Modelling Patterns for Deep Web Wrapper Generation

Thomas Kabisch and Susanne Busse

University of Technology Berlin
{tkabisch and sbusse}@cs.tu-berlin.de

Abstract. Interfaces of web information systems are highly heterogeneous. Additionally to schema heterogeneity they differ at the presentation layer. Web interface wrappers need to understand these interfaces in order to enable interoperation among web information systems.
In contrast to the general scenario it has been observed that inside of application domains (e.g. air travel) hetergeneity is limited. More in detail web interfaces share a limited common vocabulary and use a small set of layout variants. Thus we propose the existence of web interface patterns which are characterized by these two aspects: the used vocabulary on the one hand and the common layout of pages on the other. These patterns can be derived from a domain model which is structured into an ontological model and a layout model.
The paper introduces metamodels for ontological and layout models and describes a model driven approach to generate patterns from a sample set of web interfaces. We use a clustering algorithm to identify correspondences between model instances. This pattern approach allows for the generation of wrappers of deep web sources of a specific domain.

1 Introduction

Web data sources offer a huge spectrum of information. The hidden web, which refers to all web database sources, today contains more than 400 times of the amount of static web pages. Dynamic web data sources are mostly accessible through two interfaces: a form-based query interface and a result interface. Both offer an interface schema which does not need to be similar to the underlying database schema.
In contrast to classical databases, web source interfaces are designed for human usage only. Thus each interface schema is hidden behind a representation layer which tends to be unstructured and changes frequently. In order to make these sources available for higher order applications or to integrate them into federated information systems, a wrapping of these sources needs to be applied. Web data sources offer their content only by a restricted query interface which is frequently subject to change. In order to achieve interoperability with such data sources wrappers need to be constructed.

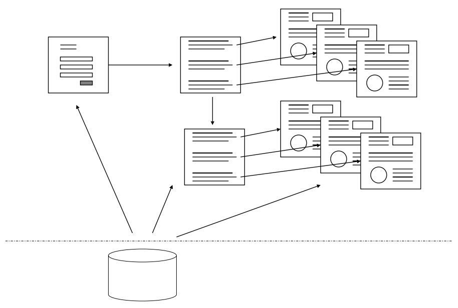
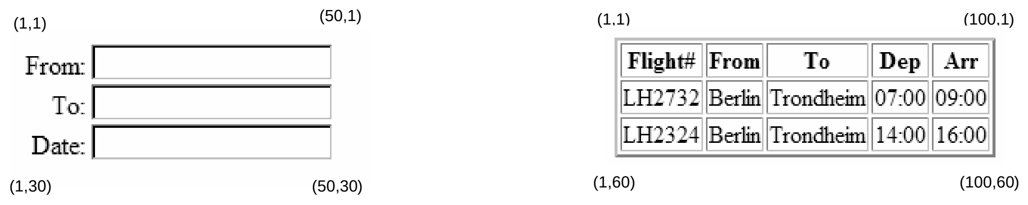


Fig. 1. General model for a deep web source

Deep web sources are only accessible in a multiple-step process via a query interface and different result representation layers (cf. Figure 1). Results are distributed over many pages. Query interfaces as well as result presentation may be spread over multiple pages, moreover sometimes a clear distinction between query- and result pages is not possible. In complex web interaction scenarios their exist pages which contain query forms as well as they present results.
The understanding of heterogeneity of web interfaces is a challenging problem and nearly impossible to generalize domain independently.

In contrast to the general scenario inside of application domains deep web sources the heterogeneity can be limited. An investigation of 100 deep web interfaces in the domain of air travel showed that about 80% of all source interfaces can be classified into four general patterns.



Flight#	From	To	Dep	Arr
LH2732	Berlin	Trondheim	07:00	09:00
LH2324	Berlin	Trondheim	14:00	16:00

Fig. 2. Examples for query- and result page fragments of airline web interfaces.

