Little Knowledge Rules The Web: Domain-Centric Result Page Extraction

Tim Furche, Georg Gottlob, Giovanni Grasso,
Giorgio Orsi, Christian Schallhart, and Cheng Wang

Department of Computer Science, University of Oxford
{firstname.lastname}@comlab.ox.ac.uk

Abstract. Web extraction is the task of turning unstructured HTML into structured data. Previous approaches rely exclusively on detecting repeated structures in result pages. These approaches trade intensive user interaction for precision.
In this paper, we introduce the Amber *("Adaptable Model-based Extraction of Result Pages")* system that replaces the human interaction with a domain ontology applicable to all sites of a domain. It models domain knowledge about **(1)** records and attributes of the domain, **(2)** low-level (textual) representations of these concepts, and **(3)** constraints linking representations to records and attributes. Parametrized with these constraints, otherwise domain-independent heuristics exploit the repeated structure of result pages to derive attributes and records. Amber is implemented in logical rules to allow an explicit formulation of the heuristics and easy adaptation to different domains.
We apply Amber to the UK real estate domain where we achieve near perfect accuracy on a representative sample of 50 agency websites.

1 Introduction

While two decades ago electronic information was often unavailable, today, we more frequently face the challenge to find the *relevant* information among the vast amount of published data. If you were looking for an apartment in Oxford, neither Google nor the major real-estate aggregators can provide you with a full picture of the market. Even after manually searching through dozens of real-estate web sites, you never loose the feeling that there still might be a property even better suited to your needs. Fully automating web data extraction for individual domains enables a wide range of applications such as object search, trend analysis, or integration with data from other domains.

As an integral part of automating web data extraction, we have to extract data records from pages with search results. All approaches addressing this problem exploit a fundamental property: Records are (visually and structurally) represented similarly, often based on a template. If we have multiple examples of records, we can use this property to identify repeating structures and thus likely record boundaries. The literature of this field includes semi-automated approaches (that need training for each template, and thus each site) as well

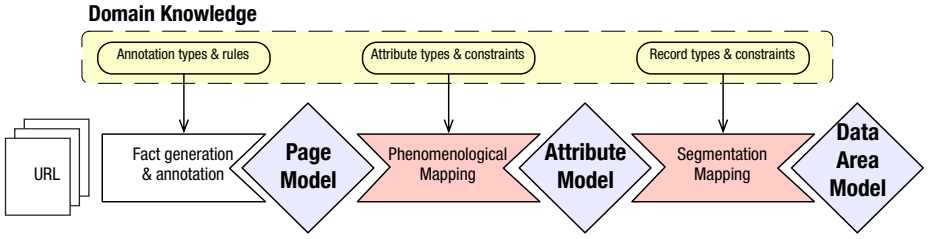

Fig. 1. AMBER Overview

as domain-independent techniques, as surveyed in [2, 6]. Template-dependent, semi-automated approaches are limited to extraction from a small number of web sites. Template and domain-independent methods, on the other hand, are considerably less accurate and perform well mostly in domains with simple entities with few attributes (e.g., news articles with title and body). However, for domains with multiple types of entities with many attributes (such as real estate), their accuracy is too low.

For overcoming the template dependence of the first class of approaches, while maintaining their accuracy, we introduce a third class: template-independent, but domain-aware. Given an ontology of domain entities and a thin layer of knowledge about their appearance on web sites, our approach, named AMBER *(Adaptable Model-based Extraction of Result Pages)*, can identify and extract all records of most sites in the given domain. The system is driven by a set of domain-independent rules that use the structure of the HTML page and domain-dependent annotations (such as UK town names) to automatically derive and verify a model of the attributes and records. For the real estate market in the UK we are able to extract records with accuracy above 99%.

AMBER analyses a web page in three phases, as shown in Figure 1. Each phase produces a (purely logical) model of the result page, increasingly enriched by semantic annotations about type and structure of the identified attributes and records. In the **(1)** phase, the *page model* is obtained from a live browser. It represents all information in the browser's DOM, the visual rendering, and both domain-dependent and domain-independent textual annotations (such as UK locations or currency values). In the **(2)** phase, the page model is used to derive the *attribute model*, which contains potential record attributes and their basic type (location, price, etc.). In the **(3)** phase, the page and attribute model are used to segment the data areas into records exploiting domain specific constraints for records (e.g., mandatory fields) and domain-independent heuristics recognizing repeated structures.

For the extraction of browser facts and the textual annotations we rely on existing APIs and NLP tools (specifically, GATE). All other mappings, phase (2) and (3), and models are purely logical, implemented using datalog.

