorithm that shifts the task of constructing the filters from the user to the wrapping system. Wrappers induced by this algorithm still preserve the expressiveness of manually constructed HTML wrappers. Our interactive wrapper generator creates filters from a visual interaction with a human wrapper designer. The user is asked to mark via mouse clicks positive and negative example instances inside of the rendered HTML page. Such interaction is then technically equivalent to the marking of nodes in the DOM tree. The goal of the interaction is to help the system to discover the filters that correctly identify all the desired patterns. The process of the interaction is outlined as a loop of the following steps:

1. Select an example input instance (or whole HTML document).
2. On this input mark missing instances not yet recognised by the system or drop instances that were incorrectly identified by the system.
3. When the system correctly matches all of the intended instances inside of the current input, user decides whether to continue with training/testing on another input instance or HTML document.



3 Wrapping Induction with Attribute Classification

3.1 Clustering of example instances

In this section we present a wrapper learning algorithm using clustering and attribute classification. We will demonstrate the learning algorithm on the Web page displayed in Figure 3 which allows us to show the interesting properties of this algorithm. Inside of the example Web page, we are interested to extract city names ('Vienna', 'Prague', 'Bratislava') and the contact links with the black text colour ('contact1', 'contact3', 'contact5'). The set of positive and negative instances received so far from the user is marked with green, respectively with red background.

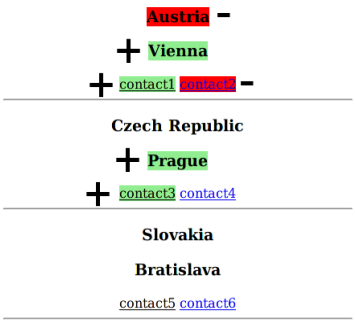


Fig. 3. Example Web page for attribute classification

During the training process the wrapper learning algorithm proceeds with the following steps:

1. Sort example instances into clusters with respect to similarity of their tree structure.
2. For each cluster build the list of features used for classification. The list is computed from attributes and their values that are contained in the training examples.
3. Build the training dataset.
4. For each cluster build a decision tree based attribute classifier.

The clustering of example instances according to the similarity of the tree structure is implemented using the nearest neighbourhood grouping. As distance metrics for two DOM trees is used the tree edit distance [12]. During the clustering process a DOM tree becomes pivot of a new cluster, if its distance from pivot trees of all existing clusters oversteps above a chosen threshold. In that case, a new cluster is created and the existing instances are regrouped to their closest

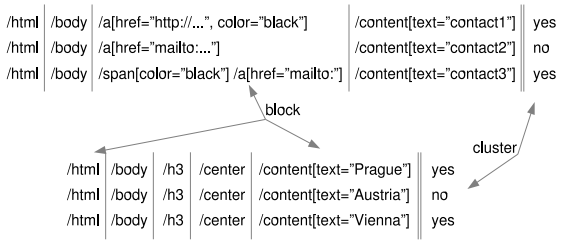


Fig. 4. Clustering of example instances

Algorithm 1 Extraction process using attribute classifiers

```
for cluster in clustering
  instances = evaluateXPath(cluster.xpath, INPUT_DOCUMENT)
  for instance in instances:
    testExample = buildTrainOrTestExample(cluster.featuresDef, instance)
    if cluster.classifier.classify(testExample)=='yes'
      # add 'instance' to matched instances
```

cluster. The Figure 4 contains a clustering of the instances from our example Web page depicted in Figure 3.

Each cluster is divided into blocks. Essentially each DOM element forms its own block, just for some types of the DOM elements, we use the built-in knowledge of HTML and merge several elements into a single block. Examples of elements merged into one block are <table>/<tbody>, <p>/ or /#text.

In the presented approach pattern contains a set of filters. Each cluster defines one extraction rule which is a pair consisting of a CoreXPath expression [6] and an attribute classifier. The XPath expression is used to find DOM nodes (instances) inside of the input DOM tree matching the particular tree shape of the cluster. Subsequently, the attribute classifier is used to sort out the instances with incorrect attributes and properties. Extraction process using this type of filter definitions is outlined in the Algorithm 1.

3.2 Building Features

Building of a decision tree requires to know which features are available and what are the possible values for these features. Therefore, before training of the DOM attribute classifier we construct a list of interesting features considered by the classifier.

The list of the features is built from the set of DOM attributes and text node values contained in the cluster. The same DOM attribute may repeat in several blocks of the same cluster with different values, as happens with the 'bgcolor' attribute in the instance /table[bgcolor='red']/tr/td[bgcolor='green']. Therefore, each feature name contains an index of the block it belongs to, for

example 'BL3_href_val'. DOM attributes inside of the same block are treated as non-repeating. That means, their values are unified into a common set.

Not all HTML attributes have equal semantics. Therefore, it does not make sense to map every DOM attribute one-to-one into a training feature. For example, our feature selection considers only attributes that have rendering effect in HTML. Because the user highlights the interested instances inside of a rendered Web page, she can not distinguish DOM nodes that differ only with an attribute that does not influent rendering. Therefore, we sorted the HTML attributes into three categories (Table 1), depending on their rendering effect.

none	by presence	by each value
alt	multiple	align
codebase	readonly	border
id	checked	color

Table 1. HTML attributes and their influence of rendering

The list of possible values for each feature is constructed in the following way. Possible values of a feature constructed from a DOM attribute with rendering influence by presence are 'present' and 'absent'. Set of possible values for a feature constructed from a DOM attribute with rendering changed by each value is union of all values of this attribute collected in the current block over all example instances.