Consider the fragments of airline web interfaces given in Figure 2. Intuitively these fragments represent common patterns for web interfaces in this domain. They are typical in two aspects: first the used layout and second in terms of the vocabulary used to describe elements. The three fields `From, To, Date` in the given query interface fragment are ordered vertically and layouted closely connected. This layout usually is significant for semantically grouped elements. Similarly the presentation of the results in a table with five columns is typical for the domain. On the other hand, the terms used on both, query interface as well as result interface, will reappear in many other source interfaces of the domain.
This paper proposes an approach to model patterns describing web interfaces of a domain such that the wrapper generation problem can be reduced to a pattern matching problem. A method how to learn these patterns automatically is also presented.
The remaining of the paper is structured as follows. Section 2 introduces domain metamodels which help to describe ontological and layout aspects of source interfaces in a domain. Section 3 describes the model driven learning process from a sample set of interfaces. Section 4 shows a clustering-based approach to derive patterns. Finally in section 5 and 6 we discuss related work and give an conclusion and outlook.

2 Metamodels for Web Interfaces

This section introduces a model-based approach to describe interfaces of web information systems in a given application domain. The model correspondences to the observation that information on a web interface is carried by two paradigms: firstly by the used vocabulary and secondly by a specific layout. Human interactors use both aspects to understand the semantic of the interface. First a user interprets terms in their context to get knowledge about objects of interest, second the arrangement of layout elements helps the user to understand the structure of information. As an example again consider a typical flight booking interface as depicted in Figure 2. There are labels to understand the semantic of adjacent fields (e.g. `Date`). Other pages miss labels, often the arrangement of elements is sufficient for human understanding of the semantic of a field (e.g. the query interface for a date field may be separated into three subfields (year, month, day) with no detail labels).
Our domain model reflects both aspects of web interface design. It is separated into two submodels describing each one of these aspects.
In this section we define a *Layout Metamodel* which generalizes possible layout variants occurring at web information systems of an application domain and an *Ontological Metamodel* describing ontological structures by terms and semantic relationships between them.

2.1 Ontological Metamodel

The *Ontological Metamodel* as depicted in Figure 3 is the frame to instantiate simple ontologies in the domain of interest. These ontologies are restricted to terms occurring on web interfaces in form of labels, names or instances.
Relation types between terms are limited to the most common ones which are needed to understand web interfaces: `isA, isSynonymTo, isInstance`. The `isA`-relation describes semantic hierarchies

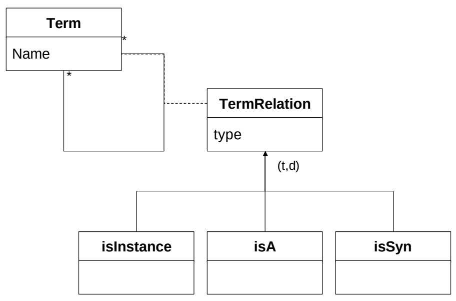


Fig. 3. Ontological Metamodel

among terms respectively terms and instances (if exact distinction to `isInstance` can not be performed). The `isInstance` is more precise to assign instance terms to concept terms. Instance terms are obtained from field domain information (e.g. predefined value lists). Finally the `isSynonym` relationship is used to group semantically similar terms. The cardinality of the relation in the model is not restricted, thus a term can be in relation with many other by multiple types of term relations.

2.2 Layout Metamodel

In contrast to the Ontological Metamodel the Layout Metamodel is based on structures which group information carried by a web interface. The Layout Metamodel is based on logical groups which strongly correspond to specific structures in terms of visualization and presentation.
The Layout Metamodel is built up hierarchically. This reflects the hierarchical structure of web interfaces. Our model is influenced from ideas of [17] and others, who have investigated primarily query interfaces and noticed the hierarchical structure. We generalize the model to all kinds of pages which are relevant for web information systems. More over hierarchies correspond to the HTML data model. We identify three levels of structural layout building blocks in web interfaces: *Page Level*, *Group Level* and *Field Level* the latter two will be denoted as *PageElement Level*. Elements of a lower level compose elements of the next higher level. This is denoted by the AssociationClasss `Composition`, whereas each composition is ordered, thus each subelement has a `Position` attribute.

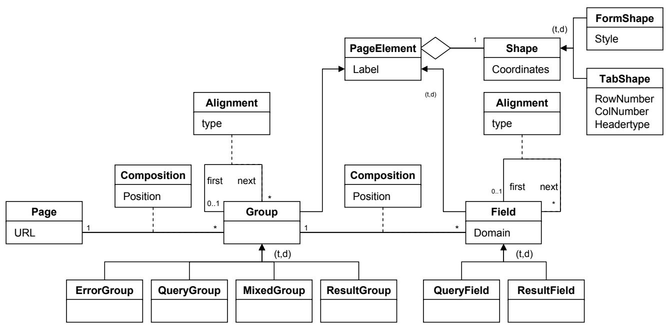


Fig. 4. Layout Metamodel

We now going to describe elements of each level in short.