Contributions. AMBER is designed around a new trade-off between generality, automation, and accuracy. Domain knowledge is used for extracting records on arbitrary sites of the considered domain with almost perfect accuracy:

(1) The integration of domain knowledge allows for *simpler heuristics* which do not rely exclusively on repeated structures for finding records, but construct records from individually identified attributes.
(2) These heuristics are implemented in *datalog rules,* using DLV as reasoning engine, and
(3) achieve *almost perfect precision and recall* (around 99% for record segmentation on a large sample of UK real estate sites).
(4) Moreover, the declarative implementation enables AMBER to be *quickly adaptable to further application domains.* We demonstrate the applicability and accuracy of our approach with our implementation and evaluation on the *UK real estate market.*

1.1 Example

We exemplify our approach with a real life example, taken from the *Zoopla* UK real estate aggregator (`http://www.zoopla.co.uk`). A typical result page from *Zoopla* is structured in two columns, the left with actual results (Figure 2), the right a sidebar with refinement form and various links and advertisements. Results are grouped in two data areas, one containing featured properties, the other the regular results.

Fig. 2. *Zoopla*, result page

For the UK real estate domain, we assume that each record (roughly a property advertisement) contains at least one of the mandatory attributes such as price and location. In Zoopla each record contains the *price* of the property (highlighted with dotted lines in Figure 2). Starting from these attributes, AMBER immediately restricts its analysis to the left column only, rather than also considering areas of the page with no relevant attributes. In our and in most other approaches, the identification of this area is the first phase of the analysis of a result page.

The salient parts of the DOM tree for the *Zoopla* data area are represented in Figure 3. The specification of the mandatory attributes allows us to detect multiple data areas (D_1 to D_3) while most other approaches only consider the largest one (thus skipping featured properties in this case). Mandatory attributes may yield false positives, however, that must be identified in a later phase. In the *Zoopla* case, D_1 contains the average price of the properties ($M_{1,1}$) and the average price paid for a property ($M_{1,2}$). These are considered as potential price attributes since the two prices belong to similar DOM structures. However, the domain knowledge again comes to rescue since this area misses all of the other

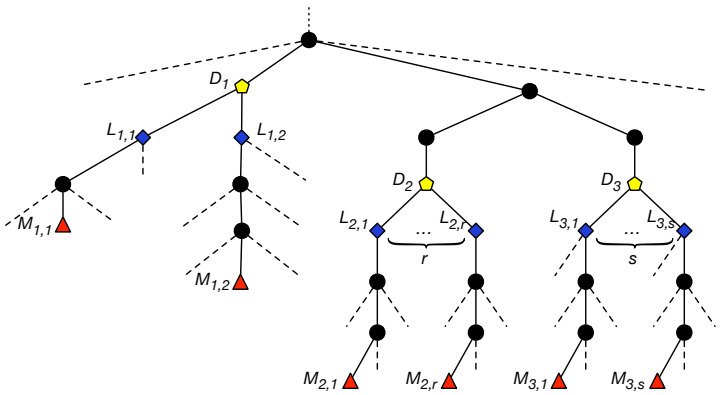

Fig. 3. Results on *Zoopla*

typical attributes for a property such as the location of the property and the number of bedrooms.

The most challenging step of the analysis is the *segmentation phase*. As a first step of the segmentation, we identify the candidate data areas, i.e., those DOM structures that potentially represent sets of data records. AMBER locates all nodes representing mandatory attributes and clusters them by structural similarity. Because records may slightly differ in structure, e.g., due to optional information, we consider a tolerance factor when testing for similarity. Clusters with a low number of nodes are dropped as false-positives. For each cluster we use the least common ancestor node as the root of that candidate data area.

Figure 3 shows the identified clusters for the *Zoopla* case: one rooted at D_2 for featured properties, one at D_3 for standard properties. The clustering procedure also identifies the area rooted at D_1 containing "spurious" prices. These clusters contain nodes at similar depth, but the pairwise distance between nodes in different clusters (e.g., $M_{2,i}$ w.r.t. $M_{3,j}$) differs significantly. The idea is that each data area contains repeated similar substructures that differ significantly from other substructures of other data areas.

Data areas must be then *segmented* into data records by identifying the elements that act as *separators* among them, i.e., DOM nodes with no content. To that end, for every data area, we start by determining the candidate leading node of each records, i.e., the beginning of a record. For pages of medium complexity like those on *Zoopla*, recognizing leading nodes is already sufficient to obtain the correct record segmentation (see nodes $L_{i,j}$ in Figure 3). However, often web pages are more complex in structure. In particular, it might be the case that sibling nodes between those selected as leading are not empty, but rather part of the records. Finding the record boundaries and grouping nodes of the same record requires more sophisticated heuristics, see Section 3.

The final phase is the alignment of attributes with the reference structure of the records provided by the background knowledge. In the *Zoopla* case, the nodes of D_1 miss the required structure since they do not contain attributes such as the number of bedrooms and the location of the property.

1.2 Related Work

For an overview on web data extraction, covering many of the older tools discussed here, see [2, 6].

There exists a large body of *supervised approaches* to web data extraction, e.g. WIEN [5], SOFTMEALY [3], or LIXTO [1]. These tools require the user to annotate example pages from each target site by marking the attributes to be extracted. In contrast to these approaches, we obtain the necessary annotations automatically from our domain knowledge.

