Real Understanding of Real Estate Forms*

Tim Furche, Georg Gottlob, Giovanni Grasso, Xiaonan Guo, Giorgio Orsi, Christian Schallhart

Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD
firstname.lastname@comlab.ox.ac.uk

ABSTRACT

Finding an apartment is a lengthy and tedious process. Once decided, one can never be sure not to have missed an even better offer which would have been just one click away. Form understanding is key to automatically access and process all the relevant—and nowadays readily available—data.

We introduce OPAL (**o**ntology-based web **p**attern **a**nalysis with **l**ogic), a novel, purely logical approach to web form understanding: OPAL labels, structures, and groups form fields according to a domain-specific ontology linked through *phenomenological rules* to a logical representation of a DOM. The phenomenological rules describe how ontological concepts appear on the web; the ontology formalizes and structures common patterns of web pages observed in a domain. A unique feature of OPAL is that all domain-independent assumptions about web forms are represented in rules, whereas domain-specific assumptions are represented in the ontology. This yields a coherent logical framework, robust in face of changing web trends.

We apply OPAL to a significant, randomly selected sample of UK real estate sites, showing that straightforward rules suffice to achieve high precision form understanding.

Categories and Subject Descriptors

H.3.5 [**Information Storage and Retrieval**]: On-line Information Services—*Web-based services*

General Terms

Languages, Experimentation

Keywords

form understanding, data extraction, deep web, web page phenomenology

*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.

WIMS'11 May 25-027, 2001 Sogndal, Norway.

1. INTRODUCTION

On the web today there are as many pages, as there are stars in the Milky Way! Through observation and analysis we have identified patterns of genesis and behavior that allow us to categorize stars and determine their properties, despite the distances involved. Unfortunately, for web pages the same principled analysis is still in its infancy.

In this paper, we focus on the particular problem of *web form understanding*. Web forms are one of the more deeply investigated aspects of the Web. Their understanding is crucial for "deep-web" search engines, for web data-extraction, and for web querying. By "understanding" we primarily mean the identification of forms and form elements, along with their logical organization beyond the asserted HTML structure.

Approaches to form understanding in the context of deep web search [11, 15, 9, 14], web querying [16, 2] and in web extraction [13] have focused on observing commonalities of general web forms exploiting in specifically tailored algorithms and heuristics. Despite reportedly good performance, two issues seriously limit their applicability in practice: *(1)* In all the above approaches, the necessary assumptions are hard-coded into the implemented algorithms and it is not easy (or even possible) to adapt them. Furthermore, in many cases mere parametrization of the heuristics does not suffice for the needed adaptability requirements, especially in an open scenario such as the Web. *(2)* Trying to define general heuristics capable of producing highly precise results in all domains is not an easy task. By generalizing the assumptions made about web forms we are automatically forced to ignore domain-specific patterns that can make a real difference in form understanding for entire classes of web sites.

In this paper, we introduce OPAL, short for **o**ntology-based web **p**attern **a**nalysis with **l**ogic. OPAL uses Prolog rules to explicitly represent assumptions about commonalities of web forms (and other types of web objects). Thus OPAL allows *(1)* the declarative definition of the needed assumptions and heuristics through Prolog rules, *(2)* the specification of multiple sets of rules to be chosen and applied in order to adapt to the situation at hand, and *(3)* the easy integration of background knowledge (e.g., about the domain, the patterns of web forms, the used vocabulary). We have implemented a prototype system for analyzing real-estate forms in the UK, that exploits background knowledge on the domain (e.g., to distinguish forms for renting and buying properties) and adapts to the observed form type by using different assumptions. We show that an encoding of those assumptions as

Prolog rules yields concise, easy to understand and modify specifications. At the same time, it competes with existing approaches using hard-wired heuristics for speed and precision.

OPAL represents information at three different levels of abstraction corresponding to three steps of the proposed form understanding process:

(1) OPAL's relational **browser page model** (Section 2) faithfully represents the HTML DOM as constructed by a browser (including any modifications through scripting). The data model is enriched by a thin layer of *annotators* (Section 2) that provide additional annotations on element nodes and textual information. In this paper, we consider visual annotations derived from the browser engine and linguistic annotations about entities on the web page and their relations.

(2) In a set of **phenomenological rules**, we employ a number of heuristics on such a representation of a web page *(1)* to assign labels to fields, *(2)* to hierarchically segment a form, and *(3)* to categorize form fields based on their appearance on the web page. These heuristics are primarily domain-independent, but parameterized by domain-specific information such as the labels or field values associated with certain form field categories.

(3) We derive a **real-estate form ontology** (Section 4) by carefully analyzing forms on over 50 UK real-estate web sites. This ontology is used to construct a model of a form based on the categories assigned to form fields. This model can be further used, e.g., to fill the form fields for deep web crawling or data extraction or to integrate the form with a global schema of a meta-query engine.

Our preliminary *experimental evaluation* (Section 5) shows that OPAL recognizes real-estate forms and their elements with very high precision on a random sampling of UK real-estate web sites. It also demonstrates that the time for analyzing web pages with OPAL is dominated by the time for rendering the page.

To illustrate how OPAL uses its three layers (browser page model, phenomenological rules and real-estate form ontology), we continue with an extended example on the search form of a local UK real-estate site. We choose this example as it is concise, but allows us to illustrate many of the salient issues in OPAL.

1.1 Running Example: Heritage4Homes

We use `http://www.heritage4homes.co.uk/` as our running example. Figure 1 shows the relevant web form of this site. This form uses a number of checkboxes to select the areas in which to search for properties. This is actually a fairly unusual case, with most forms either using drop-down lists or postcode inputs. It also provides a checkbox to indicate whether to search for retirement properties and drop-down lists for the minimum number of required bedrooms, the minimum and maximum price, and the order in which to return the properties. We process this form in three stages as discussed above.