Page Level The upper most logical unit is a page, whereas we do not distinguish at this level query and result pages because of the existence of intermixed forms of pages which share properties and behavior of query- and result pages (e.g. in the flight domain an additional refine selection page which is dynamically created from first query step results). Thus the distinguishing is shifted to the `PageElement` level.

PageElement Level PageElements are building blocks of pages. Each of them is characterized by two main properties: `Shape` and `Label`. A `Shape` refers the primary layout aspects, thus it has a surrounding bounding box and is specialized into (`TabShape`) for tabular layout of subelements and (`FormShape`) for a form-style presentation.
A `Label` refers to a text element visible at the page which is logical assigned to that `PageElement`. Moreover each `PageElement` is aligned to its neighboring `PageElements` (comp. `Alignment`). Possible alignment types are horizontal or vertical.

Group Level `Groups` represent containers of fields which build up a higher semantic concept. Consider the introducing example in Figure 2. The table there presents a group, all fields which are grouped together build up the semantic concept of a flight description. Commonly grouped elements (e.g. fields) are layouted closely together, thus a group recognition by identifying layout structures is possible.
The distinction between query interface and result representation is accomplished at group level. Thus groups are specialized into the corresponding group types `QueryGroup` and `ResultGroup`. Moreover an error detection is important for page recognition, thus an additional specialization is `ErrorGroup`. Finally sometimes an exact distinction of query or result is not possible (e.g. selecting of prior retrieved flights by the user) thus we define `MixedGroups` to carry out this issue.

Field Level `Fields` are understood as smallest building blocks of pages. Most properties of fields are similar to that of groups. Fields can be aligned to each other, composed to groups and do have a label. We distinguish query interface fields (e.g. HTML input element) and result fields (e.g. table cells). Moreover fields are associated to a bounding box and have an assigned shape. In contrast to groups fields can have information about their domain of possible values.

2.3 Integrated Metamodel

The integration step of both submodels allows a unified view to a domain of deep web sources and is the basis for the later derivation of patterns which are used to generate wrappers for the extraction of information. The integration of both submodels is characterized by two aspects: *Intermodel Correspondences* and *PageElement Clusters*.

Inter-Model Correspondences The Ontological Model and the Layout Model correspond by their used terms. The Layout Model contains terms mainly in the form of labels. All `PageElement`-instances do have a label which is known in the Ontological Model of the domain. Navigating these correspondence relation brings together layout and ontological information. It also allows to address synonymous terms, specialised terms (more precise terms) or instances for labels.
If we consider the query interface example given in Figure 2 we get the labels `From, To, Date`. The correspondence into the Ontological Metamodel will relate these terms to their synonymous and related terms, in this case e.g. `Departure Airport`.

PageElement Cluster Semantically similar `PageElements` will be aggregated to a cluster. This idea is inspired from [17] where such clusters are generated for source query interfaces at field level. Consider the examples from Figure 2. Suppose the query interface shown there is an initial centroid for a clustering. We obtain three clusters c_{From}, c_{To} and c_{Date}. Fields at other query interfaces are clustered relatively to these three initial clusters. In a next step groups will be clustered aggregating information of field clusters. As clusters sum up variants of layout and ontological extensions of web interfaces, the derivation of the desired web interface patterns can be performed directly from them.

3 Model Instantiation from Sample Sets of Interfaces

The overall process of pattern learning is structured as follows:

1. Query Interface Analysis
2. Enrichment of the Ontological Model
3. Processing of Result Pages
4. Clustering of PageElements
5. Derivation of Patterns

This section describes the model instantiation by sample interfaces which is covered by the first three points, the next section investigates the clustering and pattern derivation.