Among *unsupervised* tools, we classify the existing methods according to their domain-dependence: **(1)** *Domain-independent* approaches, such as DEPTA [13], VIPER [9], VIDE [7], or FIVATECH [4], rely on repeated structures in the HTML encoding or on the visual rendering of the analyzed web pages. In contrast to our own approach, these tools align record attributes based on their syntactic position within the discovered records, and derive a labeling—if any—from this alignment information [12]. While we search for repeated structure as well, we *first* label potential record attributes, based on some domain knowledge, and *second* search for a repeated structure to explain the potential attribute occurrences in terms of records. This allows us to extract records with higher precision, yet using less complex and easier to adapt heuristics. **(2)** Less frequent, but more recent, *domain-specific* approaches exploit specific properties to detect records on result pages. For example, the machine learning approach in [11] extracts story titles and bodies from news pages, using only a single site for training. However, the features involved for recognizing news titles and bodies are inherently domain-dependent, and the approach does not deal with more fine-grained story properties, such as author names or publication dates. Hence, most ideas in [11] cannot be generalised to other domains. In contrast, AMBER extracts detailed properties from result pages and is easily adaptable to different domains. **(3)** As AMBER, some tools are *domain-aware,* i.e., they are parametrized with a specific application domain but maintain a domain-independent framework. During its initial learning phase, ODE [10] constructs a domain ontology automatically while analyzing a number of sites with domain-independent techniques. The learned ontology is exploited during data area identification and attribute labeling. However, ODE *ignores* its ontology during record segmentation—in contrast to our own approach, which is guided by semantic annotations during the entire result page analysis.

Closest in spirit to AMBER is the approach in [8]. However, it is primarily a proof of concept with very low accuracy (40%-80% according to their own experiments). Furthermore, their approach used for record segmentation, conditional random fields, is fairly involved and far harder to adapt to differing observations than logical rules used in AMBER.

All existing unsupervised web extraction tools are implemented imperatively. AMBER is the first fully automated data extraction tool where the entire analysis is realised with logical rules.

2 Data Model

We divide our data model into three individual submodels, namely page, attribute, and data area model, as shown in Figure 1. We process any given web page in three phases, each producing one of these models.

2.1 Page Model

A rule-based web page analysis requires a logical data model for representing web pages as rendered by a browser engine: Each node has an arbitrary number of child nodes and a field text for the textual content of the subtree rooted at this node. Element nodes have additionally an arbitrary number of attribute nodes and a tag, while attributes have only a name and a value. Moreover, each element has an arbitrary number of CSS attributes from the live DOM, which have—similar to attributes—a name and a value. Different relations between nodes can be queried through relations named after XPath's axes, such as ancestor, descendant, and following.

The textual content of a web site is annotated with domain-independent and domain-specific annotations. These annotations are of several types, e.g., textual or visual, and are produced by imperative analysis components. Some of the annotations are reusable across various domains, e.g., email addresses or city names, while others are specific to a certain domain, such as the abbreviation "STC" in the UK real estate domain, meaning "subject to contract". Currently, we compute these annotations with GATE, using gazetteers and regular expression as specifications but any other annotation tool can be adopted for such a task. Like the basic DOM information, we wrap the output of GATE into logical facts.

Annotations are attached to nodes from the browser page model (in an $n : m$-relationship): a node can contain several annotations, e.g., if its textual content contains a price and a postcode. On the other hand, annotations can spawn several nodes, e.g. with inline content such as link anchors or emphasized texts.

Thus, the only domain dependent parts of the page model are the categories (and features) of annotations specific to that domain, e.g., UK locations.

Predicates. We represent nodes and their position in the DOM tree, using a start-end encoding, yielding the fact `html_node(nodeID,start,end,parentStart)` to represent the node `nodeID` whose start and end tags are the `start`-th and `end`-th tags in the document, respectively, and whose parent node is `parentID`. The textual content of these nodes is given by the relation `content(nodeID,clobID,start,end)` where the textual contents of the node `nodeID` is stored in the character large object `clobID`, starting and ending at character `start` and `end`. The further attributes of more specific node types are added with corresponding facts, not shown here. For example, the following two facts describe a link, its parent paragraph, and their textual contents:

```
html_node(e_504_p,504,509,503).
content(e_504_p,elclob_d1,42045,42579).
```

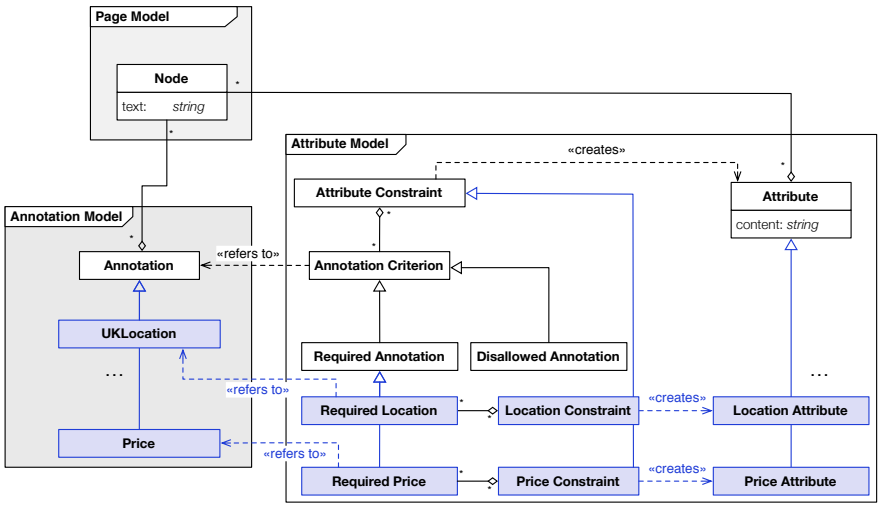

Fig. 4. Attribute Model

```
html_node(e_505_a,505,508,504).
content(e_505_a,elclob_d1,42144,42178).
```