Extracting the Browser Page Model. We employ HtmlUnit [3] to load the web page containing the form, since HtmlUnit evaluates all included scripts, renders the page approximately, and provides programmatic access to the dynamical DOM. We extract the individual form elements, their hierarchical structure, and their visual appearance. For example, we extract the facts shown below to represent the first check box of the form in Figure 1.

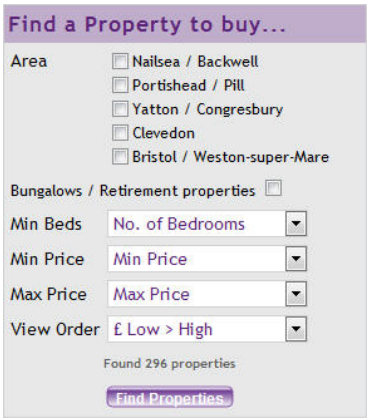

Figure 1: Form on Heritage

```
html-element(e_199_input,199,200,198,input).
html-attr(e_199_input.type,e_199_input,type).
html-attr(e_199_input.name,e_199_input,name).
html-attr(e_199_input.id,e_199_input,id).
html-attr(e_199_input.value,e_199_input,value).
html-attr(e_199_input.onclick,e_199_input,onclick).
css_attr(e_199_input,bottom,"auto").
css_attr(e_199_input,clear,"none").
css_attr(e_199_input,clip,"auto").
css_attr(e_199_input,color,"rgb(0,0,0)").
...
css_attr(e_199_input,top,"auto").
css_attr(e_199_input,visibility,"visible").
css_attr(e_199_input,width,"auto").
css-box(e_199_input,45,19,755,34).
```

Applying the Phenomenological Rules. In this stage, we complete the low-level facts in order to establish a phenomenologically complete description of the form fields in the form. For the form from Figure 1, we find the labels for the individual form fields: We locate the least common ancestor containing only the form field at hand and no other form fields, and assign all its contained text nodes as labels to the form field, resulting in the following facts:

```
hasLabel(e_199_input,"Nailsea / Backwell").
hasLabel(e_207_input,"Portishead / Pill").
hasLabel(e_215_input,"Yatton / Congresbury").
hasLabel(e_223_input,"Clevedon").
hasLabel(e_231_input,"Bristol / Weston-super-Mare").
hasLabel(e_243_input,"Bungalows / Retirement properties").
hasLabel(e_251_select,"Min Beds").
hasLabel(e_279_select,"Min Price").
hasLabel(e_321_select,"Max Price").
hasLabel(e_363_select,"View Order").
```

By recognizing the similarity of the first five elements in terms of their attributes, we group them explicitly into a logical segment which is represented by the `tbody` element `e_196_tbody`, resulting in facts such as:

```
partOf(e_199_input,e_196_tbody).
```

Once this segment has been identified, we look for a label for this segment. As in the case of the individual form elements, we find the label via the least common ancestor:

```
hasLabel(e_196_tbody,"Area").
```

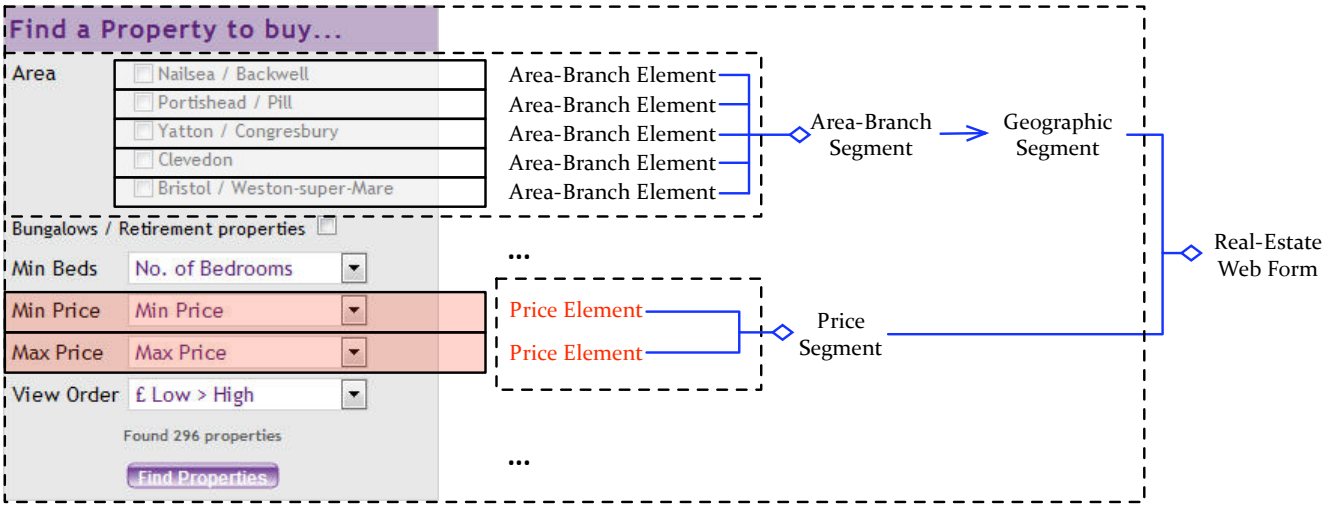

Figure 2: Annotated Example Form (Heritage)