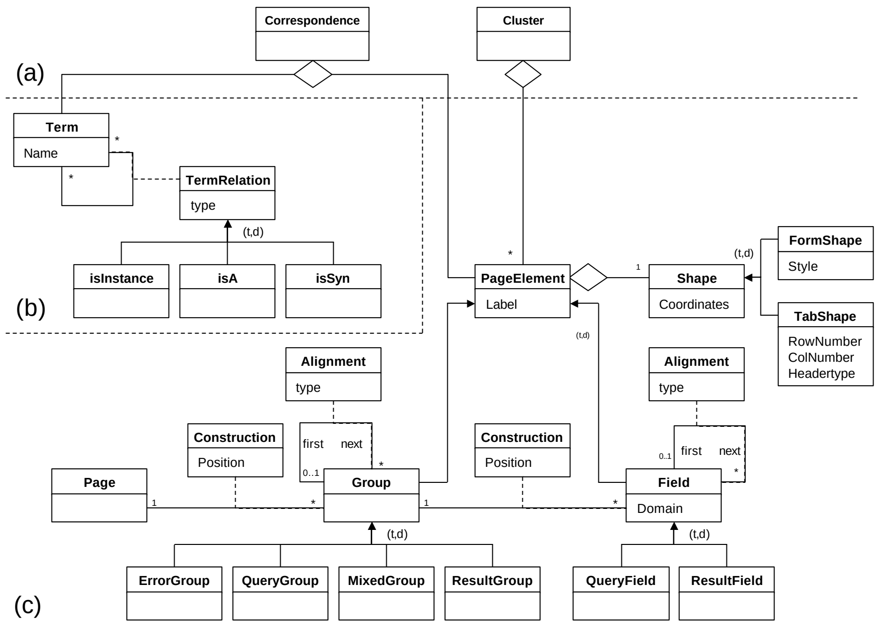


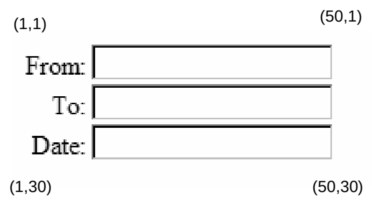
Fig. 5. Integrated Metamodel: (a) inter-Model relations (b) Ontological Metamodel (c) Layout Metamodel

3.1 Query Interface Analysis

Starting points of source analysis are the query interfaces of a sample set of web interfaces. The analysis is performed in two steps, a pre-processing step and the model initialization.

Pre-processing

Tokenization Similarly to [18] we propose a layout driven tokenization of HTML at the beginning. A token contains of a HTML source fragment and is enriched by layout information which is hold in an attached bounding box. The coordinates of bounding boxes are obtained from rendering by a HTML rendering engine.
Figure 6 gives an example of a simplified tokenization of a query interface.



```
<texttoken text = „From:“, pos = „1,1,20,10“>
<inputtoken name = „from“ pos = „20,1,50,10“>

<texttoken text = „To:“, pos = „1,10,20,20“>
<inputtoken name = „to“, pos = „20,10,50,20“>

<texttoken text = „Date“, pos = „1,20,20,30“>
<inputtoken name = „date“, pos = „20,20,50,30“>
...
```

Fig. 6. Simple query interface and result of tokenization

Logical Parsing using visual Grammar This tokenized presentation of HTML-page can be transformed into a logical representation which is compatible to the given Layout Metamodel. The transformation is achieved by using a visual grammar as introduced by [18]. Visual grammars do not only recognize strings but also analyse layout structures. This part of parsing is domain independent because grammar Productions analyse layout structures only.
Having obtained the logical representation of all pages in the sample set the Domain Model can be initialized.

Initialization of Domain Model As described before the Domain Model is divided into two submodels, the Ontological Model and the Layout Model, both get their initial instances from the learning set of source interfaces.

Initialization of Layout Model Having transformed interfaces they can be inserted into layout model. At first corresponding Elements are not grouped together. Thus after initialization the Layout Model is a collection of pages.

Initialization of Ontological Model The Ontological Domain Model consists of concepts/terms which constitute the domain of interest. Initially terms of interest are labels of PageElements, Names of Elements or Instances (e.g. obtained from predefined Selection lists). These are normalized (stemming, stop-word removal, sorting) and inserted. A heuristic divides uncommon and common labels (if a label occurs only once in learning set it is not representative and thus omitted in the Ontological Model). Common terms are saved and uncommon are pruned.