The relations between nodes such as child and descendant, are straightforwardly computed by rules exploiting the start-end encoding. For example, the following rule computes the descendant relation:

```
descendant(N1,N2) ⇐ html_node(N1,Start1,End1,_) ∧ html_node(N2,Start2,End2,_)
                    ∧ Start1 < Start2 ∧ End2 < End1.
```

For representing annotations, we use the following predicates:

- `annotation(annotationID,typeID)`: the annotation `annotationID` is of type `typeID`.
- `referredNode(annotationID,nodeID)`: `annotationID` occurs in the text of node `nodeID`.
- `annotationFeature(annotationID,featureType,featureValue)` associates further features and their value with the annotation `annotationID`.

For example, a text fragment `GBP 70,000` is annotated as follows:

```
annotation(ann1,price).
referredNode(ann1,t_1).
annotationFeature(ann1,value,70000).
annotationFeature(ann1,currency,GBP).
```

The first fact defines the annotation with id `ann1` of type `price`, and the second fact relates the annotation to the text node `t_1`. The final two facts state the price value and currency, i.e., 70.000 British pound sterling (GBP).

2.2 Attribute Model

Based on the page model, we derive the attribute model which describes potential record attributes, as shown in Figure 4. As annotations, attributes are associated with nodes from the page model. For each annotation type, one or more attribute constraints specify whether instances of that attribute should be created. The

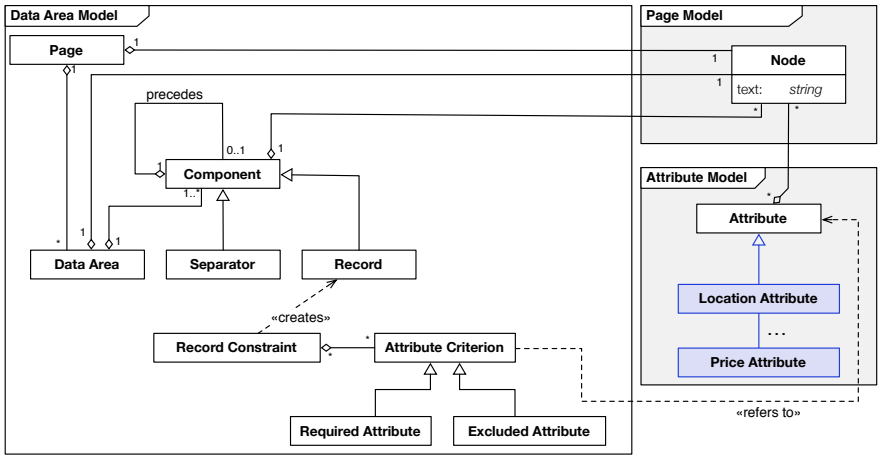

Fig. 5. Data Area Model

constraints are conditioned on the presence or absence of certain annotations: To satisfy all criteria of a constraint, we require that there exists a single node, which **(1)** is referred to by all required annotations, and **(2)** is not referred to by disallowed annotations. Each attribute constraints can be easily seen as a first-order formula and therefore checking it can be reduced to answering a boolean query over the predicates of the attribute model.

The attribute model is represented by

- `attribute(attributeID,typeID)`: the attribute `attributeID` belongs to type `typeID`.
- `referredNode(attributeID,nodeID)`: `nodeID` is a node satisfying the attribute constraint for the attribute type of `attributeID`.
- `attributeConstraint(constraintID,typeID)`: the constraint `constraintID` triggers the creation of an attribute of type `typeID`.
- `requiredAnnotation(constraintID,typeID)`: an annotation of type `typeID` must be present to satisfy the constraint `constraintID`.
- `disallowedAnnotation(constraintID,typeID)`: an annotation of type `typeID` prevents the satisfaction of `constraintID`.

The attribute model provides abstract and domain-independent entities that are then "instantiated" by concrete and domain-independent entities. This separation of concerns enables an easy adaptation of the domain-dependent part of the attribute model to different domains.

2.3 Data Area Model

In the data area model, we describe data areas, records and their attributes. Each page, represented with the predicate `page(pageID)`, is associated with data areas through the predicate `dataArea(areaID,pageID)`. A data area may contain several components such as result records and separators, specified by the predicates `record(recordID,areaID)` and `separator(separatorID,areaID)`, respectively. All these entities are associated with nodes from the page model, i.e., the model contains predicates such as `referredNode(recordID,nodeID)`, if the node `nodeID` is the root of

`recordID. recordAttribute(attributeID,recordID)` associates records with attributes. In analogy to the attribute model, record creation is based on the satisfaction of record constraints which are controlled by criteria such as required and excluded attributes. We omit the predicates as symmetric to the attribute model case.