Based on the associated labels, we classify the fields using the label annotations of the corresponding form element concepts from the ontology. The first five fields are identified as `areaBranchElement` with `granularity` set to `area`. Such an element designates a certain area either by its name or responsible branch of the real estate agency. The first checkbox yields the following two facts.

```
areaBranchElement(e_199_input).
granularity(e_199_input,area).
```

As another example, the first select box is classified as `roomElement` which prescribes the required number of rooms of a certain type. In our case, the room's `category` is `bedroom`.

```
roomElement(e_251_select).
category(e_251_select,bedroom).
```

Similarly, the next two fields are classified as `priceElement`. Using the values in the drop down lists, we determine the purpose of these prices as `buy`, indicating that the prices refer to buying prices as opposed to renting prices. Finally, using the labels, we find that the first price refers to the minimum and the second one to the maximum price of a specified range.

```
priceElement(e_279_select).
priceElement(e_321_select).
purpose(e_279_select,buy).
purpose(e_321_select,buy).
priceType(e_279_select,min).
priceType(e_321_select,max).
```

Ontological Processing. Using our ontology, we determine the logical structure of the web form. Figure 2 shows the form together with the annotated form model. In the figure area branch elements are shaded in gray, price elements are shaded in red. The blue lines correspond to the relation in the ontology (Section 4). Segments are also visually illustrated as boxes with dashed border. Where a segment is "virtual" (here: the price segment), i.e., does not correspond directly to an HTML node, it is drawn only around the contained elements. Otherwise it is drawn around the contained elements and their visual representation. In particular, we recognize that the five `areaBranchElement`'s all share a `granularity` of `area`, and group them into an `areaBranchSegment` of the same `granularity`. The `areaBranchSegment` is a type of `geographicSegment`.

```
geographicSegment(e_196_tbody).
areaBranchSegment(e_196_tbody).
granularity(e_196_tbody,area).
areaBranchElement(e_199_input).
areaBranchElement(e_207_input).
areaBranchElement(e_215_input).
areaBranchElement(e_223_input).
areaBranchElement(e_231_input).
```

Similarly, since the two `priceElement`'s have complementary types `min` and `max` and a compatible `purpose` set to `buy`, we group the two elements into a `priceSegment` with the same `purpose`.

```
priceSegment(priceSegment(e_190_tbody)).
purpose(priceSegment(e_190_tbody),buy).
priceElement(e_279_select).
priceElement(e_321_select).
purpose(e_279_select,buy).
purpose(e_321_select,buy).
priceType(e_279_select,min).
priceType(e_321_select,max).
```

In this case, the `priceSegment` cannot be identified as an HTML element but must be annotated to an *invented value* which we introduce to this end (*priceSegment*`(e_190_tbody)`).

The `geographicSegment` and `priceSegment` are sufficient to identify the whole form as a `realEstateWebForm`, as witnessed by the following fact:

```
realEstateWebForm(e_190_tbody).
```

1.2 Related Work

Web form understanding [7] is essential to web automation tasks dealing with the deep web, for surfacing the content behind forms for search index generation, wrapper construction, or interface integration, see for example [9, 13, 15]. While earlier approaches [6, 8, 9, 10, 11] treated web interfaces as flat structures, current approaches [2, 5, 14] model web interfaces hierarchically. Since our approach is hierarchical as well, we relate our own work with these approaches.

Any approach for form understanding must rely on assumptions on human form understanding and the corresponding design principles. Most of these rules describe how logical attributes are mapped onto physical form elements—however, these *rules are not formalized as rules* but only underlie the employed heuristics or grammar. In contrast, our technique has a *phenomenology* at its heart, which explicitly describes this mapping in terms of a logic rule base. By having these rules represented in a formal and machine processable language such as Datalog, we can flexibly combine phenomenological information with domain knowledge to improve the accuracy of our approach.

In [2], the tree structure of a form is derived in two steps: First, separate trees for text and form fields are built. Then these two trees are iteratively merged, yielding a label assignment in the process. This approach is based on nine general rules on web forms which attempt to capture common web design practice. For example, these rules say that query interfaces are organized top-down and left-to-right, or that group and element labels are styled consistently. Also, the elements are grouped hierarchically, based on so-called *inflection points,* which mark the points where a human agent supposedly changes the reading direction during processing the form.

ExQ [14] works in two phases, by *(1)* extracting an unlabeled tree of elements and *(2)* decorating the obtained tree with labels. The unlabeled tree is generated from topological relations, e.g., containment or disjointness, 4-way neighborhood relations, and alignment relations. For assigning the labels, ExQ uses *annotation blocks* which are the bounding boxes around a label and the labeled elements. Guided by a number of assumptions, such as that annotation blocks do not overlap, ExQ processes the element tree in a bottom up manner and assigns labels to all elements in a group simultaneously in each step.

Wise-iExtractor [5] is implemented in two phases: It performs an *attribute extraction* and a separate *attribute analysis* which aims at revealing "hidden" meta-information. In this process, Wise distinguishes *logical attributes* which are represented by *physical elements* on the form. The attribute extraction tokenizes the form to obtain an *interface expressions (IEXP)*, distinguishing text fragments, form elements, and delimiters, such as line breaks. Wise computes the *association weight* between any given element and the labels in the same line and the two preceding lines and assigns labels accordingly. The associative weight is computed with heuristics that exploit ending colons, similarities in the HTML name attributes of elements and labels, and the distance between element and label. During *attribute analysis*, Wise determines the domain values, the default value, and the unit of form elements. It also groups together the elements forming a single logical attribute and analyzes their relationships: For example, they are classified to be in one out of *four relation types,* namely range (e.g. from, to), part (e.g. first and last name), group (e.g. radio buttons), or constraint (e.g. exact match required).