Each feature has two additional possible values with special meaning. The '!missing' value is used during building of the training data to express that the current example instance does not contain a particular attribute in a block, while other instances contain this feature. The value '!other' is used during the extraction process to express that an instance has a particular attribute, but with a value that has not been known during the training process.

We extended our feature list also with ontology-based features know already from other wrapping systems [1]. From content of the text nodes we compute boolean features of syntactic concepts such as date, year, number, currency and semantic concepts such as city, country, continent. List of possible values for these concept-based features is 'yes' and 'no'. The Figure 5a contains the features chosen for training from the example in the Figure 3.

3.3 Training attribute classifier

Building of the training set is done by iterating over all instances in the cluster and computing values of the previously constructed features. As indicated already earlier, there is a case possible, when value of a feature can not be computed for an example instance. This may happen, when a DOM element in the example instances does not contain the needed DOM attribute. In that case the special value '!missing' is used.

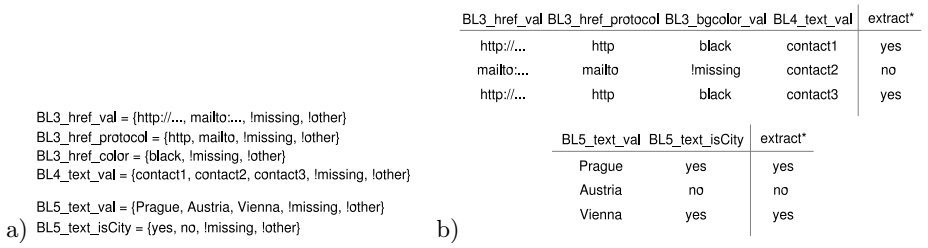


Fig. 5. a) Feature list for example instances b) Training set constructed from example instances

Moreover, a special target feature called 'extract*' is added to each training example. The possible values of that feature are 'yes' or 'no', depending whether the processed example instance is a positive or a negative example. The Figure 5b contains the training set constructed from the example in Figure 3.

Prediction of the target feature 'extract*' is then trained by the decision tree classifier. The constructed list of features and the dataset are fed into the training algorithm that builds the classifier. For implementation of our learning algorithm we used the Weka package [14]. Using its ID3 decision tree learning algorithm we get the following classifier:

Cluster1 ID3:	Cluster2 ID3 :
BL3_bg_color != '!missing' : **yes**	BL5_text_isCity = 'yes' : **yes**

4 Experimental results and Conclusions

For initial experiments with our system have been chosen five example sites, which were already used before for evaluation of other wrapper induction algorithms. The Table 2 shows the small number of positive and negative example instances the human wrapper designer had to enter before the correct filters for a pattern were induced. Comparing to the previous methods for example in [1], construction of wrappers using this approach significantly reduces the effort required by the human designer. Thus, it accelerates the wrapper creation and enables less experienced users to create the wrapper.

Experiments with our current implementation show that further improvements are possible. Therefore, we investigate possibilities of

- better handling of tree regions - that is, instances formed from a sequence of neighbourhood tree elements.
- additional consideration of contextual "before/after" information, to capture instances such as a "table" with an "image" somewhere before.

References

1. R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web information extraction with Lixto. In *The VLDB Journal*, pages 119–128, 2001.

Source	URL	Pattern	Examples
Amazon Camera List	http://www.amazon.com/ exec/obidos/tg/browse/...	Camera	1+2
Google Search	http://www.google.at/search?...	SearchResult	2+0
Yahoo Email Search	http://email.people.yahoo.com/ py/psEmailSearch.py?...	PeopleEntry	1+0
IMDb Title Details	http://imdb.com/title/...	Actor	1+0
IMDb Title Details	http://imdb.com/title/...	Director	1+3
Excite Weather	http://my.excite.com/weather/ obs.jsp?...	Forecast	2+1

Table 2. Evaluation of the pattern induction, number of necessary positive and negative examples

2. J. Carme, A. Lemay, and J. Niehren. Learning node selecting tree transducer from completely annotated examples. In *ICGI*, pages 91–102, 2004.
3. M. Ceresna and G. Gottlob. Query based learning of XPath fragments. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, 2005.
4. R. Gilleron, P. Marty, M. Tommasi, and F. Torre. Statistical classification for wrapper induction. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, 2005.
5. B. Goetz. Easy screen-scraping with XQuery, 2005. Published on http://www-128.ibm.com/developerworks/java/library/j-jtp03225.html.
6. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *Proc. of the 28th International Conference on Very Large Data Bases (VLDB 2002)*, 2002.
7. J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting semistructured information from the Web. In *Proceedings of the Workshop on Management for Semistructured Data*, 1997.
8. N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Intl. Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.
9. A. McCallum and D. Jensen. A note on the unification of information extraction and data mining using conditional-probability, relational models. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, 2005.
10. M. Michelson and C. A. Knoblock. Semantic annotation of unstructured and ungrammatical text. In *Proceedings of the Dagstuhl Seminar on Machine Learning for the Semantic Web*, 2005.
11. W. Recommendation. Document Object Model (DOM) Level2 HTML specification, 2003. Published on http://www.w3.org/TR/DOM-Level-2-HTML.
12. K.-C. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.
13. K. Technologies. Kapowtech Robosuite software, 2000-2005. Homepage on http://www.kapowtech.com.
14. I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 2000.