3 Rules

Logical rules play a key role in AMBER, enabling a declarative specification of the analysis steps that makes the entire process transparent and traceable.

We start from the *attribute* relation of the attribute model introduced in Section 2, that represents the mapping between the attributes of a record in a Web page (e.g., price or a location) and the corresponding DOM node in the page model. The output of the analysis is an instance of the data area model representing the optimal record segmentation of the data areas.

The proposed technique proceeds in three phases. In the **(1)** *identification* phase, we locate the candidate data areas in the web page and we hand them over to the **(2)** *understanding* phase, which produces a record-oriented segmentation of each data area, maximising record similarity. The **(3)** *record disambiguation and alignment* phase ensures that the structure of a record is mapped to a unique DOM structure and that the record's attributes are coherent with the record constraints (e.g., apartments must have at least one room).

3.1 Identification phase

During this phase we identify the data areas on a Web page. The analysis starts by identifying DOM nodes corresponding to *mandatory attributes* (hence *mandatory nodes*); each mandatory node is a clue for the presence of a record and, in the majority of the cases, appears as a leaf of the DOM tree. The notion of mandatory attributes is part of the background domain knowledge and is derived from the required annotations of record constraints (that are also used in the final verification phase below).

The next step is the *clustering* of the mandatory nodes. Each cluster represents a **potential data area** and must enjoy the following three properties:

(1) *Continuity*, i.e., given any two nodes in a cluster, there is no mandatory node between those two nodes in document order that is not in the cluster. Continuity of two mandatory nodes can be easily encoded using the following rule that produces all the non continuous pairs (N_1, N_2) of mandatory nodes in the DOM by establishing the presence of a third node N_3 that falls between N_1 and N_2 in document order.

```
interleaved(N1,N2) ⇐ mandatory(N1) ∧ mandatory(N2) ∧ mandatory(N3)
                   ∧ htmlElement(N1,P1,_,_,_,_) ∧ htmlElement(N2,P2,_,_,_,_)
                   ∧ htmlElement(N3,P3,_,_,_,_) ∧ P1 < P3 ∧ P3 < P2.
```

(2) *Similar depth*, i.e., all the nodes have depth $d \pm \delta$ where d is the distance from the DOM root node and δ is the tolerance (currently 1). This property is

captured by the following rules, where the first two compute the depth of each mandatory node in the DOM, and the third constructs the pairs of nodes at similar depth by testing that the difference in depth is less than δ.

```
nodeDepth(N,1) ⇐ htmlElement(N,_,_,root,_,_).
nodeDepth(N,D) ⇐ +(1,PD,D) ∧ nodeDepth(P,PD) ∧ htmlElement(N,_,_,P,_,_).
simDepth(N1,N2) ⇐ nodeDepth(N1,D1) ∧ nodeDepth(N2,D2) ∧ −(D1,D2,DIFF) ∧ DIFF ≤ δ
                  ∧ N1 ≠ N2 ∧ mandatory(N1) ∧ mandatory(N2).
```

(3) *Similar distance*, i.e., the tree distance between any two nodes in the cluster is $k \pm \epsilon$, where ϵ is the tolerance (currently 1). This property is also easily encoded as rules as shown below. The first rule computes the tree distance between two nodes where the predicate `lca` represents the least commons ancestor of two nodes N_1 and N_2. The second rule computes the incompatible pairs of nodes: if N_1 and N_2 belong to the cluster C, then N_3 and N_4 cannot belong to C, if the pairwise tree distance between N_1 and N_2 and between N_3 and N_4 differs by more than ϵ.

```
treeDistance(N1,N2,D) ⇐ lca(LCA,N1,N2) ∧ nodeDepth(LCA,LD) ∧ nodeDepth(N1,D1)
                        ∧ nodeDepth(N2,D2) ∧ −(D1,LD,DIFF1) ∧ −(D2,LD,DIFF2)
                        ∧ +(DIFF1,DIFF2,D) ∧ mandatory(N1) ∧ mandatory(N2).
bad_quad(N1,N2,N3,N4) ⇐ treeDistance(N1,N2,D12) ∧ treeDistance(N3,N4,D34)
                        ∧ −(D12,D34,DIFF) ∧ DIFF > ϵ ∧ N1 ≠ N2 ∧ N3 ≠ N4
                        ∧ mandatory(N1) ∧ mandatory(N2) ∧ ...
```

The identification of the *candidate data areas* is based on the clusters generated at the previous step. The analysis proceeds differently depending on the number of elements inside each cluster. If only one mandatory node appears on the page, we consider the one cluster containing it as a candidate data area. However, when more than one mandatory node is identified on the page, we consider only clusters containing at least two mandatory nodes. The above strategies have been derived from the empirical analysis of the structure of real web-pages.