Closest in spirit to our own approach, the method in [16] follows the observation that forms "seem to reveal some 'concerted structure', by sharing common building blocks" and presupposes the existence of a *hidden syntax*, shared by most web forms, which "connects semantics to presentations." This hidden syntax is formalized into a visual language which is described in a *2P grammar* which is based on common patterns and relative preferences between these patterns—providing a declarative description of the phenomena commonly encountered in web forms. However, in contrast to our logical approach, the formalization as a grammar does not lend itself easily to the integration of the phenomenological rules with domain-specific ontological knowledge. It is also unclear how to contextualize grammar rules so as to select which set of rules best applies to a given form.

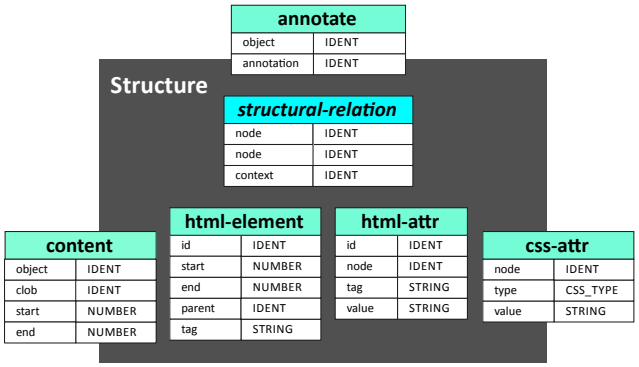

Figure 3: Logical model of HTML DOM structure

2. BROWSER PAGE MODEL

To analyse forms (and web pages in general) with rules, we first need a logical data model for browser pages. This data model faithfully represents the structural, textual, and visual properties of the web page as rendered by a browser engine. We represent the browser's DOM tree (including element, text, comment, and attribute nodes) in structural relations (Figure 3), the associated textual content of the DOM (Figure 4) in a large character object (CLOB) with references from the DOM nodes to start and end positions in the CLOB, and the browser's rendering including all CSS attributes and bounding boxes (Figure 5). Since the encountered web sites do not employ semantic web technologies [12], such as RDF, OWL-DL, or RIF, we did not incorporate them into our data model.

Structural information. The structural information from the DOM (which nodes there are and how they are related and ordered) is represented in the extensional relations `html-element`, `html-text`, `html-comment`, and `html-attr`. We encode the tree structure using the start/end encoding from [1]. We provide derived structural relations for all XPath axes, and for first-child, next-sibling, and next. For instance, `next-sibling(X,Y)` holds if X and Y have the same parent and Y follows immediately after X (`node` is a convenience relation for accessing all nodes in the DOM):

```
next-sibling(X,Y) ⇐ node(X,_,EndX,Par) ∧
2       node(Y,StartY,_,Par) ∧ StartY is EndX + 1.
```

DOM nodes are related to their textual content and CSS attributes by the `content` and `css-attr` relations, and may be annotated with arbitrary properties via `annotate`. Figure 3 shows the schema for the structural information, omitting `html-text` and `html-comment` for space reasons.

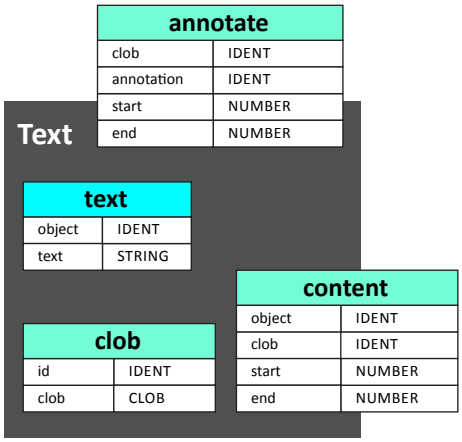

Figure 4: Logical model of HTML textual content

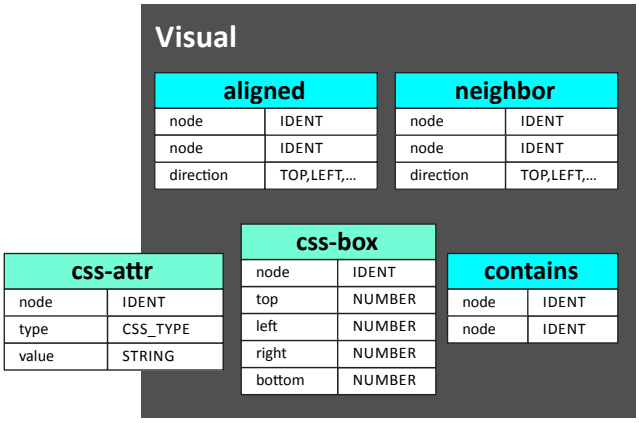

Figure 5: Logical model of HTML rendering

Textual information. DOM nodes are linked (by `content`) to textual information specifying the start and end position of the textual content in the given CLOB. We use start and end positions to allow annotations crossing element borders and to concisely represent the textual content of nested elements, but offer a convenience relation `text` to access the contained text of a node directly (Figure 4).

Visual information. From the visual rendering of the web site we extract all CSS attributes of DOM nodes (accessed through `css-attr`) as well as their bounding boxes (`css-box`). From these information we derive a number of spatial relations such as `contains`, `aligned`, and `neighbor`, see Figure 5. In particular alignment and neighborhood are essential for form segmentation.