3.2 Enrichment of the Ontological Model

Acquiring Semantic Information from the Web

Application of Language Patterns Specific linguistic patterns in the English language guide to semantic relationships among terms. This idea was first stated out by Hearst [11] for the case of hyponymy. According to Hearst the following patterns describe hyponymy among noun phrases (NP):

1. NP such as $\{NP$, NP,... (and | or) $\}$ NP
2. such NP as $\{NP,\}*$ $\{$(or|and)$\}$ NP
3. NP $\{,NP\}*$ $\{,\}$ (and|or) other NP
4. NP $\{,\}$ (including|especially) $\{$NP,$\}*\{$or|and$\}$ NP

Hyponymy can be interpreted as instance-relationship. Thus the above given patterns can be used to extract instances from the Web or to validate instance candidates. The approach is based on [15]. If new instances are searched, a query containing the hyponym and an extraction pattern is issued. In the case of the attribute `airport code` the query `"airport codes such as"` would be issued to look up potential instances. The result snippets will contain sentences which contain instance candidates near the pattern. In this example the first result contains the phrase `airportcodes, such as LAX` where `LAX` can be extracted as potential instance.

The second option to use Google and linguistic patterns is the validation of instance candidates. In this case a whole phrase containing a linguistic pattern is issued (Example query: `"airport codes such as LAX"`) The approach applies Pointwise Mutual Information (PMI) which is a measure for co-occurrences of two terms in information theory. It is formally defined

$$PMI(x,y) = \log_2 \frac{Pr(x,y)}{Pr(x) * Pr(y)} \tag{1}$$

In our context the probabilities will be approximated by the number of results returned by Google. Thus the above equation is reformulated:

$$PMI(x,y) = \frac{Num(x,y)}{Num(x) * Num(y)} \tag{2}$$

where x denotes the concept term and x the potential instance.

Wikipedia The online encyclopedia Wikipedia provides a huge corpus of texts defining concept terms. In the case Wikipedia contains an article that describes the concept term, there is a high probability that instances are contained in the article. The structure of Wikipedia can be exploited to find that instances. Many articles contain lists encounting related instances explicitly. Consider again the example for `airport code`. The related Wikipedia article contains a list of all known airports and their codes.

This issue has be subject to prior publication [12]

The Process of Semantic Enrichment In the process of semantic enrichment the above summarized technologies are exploited in different ways. We use Web knowledge to construct the Ontological Model as described in the following subsection.

Mining of Relationships First relationships between terms already in the initial Ontological Model will be mined. Here the language pattern based approach is used in combination with Google and Wikipedia (comp. [2], [12]). Queries containing the above described linguistic patterns are issued against Google in order to validate instance relationships, the PMI measure as described above is used to validate the assumed semantic relations.

Finding additional Synonyms Synonyms are essential to enable powerful patterns. Thus the initial model is enriched by synonyms of initial terms from web. Synonyms are acquired from WordNet.

Finding Instances The most important process is the enrichment of the model with new instances. Here we combine both strategies described above. If Wikipedia articles to the retrieved concept exist, first instances from associated lists are extracted. If no sufficient list of instances can be extracted from Wikipedia, the linguistic pattern approach against Google is facilitated.

3.3 Result Processing

Having enriched the Ontological Model in the way described the analysis of result pages can be executed.

Generating Queries Prior to analysing the result pages need to be obtained. Thus queries against existing query interfaces need to be performed. If instances in the Ontological Metamodel or domain information at query interface level allow an automatic query generation the process is done automatically, otherwise a user is requested to provide useful query parameters.

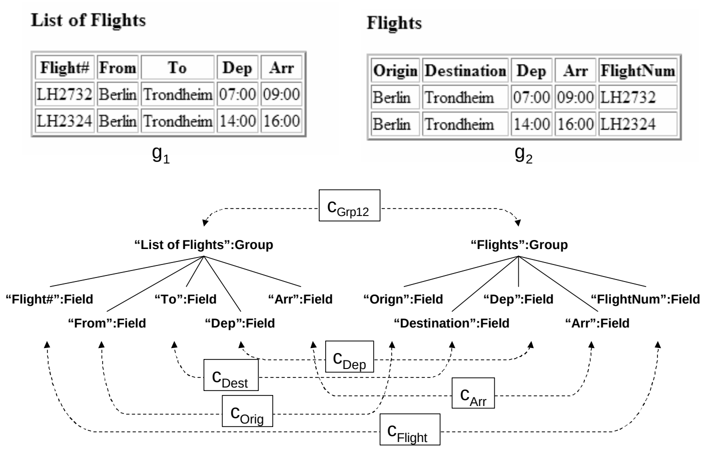
Page Analysis The result analysis is different to the analysis of query interfaces because the structure of result pages is of a higher degree of heterogeneity. Thus parsing of result pages is more difficult than of query interface pages. Here a comparison-based approach such as [6] will support the tokenization and visual grammar parsing process.
Having parsed result interfaces the model instance can be updated by layout representation and new terms found while analysing result pages.
The model now is ready for the pattern derivation process, which is based on Clustering of semantically close PageElements.