3.2 Understanding phase

The goal of the understanding phase is to produce the best record segmentation for each data area. The segmentation assumes that the records are modeled with repeated structures possibly interleaved by separators.

We first locate the root node of each data area, the least common ancestor among the mandatory nodes in the cluster. Then, determine the nodes representing the main part of each record (called *leading nodes*). We say that a DOM node is a *candidate* leading node if it is an ancestor of a mandatory node (including the node itself) and it is a child of the data area's root node. The rules are encoded as follows.

```
leading(ROOT,LN) ⇐dataArea(_,ROOT) ∧ child(ROOT,LN) ∧ mandatory(N) ∧ ancestorOrSelf(LN,N).
```

An iterative pruning procedure is then applied to determine whether some of the candidate leading nodes can be discarded. Indeed, it might be possible to have false positive mandatory attributes that are not part of the repeated

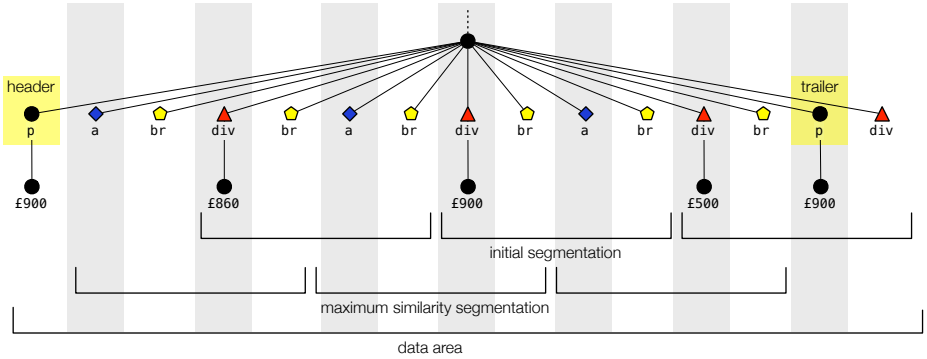

Fig. 6. Complex Record Segmentation

structure within the data area. To this end, we compute the sibling-distance for each pair of adjacent candidate leading nodes as the number of non-leading sibling nodes between them. Next, the following elimination procedure is applied until a fix-point is reached: Consider the first (resp. last) two candidate leading nodes n_1 and n_2 (resp. n_{k-1}, n_k) in the data area. If no other pair of candidate leading nodes in the data area shares the tags of n_1 and n_2 (resp. n_{k-1} and n_k) or their sibling-distance $d(n_1, n_2)$ differs from $d(n_i, n_{i+1})$ for all $i \in [2, k-1]$, then discard n_1 (resp. n_k).

For instance, in the DOM tree of Figure 6—taken from a result-page of *Church Gribben* (`http://www.churchgribben.co.uk/`)—the first candidate leading node is discarded since the sibling-distance between the first two candidate leading nodes is 2, and differs from the distance among all the other pairs of candidates (3). The same principle applies to the last candidate leading node of the data area, that is at distance 1 from its adjacent candidate leading nodes.

The last step of the understanding phase is the identification of the separators between the records. We start by computing the *length* of a record as one plus the number of sibling nodes between two consecutive leading nodes. If this number is greater than one, it means that either **(1)** the content of the record consists of more than one node, **(2)** there exists a separator consisting of one or more nodes, or **(3)** there is a combination of the previous two cases. In the *Church Gribben* case, the data areas do not start with the candidate leading node, as in data area D_2 in the *Zoopla* case. In Figure 6, the data area starts with the nodes of type `a` representing a link to other pages. To address such cases, AMBER's segmentation heuristic proceeds as follows: **(1)** merge all the adjacent separators that appear between two leading nodes into a unique separator. **(2)** if the length of the records is still greater than one, we are in the situation where the records consist of one or more nodes containing data. As a matter of fact, it might be the case that the leading nodes are not the real starting nodes for the records and, therefore, we must "shift" our segments in order to find the optimal segmentation while considering the length of the record as fixed. Each segmentation induces a set of forests where each element of the set is the forest of DOM subtrees of the data area corresponding to a record. The optimal segmentation is the one

with maximum shallow tree similarity between the trees in the different forests. Shallow tree similarity compares only the tags of the first level in the sub-trees.

Segmentation is also encoded as rules as shown below. The first two rules generate all segmentations as sequences of consecutive nodes that are children of the data area. The third and fourth rule prune segmentations that do not contain mandatory nodes and whose length does not match the record length.

```
segment(X,X,1) ⇐dataArea(_,ROOT) ∧ child(ROOT,X).
segment(X,Z,P) ⇐segment(X,Y,P) ∧ +(Pos,P,1) ∧ recordLen(R) ∧ nextSibling(AREA,Y,Z)
                ∧ siblingDistance(AREA,X,Z,DIST) ∧ DIST < R.
goodSegment(X) ⇐segment(X,Y,_) ∧ mandatory(Y) ∧ recordLen(L) ∧ L=#count{V:segment(X,V,_)}.
record(R,M,P)  ⇐goodSegment(R) ∧ segment(R,M,P).
```