Example. To illustrate the logical data model of browser pages in OPAL, consider the fragment of a DOM tree shown in Figure 6. It shows a part of a table with a `tr` containing two `td`'s, the first a label ("Price:") for the form `input` field contained in the second. The figure shows the `html-element`, `html-attr`, and `html-text` facts for representing these DOM nodes (and the `content` fact for associating the text node with its textual content). The visual information extracted for these elements are all their CSS attributes (including

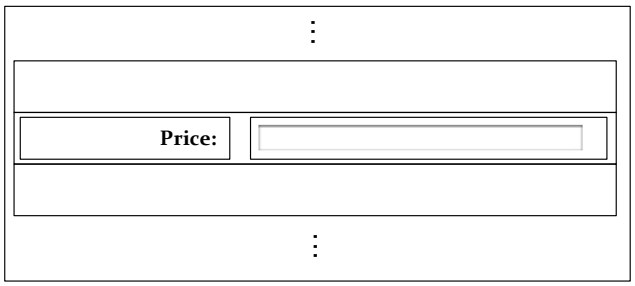

Figure 7: Example (rendering)

margin, text and background color, text font, etc.) and the bounding boxes shown in Figure 7.

3. Phenomenological Rules

Based on the browser page model, the phenomenological rules label, segment, and classify the form fields contained in the web page. We call these rules "phenomenological" as they connect the abstract concepts of the ontology (such as price input element) with observable phenomena on the web sites (such as an input field with a certain label). To that end, we

1. group form fields (and thus their labels) into **segment**,
2. pair form fields with their labels into **form elements** (form fields with their labels),
3. **classify** form elements according to the domain ontology.

3.1 Segmentation

In the UK real estate domain, a single rule proved sufficient to group related segments recursively into compound segments. The predicate `segment(grp(Es))` holds, if `grp(Es)` is a segment, where `Es` is a list of (other) segments or form fields which satisfy the following two conditions: *(1)* They occur in a sequence in the web form and are similar to each other, where similarity means that either their type and style attributes or their class and name attributes coincide. *(2)* Their least common ancestor contains no other form segments. The least common ancestor is the node which contains all these segments, but has no children that also contains all of them. We translate these conditions directly into the following rule:

```
segment(grp(Es)) ⇐ similarSegmentSequence(Es) ∧
        leastCommonAncestor(Es,A) ∧
         ¬ hasAdditionalSegment(A,Es).
```

Therein, the first body atom `similarSegmentSequence(Es)` identifies the segment sequences which match condition (1). For condition (2), we determine `A` as the least common ancestor of the segments in `Es`, using the `leastCommonAncestor(Es,A)` predicate, as defined in the following rule. It asks for a common ancestor having no child being a common ancestor of the segments in `Es`.

```
leastCommonAncestor(Es,A) ⇐ commonAncestor(Es,A) ∧
         ¬ (child(A,C) ∧ commonAncestor(Es,C)).
```

We define a `partOf` relation between the identified segments with the following rule:

```
partOf(E,grp(Es)) ⇐ segment(grp(Es)) ∧
        member(E,Es).
```

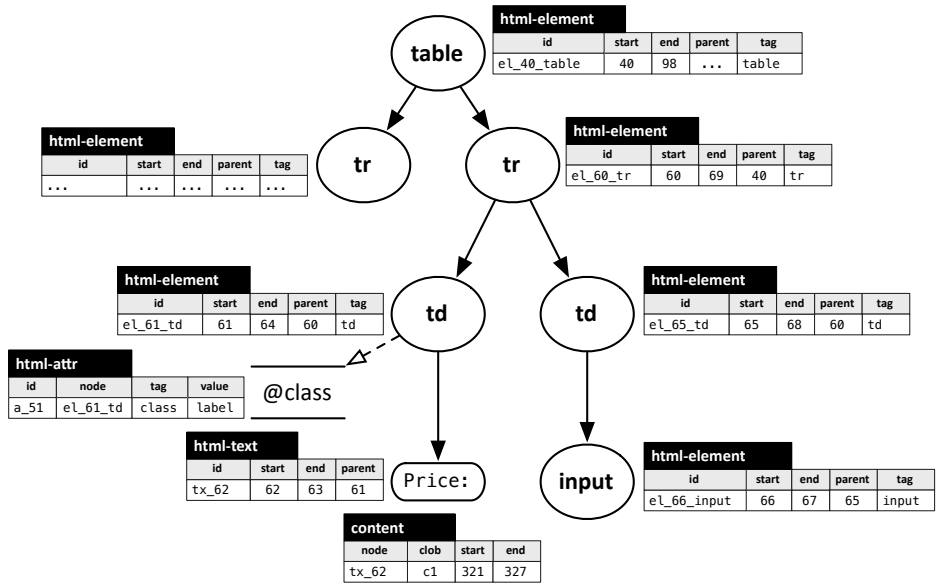

Figure 6: Example (structure and textual content)

3.2 Label Assignment

We use three heuristics for assigning labels to form fields or segments, such that `hasLabel(E,L)` is true, if field or segment `E` has label `L`. An field may be associated with multiple labels. We need to introduce a stratum in the definition of `hasLabel(E,L)` for assigning labels to fields based on their position in a sequence of (partially labeled) fields (see label alignment below). To that end, we introduce `hasBasicLabel(E,L)` and use

```
hasLabel(E,L) ⇐ hasBasicLabel(E,L).
```

to add all basic labels to `hasLabel`.

HTML label. The first heuristics extracts the *HTML labels* from web forms, as shown below (where _ represents a don't care variable):

```
hasBasicLabel(E,L) ⇐  html-element(E,_,_,_,input) ∧
    html-attr(_,E,id,Id) ∧
    html-element(LNode,_,_,_,label) ∧
    html-attr(_,LNode,for,Id) ∧
    child(L,LNode) ∧
    html-text(L,_,_,_).
```

In this rule, we look for HTML form fields `E` and HTML labels `LNode` such that `LNode` refers to `E` via the `for` attribute. We take the text nodes `L` of `LNode` as labels for `E`.