4 Pattern Generation

So far, we have instantiated the layout and ontological model with a sample set but do not have analyzed relationships and similarities between these instances. Now, patterns are identified that show similarities between different web sources (i.e. instances). The analysis consists of two steps: firstly, model instances are clustered to identify similar elements, secondly, patterns are derived from these clusters.

4.1 Clustering

After importing all sources of the learning set, the clustering can be accomplished. The clustering algorithm identifies clusters on field level, on group level and on page level using a bottom-up approach.


List of Flights

Flight#	From	To	Dep	Arr
LH2732	Berlin	Trondheim	07:00	09:00
LH2324	Berlin	Trondheim	14:00	16:00

Flights

Origin	Destination	Dep	Arr	FlightNum
Berlin	Trondheim	07:00	09:00	LH2732
Berlin	Trondheim	14:00	16:00	LH2324

Fig. 7. Clustering between two Interfaces

Figure 7 shows an example of identifying a cluster on group level. Using a bottom-up approach field level clusters are identified at first. In a second step, an aggregation to higher levels (group and page-level) is processed.

Field-Level Clusters Field-level clustering is based on ideas in [17] and [7]: two fields belong to the same cluster if they share similar names, labels or domains. Similarity is defined on synonyms and on linguistic methods.

Group-Level Clusters Two groups belong to the same cluster if they share all field clusters or at least if the set of fields is a real subset of the other.

Page-Level Clustering Two pages belong to the same cluster if they share all group clusters or at least if the set of group clusters of the one page is a subset of the set of the other page.

Group- and Page-Level Clustering Consider the example in Figure 7. Supposing all fields are clustered correctly as shown in the figure we can build up the group level clusters. Initially, the following clusters exist at field level:
$clusters = \{c_{Flight} = \{"Flight\#", "FlightNum"\}, c_{Orig} = \{"From", "Origin"\},$
$c_{Dest} = \{"To", "Destination"\}, c_{Dep} = \{"Dep"\}, c_{Arr} = \{"Arr"'\}\}$
We choose one page and try to build up higher order clusters based on groups existing on that page. Suppose we start with g_1 = (Field("#Flight"),Field("From"), Field("To"), Field("Dep"), Field("Arr")). In this case our initial field clusters are all that which appear in g_1.
Now, we check other groups whether their fields belong to one of the clusters which build up the initial group and other way round. If there exist no fields that belong to clusters of fields of other groups, a group-level cluster is found. In the given example we would check whether the fields of g_2 are in a cluster of g_1 and whether all clusters in g_1 have corresponding elements in g_2.
The algorithm in Figure 8 gives a detailed description on how to identify group-level clusters. It consists of the following steps:

1. Take the groups of one page to initialize group-level clusters.
2. For each group of other pages check if it could be assigned to one of the initial group-level clusters using the condition on the groups' fields.
3. For the remaining groups that could not assigned yet, repeat step one and two to find clusters in the remaining set.

```
$groupClusters := \emptyset$
$restGroups := \{g| \in Group\}$
// if there are remaining groups (step 3)
while $restGroups \neq \emptyset$
  // initialise group-level clusters:
  // all (remaining) groups of one selected page build group clusters (step 1)
  choose $p_i$ from $\{p_i | p_i \in Page \wedge \exists g \in restGroups : g$ part of $(p_i)\}$
  $curCentroids := \{cg | cg \in restGroups \wedge cg$ part of $p_i\}$
  for each $cg \in curCentroids$
      $groupClusters := groupClusters \cup \{\{cg\}\}$
  $restGroups := restGroups - curCentroids$
  // for each other group check whether it can be associated to a group cluster (step 2)
  for each $rg \in restGroups$
    for each $cg \in curCentroids$
      // $curGroupCl$ is the set of all groups part of the cluster of $cg$
      $curGroupCl := \{g \in gCl$ — $gCl \in groupClusters \wedge cg \in gCl\}$
      // it can be associated if no field belongs to another cluster
      if    $((\forall f_i \in rg, cl_{fi} \in \{cl_{fi} | cl_{fi}$ part of cluster$(f_i) \wedge cl_{fi}$ part of $p_i\} : cl_{fi}$ part of $cg)$
         and $(\forall f_j \in cg, cl_{fj} \in \{cl_{fj} | cl_{fj}$ part of cluster$(f_j) \wedge cl_{fj}$ part of $p_i\} : cl_{fj}$ part of $rg))$
      then $curGroupCl := curGroupCl \cup \{rg\}$
           $restGroups := restGroups - \{rg\}$
return $groupClusters$
```