Each predicate `record(R,M,P)` encodes each record with its root R, its members M, and the member's position P within R. Among these, we have to identify the one(s) with maximum shallow tree similarity. To this end, we compute the pairs of records such that, in some position the spanned nodes have a different HTML tag (mismatch):

```
mismatch(R1,R2) ⇐ record(R1,X1,P) ∧ record(R2,X2,P) ∧ tag(X1,T1) ∧ tag(X2,T2) ∧ T1 ≠ T2.
similar(R1,R2)  ⇐ record(R1,X1,_) ∧ record(R2,X2,_) ∧ R2 > R1 ∧ ¬ mismatch(R1,R2).
```

Since we now know how many records are produced by each segmentation, we can select those that show the highest number as final record segmentation. The effect of the above heuristics in the *Church Gribben* example is to shift the record segmentation determined by the candidate leading nodes to the left (see Figure 6). This maximizes the shallow tree similarities between the induced forests and, as a result, allows a correct identification of the records.

3.3 Disambiguation and alignment phase

After the understanding phase, it is possible to have inconsistent mappings between the intended record structure and the DOM tree, e.g., we might miss the assignment for a mandatory attribute (apartment with no rooms). This is due to the unavoidable uncertainty introduced by the textual annotations and by the heuristic analysis rules. In these cases some form of data-reconciliation process must be applied. We leverage the constraints of the data area model to filter records that violate these constraints or to reconcile records with minor flaws.

When it is not possible to use background knowledge to disambiguate a multiple assignment, we adopt a scoring mechanism that takes into account the position within the record of the node associated to the attribute's value and the length (in characters) of the value. In particular we privilege nodes at the beginning of records (considering the DOM document order) and nodes with short content. The reason is that meaningful content usually appears in the top left corner of the records and that short content gives higher confidence that the entire content is a value for the attribute.

Site	Areas	Records	Attributes	Price	Location	Site	Areas	Records	Attributes	Price	Location
1	100	100	100	100	100	26	100	100	100	100	100
2	100	100	**99.0**	100	100	27	100	100	**94.9**	100	**73.3**
3	100	100	100	100	100	28	100	100	100	100	100
4	100	100	**97.1**	100	100	29	100	100	**99.3**	100	**96.7**
5	100	100	100	100	100	30	100	100	**99.7**	100	100
6	100	100	**90.9**	100	**92.9**	31	100	100	100	100	100
7	100	100	**97.0**	100	100	32	100	100	**99.3**	100	**96.7**
8	100	**90.9**	**94.7**	**90.9**	**90.9**	33	100	100	100	100	100
9	100	100	**98.5**	100	100	34	100	100	**98.7**	100	**93.3**
10	100	100	100	100	100	35	100	100	100	100	100
11	100	100	100	100	100	36	100	100	100	100	100
12	100	100	100	100	100	37	100	100	100	100	100
13	100	100	100	100	100	38	100	100	100	100	100
14	100	100	100	100	100	39	100	100	**97.9**	100	**87.5**
15	100	100	100	100	100	40	100	100	100	100	100
16	100	100	**99.2**	100	**98.0**	41	100	100	**99.2**	100	**96.5**
17	100	100	**99.2**	100	100	42	100	**96.3**	**93.9**	**96.3**	**80.0**
18	100	100	**98.8**	100	100	43	100	100	100	100	100
19	100	100	**98.2**	100	100	44	100	100	100	100	100
20	100	100	**98.1**	100	100	45	100	100	**99.6**	100	**98.3**
21	100	100	100	100	100	46	100	100	100	100	100
22	100	100	100	100	100	47	100	100	100	100	100
23	100	100	100	100	100	48	100	100	**96.6**	100	**76.0**
24	100	100	100	100	100	49	100	100	100	100	100
25	100	100	100	100	100	50	100	100	**99.8**	100	100
						Avg.	100	**99.7**	**99.0**	**99.7**	**97.6**

Table 1. F_1-Scores for 148 pages from 50 websites

4 Evaluation

The current prototype of AMBER uses an embedded Mozilla Firefox browser, to access a live DOM and extract the structural, visual, and textual content of a web site as logical facts. We use GATE to annotate the content and wrap its output in logical facts (Section 2.1). All remaining steps, i.e., the identification, disambiguation and alignment, are performed by DLV with datalog rules which are extended with finite domains and non-recursive aggregation.

We evaluate AMBER on 50 UK real-estate web sites, randomly selected from 2810 web sites extracted from the yellow pages. For each site, we submit its main form with a fixed sequence of fillings until we obtain a result page with records. If the same query produces more than one result page, we take the first two pages into our sample. The validation set consists of manually-labeled data areas, records, and attributes on selected result pages.

Table 1 summarizes the results of our evaluation: For each of the 50 sites, we show the F_1-score (harmonic mean of precision and recall) for the extracted data areas, records, and their attributes. For the two major attributes, price and location, we also show the individual F_1-scores. The *averages* for all scores provide a summary of the evaluation at the bottom of the table. The experiment takes on average about 5 seconds per page (Intel Core2 Q9650 3GHz with 8GB RAM), including page rendering of the browser.