Greatest unique ancestor. The second heuristics uses the *greatest unique ancestor* of a form field `E` to search for labels `L` of `E`. The *greatest unique ancestor* `A` of `E` is the ancestor of `E` closest to the document root that does not contain any other form fields. We say that `E` is the *unique field descendant* of `A` if `A` contains no other form fields beside `E` and define the following rule to obtain the greatest unique ancestor.

```
greatestUniqueAncestor(E,A) ⇐ uniqueDescendant(E,A) ∧
        ¬ (parent(A,A1) ∧ uniqueDescendant(E,A1)).
```

Then, we take `L` as label for `E`, if `E` is a segment (a form field or a group of segments), `L` is a text node, `A` is the greatest unique ancestor of `E`, and `L` is a descendant of `A`.

```
hasBasicLabel(E,L) ⇐ segment(E) ∧
        html-text(L,_,_,_) ∧
        greatestUniqueAncestor(E,A) ∧
        descendant(A,L).
```

This heuristic sometimes yields multiple labels for the same segment.

Label alignment in segments. Our third heuristics *assigns labels based on their position in their parent segment.*

```
hasLabel(E,L) ⇐ partOf(E,G) ∧
        children(Es,G)
        hasNoLabel(Es) ∧
        labelLists(LLs,G) ∧
        sameLength(Es,LLs) ∧
        labelOneToOne(E,L,Es,LLs).
```

In this heuristic, we

1. find a segment `E` which is member of another segment `G` together with its members `Es`,
2. check that no member of `Es` already has a label,
3. identify the list of all label groups in `G`: We retrieve all text nodes that are descendants of the least common ancestor of `Es` and partition this list on each occurrence of a segment from `Es`,
4. check that the list `LLs` of label groups and `Es` have the same length, and
5. map the i-th member of `Es` to all labels in the i-th label group from `LLs`.

If the list of label groups is by one longer than the list of members of `Es` we use the first label as the label for the entire segment:

```
hasLabel(E,L) ⇐ partOf(E,G) ∧
        children(Es,G) ∧
        hasNoLabel(Es) ∧
        labelLists([Ls|LLs],G) ∧
        sameLength(Es,LLs) ∧
        labelOneToOne(E,L,Es,LLs).

hasLabel(G,L) ⇐ partOf(E,G) ∧
        children(Es,G) ∧
```

```
        hasNoLabel(Es) ∧
        labelLists([Ls|LLs],G) ∧
        sameLength(Es,LLs) ∧
        member(L,Ls).
```

We also consider the case when some segments have already assigned a label; however, for space reasons, we do not show the corresponding rules here.

3.3 Classification of Form Elements

From the label assignments and segmentation of the previous section, we obtain "segments", which are either individual form fields or groups of them, together with proper labels. However, these segments need to be classified according to our ontology to provide a complete model of the web form.

To this end, we employ a set of logical rules of which we show here, for space reasons, only those needed for classifying Area Branch Elements, Room Elements, and Price Elements, as well as their facets, such as *Granularity, Price-Type, Category* (Section 4).

Our rules necessitate real estate domain knowledge to classify form fields according to the concepts they implement. In particular, form elements in the ontology are annotated with a set of terms that are used to recognize the labels of these elements, and a specification of values typically appearing as values of corresponding fields, e.g., as options in down lists.

From an extensive corpus of real estate web pages, we have collected such label and value annotations for our ontology concepts. For each concept `C` in the ontology there are predicates `CLabel(L)` and `CValue(V)` to indicate that `L` and `V` are possible labels or values of `C`. The same applies to aspects of ontology concepts.

Each of these predicates contains terms of a particular concept in the real estate domain, represented as logical facts. As an example, consider `areaBranchLabel`, which contains facts such as:

```
areaBranchElementLabel("london").
areaBranchElementLabel("area").
areaBranchElementLabel("place").
...
```

The `priceTypeMinLabel` dictionary (resp. `priceTypeMaxLabel`) includes facts for terms like "min", "minimum" (resp. "max", "maximum"). In general, we classify a form element with concept `C` as follows:

```
C(X) ⇐ leafSegment(X) ∧
    hasLabel(X,Lab) ∧
    CLabel(Y) ∧ label-match(Lab,Y).
```

For instance, for Area Branch Elements we use the following classification rule:

```
areaBranchElement(X) ⇐ leafSegment(X) ∧
    hasLabel(X,Lab) ∧
    areaBranchElementLabel(Y) ∧ label-match(Lab,Y).
```

`leafSegment` is only true for those segments that have no further sub-segments, i.e., only form elements. `hasLabel` is as defined above. `label-match` holds if the the area branch element label `Y` occurs in `Lab`.[1]

Thus, our rule says that a leaf segment, which has a label that matches `areaBranchElementLabel`, is an Area Branch Element. In our running example in Section 1.1, such rules recognize all first five check-boxes as Area Branch Elements.

Another essential concept of real estate web forms is price. Here, we use a set of more sophisticated rules in order to identify Price Elements, and their facets *Type* (min, max, etc.), and *Purpose* (buy or rent). For Price Elements we have the followings:

```
priceElement(X) ⇐ leafSegment(X) ∧
    hasLabel(X,Lab) ∧
    priceElementLabel(Y) ∧ label-match(Lab,Y).
priceElement(X) ⇐ leafSegment(X) ∧
    priceElementValue(Y) ∧ value-match(X,Y).
```

The first rule is analog to the one for Area Branch Element. The only difference is the reference to the labels for Price Element. In the second rule, we retrieve the price element value specification and match that against X. Y is, e.g., a regular expression testing if a contained text node starts or ends with a currency symbol. `value-match` checks if this expression holds on X.