Fig. 8. Algorithm for group level clustering

4.2 Patterns Extraction

We can now derive a grammar representing the identified patterns. Again, we construct the grammar bottom-up. Considering our example in Figure 7 we derive the following productions representing semantical patterns.

```
//Field Level Clusters
C_Flight ::= Field("#Flight") | Field("FlightNum")
C_Orig   ::= Field("From") | Field("Origin")
C_Dest   ::= Field("To") | Field("Destination")
C_Dep    ::= Field("Dep")
C_Arr    ::= Field("Arr")

//Group Level Cluster
C_Group12 ::= Group(C_Flight, C_Orig, C_Dest, C_Dep, C_Arr) |
  Group(C_Orig, C_Dest, C_Dep, C_Arr, C_Flight)
```

This grammar extends the initial visual grammar toward domain specific tokens.

4.3 Discussion

The derived grammar gives a detailed description of query interfaces and result pages down to the level of tokens. By analyzing a set of web sources of a domain and by clustering, we identify similarities in layout and vocabulary. Thus, the grammar consists of many variants that can be used to generate wrappers for other, unknown web sources.
But this is also a problem: The grammar also covers uncommon variants of patterns. This leads to an undesirable complexity of the grammar causing an exhausting computation time when analyzing a new page. Therefore, we make experiments on reducing the complexity on the basis of the clustering results.
There are two possibilities for reduction:

1. a brute-force pruning of uncommon production rules: small-sized clusters indicate uncommon patterns that should not be represented in the grammar. Thus, these production rules are removed.
2. enhancing the grammar by prioritization of production rules: rules representing large-sized clusters are checked at first when analyzing a new page, rules determined from small-sized clusters not until a possible rule could not be found before.

This process is accomplished in a top-down manner: A pruning of higher level productions allows for a pruning on lower levels. As already said, the size of the clusters controls the reduction process. Thereby, a fixed number can be used as the threshold of a small cluster or a relative distinction can be done eliminating rules determined from the k smallest clusters.

5 Related Work

Early work in the field of web wrapper generation investigates the extracting structures [4], [3], [1], [5] and others. The main focus of these works is to recognize regular structures at result pages. Other approaches use domain specific extraction ontologies [8].
Another direction of research concentrates on query interfaces with focus of source integration [9], [17]. Whereas [17] develops a measure how to integrate fields by clustering them to semantically close groups. [7] investigates the special aspect of choosing a right label for fields in integrated query interfaces.
[14] brings both interfaces together, reuses information which reappears in both interfaces in order to infer schemes.
An interesting aspect is the supposing of an "hidden" visual grammar, which was first published by [18].
The idea to use language patterns to mine semantic relations was first published by [11], [2] first uses these ideas in the web context, finally [15] analyses query interfaces for integration by language patterns.
Related to these approaches are ontology generation tools, like the Ontobuilder [13] or the DeepMiner-System [16], which takes query interfaces as basis to generate domain ontologies. This part is related to the semantic enrichment in our approach.
All of related work delivers solutions for aspects of the wrapping of web information systems. They all concentrate to specific detail problems. Some of the solutions are part of bigger systems such as the WISE-Integrator [10], but the general focus of the work is different to our approach. To the best of our knowledge no one presents a model based solution and a clustering based pattern generation approach.