The analyzed pages contain 128 data areas, 1477 records, and 7033 attributes. AMBER extracts *all data areas correctly* and achieves *on records perfect accuracy with nearly perfect F_1-score* (99.7% on average). For attributes, AMBER reaches on average 99.4% and 98.6% precision and recall, respectively. AMBER misses almost no price attributes and has near perfect accuracy (99.7%). It is, however, harder to identify locations, whose descriptions are variant. Nevertheless, AMBER still achieves an accuracy of precision 99.2% and recall 96.5% for locations. We also extract postcodes, detail page links, bedroom numbers, and the legal status (sold, for sale, etc.). We do not show the results individually, but consider them in the cumulative numbers for attributes, e.g., for site 4 the legal status is not always identified, causing a drop in accuracy to 97.1%.

These results are comparable or better than those published for other unsupervised systems. For example, VIPER [9], applied to search engine results, reaches F_1-scores between 98.0% and 98.2% for the extracted records. More recently, the system in [11] extracts a single news story from any suitable page, reaching an accuracy of 99.6%, but only considers one title and one body per web page. Compared to manually inserted offerings on a small real estate agency's site, search engine results and news stories are particularly well-structured domains. Also the larger number of attributes in the real-estate domain significantly complicates the task. On more similar domains, e.g., publication records from sites such as Google scholar, DBLP, or ACM, [8] reports much lower accuracy than our approach ($50 - 80\%$).

We strongly believe that the AMBER approach, which already competes and outperforms existing systems, has the potential to reach $99\% - 100\%$ accuracy on a properly configured domain with four further improvements: **(1)** AMBER's ability to recognize mandatory attributes depends on the quality of the gazetteers and annotation rules. Depending on the attribute, our textual annotators already reach near perfect accuracy—but some of them still fail for specific sites, e.g., if they offer properties at some out-of-the-way location not covered by dedicated gazetteers. **(2)** More interestingly, AMBER does not attempt to identify attributes which have been missed by the annotators. But this would be possible, e.g., assuming that the same attribute appears (roughly) at the same position within the record. **(3)** AMBER is deterministic and in some cases has to choose between low-confidence alternatives—premature choices that prevent better results in later stages. We have started to investigate the careful use of probabilistic reasoning to address this issue. **(4)** AMBER relies solely on the HTML structure. We plan to integrate heuristics use visual information, e.g., the bounding boxes of HTML elements, fonts, or colours.

5 Conclusion

AMBER's performance demonstrates the effectiveness of its rule- and knowledge-based approach in unsupervised web data extraction—with all the advantages of a declarative implementation. In future work, we plan to improve AMBER following the ideas **(1-4)** outlined at the end of the previous section. We are currently

devising a methodology for the application of AMBER to different domains. AMBER is quickly adaptable to further application domains. The identification phase and understanding phase are domain independent. Since the structure of result pages in different domains are similar, all we need is to find a mandatory field of the domain, e.g. price. We also plan to use machine learning to identify titles of records, which is a mandatory field for all records in all domains. The only domain-dependent components of AMBER are the gazetteers built by regular expressions and the domain-specific entities of the attribute model. These artefacts can be seen as parameters for AMBER, therefore their modification does not require any change at algorithmic level.

Acknowledgments

The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement no. 246858 (DIADEM), http://diadem-project.info.

References

1. R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web Information Extraction with Lixto. In *VLDB*, 2001.
2. C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A survey of web information extraction systems. *TKDE*, 18(10), 2006.
3. C. Hsu and M. Dung. Generating finite-state transducers for semistructured data extraction from the web. *IS*, 23(8), 1998.
4. M. Kayed and C.-H. Chang. FiVaTech: Page-Level Web Data Extraction from Template Pages. *TKDE*, 22(2), 2010.
5. N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper Induction for Information Extraction. In *VLDB*, 1997.
6. A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2), 2002.
7. W. Liu, X. Meng, and W. Meng. Vision-based Web Data Records Extraction. In *WebDB*, 2006.
8. P. Senellart, A. Mittal, D. Muschick, R. Gilleron, and M. Tommasi. Automatic wrapper induction from hidden-web sources with domain knowledge. In *WIDM*, 2008.
9. K. Simon and G. Lausen. ViPER: Augmenting Automatic Information Extraction with visual Perceptions. In *CIKM*, 2005.
10. W. Su, J. Wang, and F. H. Lochovsky. ODE: Ontology-Assisted Data Extraction. *TODS*, 34(2), 2009.
11. J. Wang, C. Chen, C. Wang, J. Pei, J. Bu, Z. Guan, and W. V. Zhang. Can we learn a template-independent wrapper for news article extraction from a single training site? In *KDD*, 2009.
12. J. Wang and F. H. Lochovsky. Data extraction and label assignment for Web databases. In *WWW*, 2003.
13. Y. Zhai and B. Liu. Structured Data Extraction from the Web Based on Partial Tree Alignment. *TKDE*, 18(12), 2006.