Once we have identified a price element, we try to understand its attributes. In the real estate domain, a price element has either "buy" or "rent" as purpose. If a price element's values are above a threshold (in our case 50,000 expressed with formTypeBuyPriceValue(50000)), that form is likely a form for buying, rather than renting properties.

```
purpose(X,buy) ⇐ priceElement(X) ∧
    formTypeBuyPriceValue(Y) ∧ value-match(X,Y).
purpose(X,rent) ⇐ priceElement(X) ∧
    formTypeRentPriceValue(Y) ∧ value-match(X,Y).
```

As far as price type is concerned, we try to identify attributes min and max by matching their labels and values with the appropriate dictionary. Very often, indeed, price select fields on real estate web forms, include items such as "no minimum", "no maximum". The following rule exploits such knowledge, to derive the assign the "min" price type to a price element.

```
priceType(X,min) ⇐ priceElement(X) ∧
    hasLabel(X,Lab) ∧
    priceTypeMinLabel(Y) ∧ label-match(Y,Lab).
priceType(X,min) ⇐ priceElement(X) ∧
    priceTypeMinValue(Y) ∧ value-match(X,Y).
```

Similar rules are used for "max", "average" and "range" price types.

We conclude this description of our phenomenological rules by showing how we classify a segment as Room Element of *Category* bedroom. For both, we employ proper dictionaries to match labels (and possibly values) for concept Room and attribute *Bedroom*, respectively.

```
roomElement(X) ⇐ leafSegment(X) ∧
    hasLabel(X,Lab) ∧
    roomElementLabel(Y) ∧ label-match(Lab,Y).
roomElement(X) ⇐ leafSegment(X) ∧
   roomElementValue(Y) ∧ value-match(X,Y).
category(X,bedroom) ⇐ roomElement(X) ∧
    hasLabel(X,Lab) ∧
    roomElementCategoryLabel(Y) ∧ label-match(Lab,Y).
category(X,bedroom) ⇐ roomElement(X) ∧
    roomElementCategoryValue(Y) ∧ value-match(X,Y).
```

4. REAL-ESTATE FORM ONTOLOGY

In order to understand the conceptual and the logical structure of a web form, we require a reference description of

[1] We employ regular expressions for this matching task.

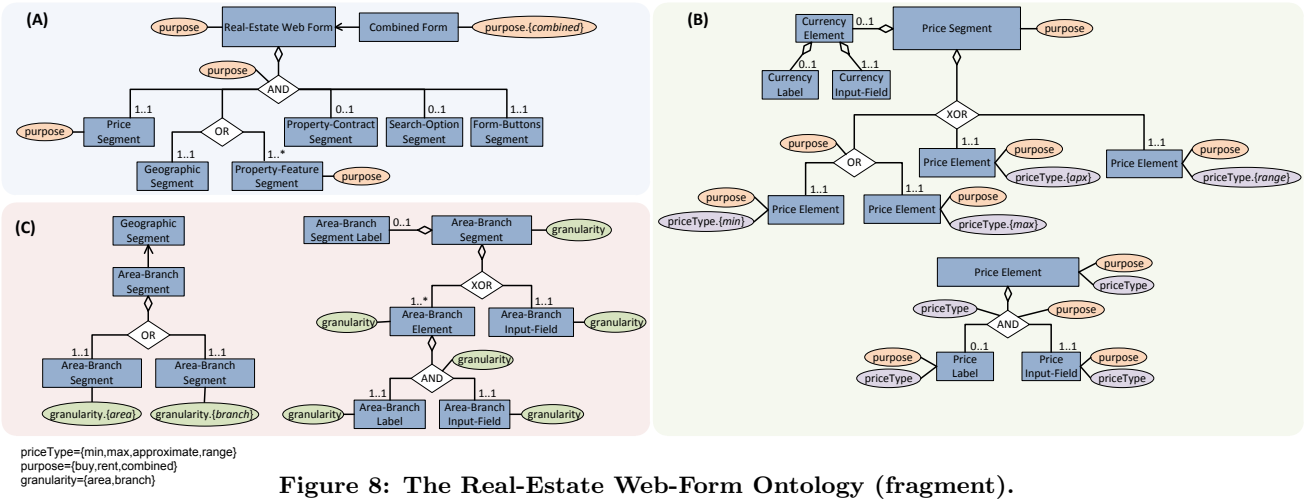

Figure 8: The Real-Estate Web-Form Ontology (fragment).

how such forms occur in practice on the considered websites. In addition, such a description must be formal and machine readable since we need to combine it with the information coming from the browser page model and the phenomenological rules to produce a meaningful model of the web form. Due to the intrinsic object-oriented and hierarchical nature of a web form, a natural solution is to resort to an ontological description [4], where each component of a web form is represented as an ontological concept, corresponding to a class in the object-oriented terminology. The adoption of an ontological language also enables reasoning over such entities by means of standard inference procedures and technologies (e.g., rule-based reasoning or tableaux reasoning).

The role of the ontology in web form understanding is twofold: (i) The ontology *formally models* the design-patterns of web forms. Such reference structures can then be used to perform *coherence checks* against the facts provided by the phenomenological rules. This is needed since the heuristics implemented in phenomenological rules may sometimes provide inconsistent information because there are, in general, multiple valid labeling, grouping, and classification possibilities for the same form elements. (ii) The information extracted by phenomenological rules represents only a partial description of the web form. We exploit reasoning procedures to produce annotations for the untyped components in the form and, in particular, for the form itself.