6 Conclusion

Even if interfaces of web sources in general are highly heterogeneous, the interfaces and result pages of deep web sources of a specific domain share both common layouts and vocabulary. This paper has shown how we can use this fact to identify patterns at layout and ontological level to make web source wrapping easier. We presented a model-based approach for the recognition of patterns for web wrapper generation. The main contributions are

– an integrated view to ontological and layout-based aspects of web query interfaces and result pages
– a clustering-based approach to learn semantic web patterns from a given set of web sources

The patterns allows for generation of a grammar useful for the wrapping of unknown sources by pattern matching. Moreover, it becomes easier to handle changes of web interfaces due to design or schema adoptions as this evolution can be understood as pattern transformation. Thus, the re-adjustment of a wrapper is no big effort.

Currently, the pattern grammar is directly discovered from clusters. As the clustering algorithm produces clusters for all fields of the web sources, the grammar can contain many alternative rules, even if some of these rules only represent a small amount of web sources. We proposed to improve the grammar complexity by pruning strategies, that identifies rules representing such outliers.

The next steps of work will contain

– more experiments with different pruning strategies for grammar reduction;
– the analysis of grammar transformations to integrate a precedence of production rules;
– the integration of algorithms that allows for inference of more semantic relationships between terms resulting in a better ontological model of the domain;
– the identification of patterns on a higher abstraction level than fields or groups. Particularly, we want to learn patterns on a dialog level.

References

1. Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 337–348, New York, NY, USA, 2003. ACM Press.
2. Philipp Cimiano and Steffen Staab. Learning by googling. *SIGKDD Explorations*, 6(2):24–34, DEC 2004.
3. William W. Cohen. Recognizing structure in web pages using similarity queries. In *AAAI/IAAI*, pages 59–66, 1999.
4. William W. Cohen, Matthew Hurst, and Lee S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *WWW '02: Proceedings of the eleventh international conference on World Wide Web*, pages 232–241. ACM Press, 2002.
5. Valter Crescenzi and Giansalvatore Mecca. Automatic information extraction from large websites. *J. ACM*, 51(5):731–779, 2004.
6. Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109–118, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
7. Eduard C. Dragut, Clement Yu, and Weiyi Meng. Meaningful labeling of integrated query interfaces. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, pages 679–690. VLDB Endowment, 2006.
8. David W. Embley, Douglas M. Campbell, Randy D. Smith, and Stephen W. Liddle. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *CIKM '98: Proceedings of the seventh international conference on Information and knowledge management*, pages 52–59. ACM Press, 1998.
9. Bin He, Kevin Chen-Chuan Chang, and Jiawei Han. Discovering complex matchings across web query interfaces: a correlation mining approach. In *KDD '04: Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 148–157. ACM Press, 2004.
10. Hai He, Weiyi Meng, Clement Yu, and Zonghuan Wu. Automatic integration of web search interfaces with wise-integrator. *The VLDB Journal*, 13(3):256–273, 2004.
11. Marti A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, pages 539–545, Morristown, NJ, USA, 1992. Association for Computational Linguistics.
12. Thomas Kabisch, Ronald Padur, and Dirk Rother. Using web knowledge to improve the wrapping of web sources. In *ICDE Workshops*, page 4, 2006.

13. Haggai Roitman and Avigdor Gal. Ontobuilder: Fully automatic extraction and consolidation of ontologies from web sources using sequence semantics. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany*, pages 573–576, 2006.
14. Jiying Wang, Ji-Rong Wen, Fred Lochovsky, and Wei-Ying Ma. Instance-based schema matching for web databases by domain-specific query probing. In *VLDB '04: Proceedings of the 2004 Conference on Very Large Databases*, 2004.
15. Wensheng Wu, AnHai Doan, and Clement T. Yu. Webiq: Learning from the web to match deep-web query interfaces. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 44, 2006.
16. Wensheng Wu, AnHai Doan, Clement T. Yu, and Weiyi Meng. Bootstrapping domain ontology for semantic web services from source web sites. In *Technologies for E-Services, 6th International Workshop, TES 2005, Trondheim, Norway, September 2-3, 2005, Revised Selected Papers*, pages 11–22, 2005.
17. Wensheng Wu, Clement Yu, AnHai Doan, and Weiyi Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 95–106. ACM Press, 2004.
18. Zhen Zhang, Bin He, and Kevin Chen-Chuan Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 107–118, New York, NY, USA, 2004. ACM Press.