We developed our ontology through a systematic study of 57 UK real estate sites. During this analysis, we took into account both software and visual design principles in order to identify the most common design-patterns behind the real-estate web forms. The ontology represents the hierarchical structure of a web form in terms of its components, their properties and the relationships among them. The vocabulary of our ontology includes terms to denote basic components of a web form such as *labels*, *buttons* and *input-fields* (e.g., text fields, dropdown lists, check boxes, etc.) along with the logical relationships among them. In particular, we introduce two logical structures that do not always have a counterpart in the DOM tree of the form namely *form elements* and *segments*. A form element is a logical element consisting of an input field and a label describing it. On the other hand, a *segment* denotes a group of form elements with a common goal w.r.t. the user interaction such as a price input-facility or a group of search options. Though in many forms we can find a representative DOM node for these logical concepts, in some cases we need to invent specific representative nodes and inject them into the object hierarchy.

For the representation of such structures, we adopt a simple ontological language consisting of the following constructs:

Classes (or concepts), represented in the ontology as unary first-order predicates and denoting the type of an object in the form e.g., an input field for the number of rooms or a price segment.

partOf relation, represented in the ontology as a binary relation between two concepts and used to encode the hierarchical structure of the components of a web form e.g., the fact that a price label and a price input field are part of a price element.

Attributes, represented in the ontology as binary relations between a concept and an XML data-type (e.g., strings, integers, etc.) and denoting the class attributes. An attribute may, for example, represent the fact that a price field is a minimum/maximum price field or that an order-by input field is ordering in ascending or descending order.

In addition, we need to model some additional constraints for the attributes and the *partOf* relation that are required to fully capture the constraints we identified on the real-estate web forms. In particular we introduce:

1. *Attribute constraints.* These constraints are expressed over the attribute values of certain objects. For example the fact that all such objects with a given type have mandatory values for their attributes or that a group of objects must agree on the values of their attributes.

2. *Cardinality constraints.* These constraints are useful to specify constraints over the hierarchical structure of the ontology. For example the fact that an object cannot be part of two other objects at the same time (i.e., the web forms are conceptually trees) or that a certain component is optional in a web form.

Due to lack of space, we do not describe the entire ontology (available, both as rules and in OWL format, from the project home page at `diadem-project.info/opal`) but only some fragments relevant for our running example.

We use squared boxes to denote the classes of our ontology, ellipses to denote the attributes of a class, "◊" arrows to

denote the *partOf* relation and "→" arrows to express inheritance. In addition, we use the notation $a.\{v\}$ to denote the fact that the attribute a must have the value v. We associate an attribute a to an aggregation operator (i.e., AND, OR and XOR) whenever we require that all the objects participating in the aggregation have compatible values for a. Given two values v_1, v_2 for an attribute a, we say that v_1 and v_2 are compatible if either $v_1 = v_2$ or at least one of them is a null (i.e., unknown) value. The notion of compatibility is easily extended to sets of values.

Figure 8 shows three relevant fragments of the ontology namely: *(A)* the top concepts of our ontology defining how a real-estate web form is constructed, *(B)* the price segments, modelling the structure of a price input/selection facility in a web form and, *(C)* the area/branch segment describing a search facility based on location information i.e., a geographic area (e.g., London) or an area identified by the branch of a real-estate agency (e.g., North London branch).

A real-estate web form (see Figure 8.*A*) necessarily contains a price segment, used to input the property price-range, and a segment containing the buttons for the submission of the form or other operations (e.g., clearing a form). Moreover, it must contain either a geographic-segment, containing location-based search options, or a property-feature segment, describing the features of the property. Optionally, we may find a contract segment defining the type of contract for the property and additional search options, e.g., to order the search results according to some attribute. Each form has a purpose (e.g., buy, rent or combined) represented as an attribute. Since price and property-feature segments also have a purpose attribute, the purpose value must be the same whenever they belong to the same web form. We also say that a combined form, represented in the ontology as a subclass of a real-estate web form, must have "combined" as value for the purpose attribute.

A price segment (see Figure 8.*B*) is composed by a currency element and one or more price elements. The *priceType* attribute is used to denote different price-input facilities occurring in real-estate web forms e.g., we may select a price range, an approximate price, or a pair composed by a minimum and a maximum price. In the latter case, if they occur together, they must agree on the value of the purpose attribute. Each price element consists of a label and a price input field (i.e., a list or a text field).

An area/branch segment (see Figure 8.*C*) is part of a geographic segment and must contain an area/branch element (possibly with a label) providing the type of location-based search that can be used by the user and represented here through the granularity attribute. As usual, an area/branch element consists of a label and an input field.

Using our ontology, we annotate the web form of Figure 1 in a bottom-up fashion: Based on the phenomenologically identified labels and fields, we identify and annotate composite objects on the page, e.g., an element which consists of a label and a field. In particular we associate an area/branch element to the first five pairs of labels and check boxes in the form. All these form elements are then part of an area/branch segment whose label is "Area". On the other hand, the check box denoting the retirement properties and bungalows is annotated as a property-contract segment and the dropdown list for the number of bedroom, along with the corresponding labels, identifies a property-feature segment, while the minimum and maximum price dropdown lists are identified as a price segment. Finally, the order-by dropdown list and the button will be annotated as search-option segment and form-buttons segment respectively. Notice that in some cases, the annotations for form elements and segments can be made directly on an HTML element in the DOM tree e.g., by annotating a `div` element surrounding a group of other HTML elements. However, it is often the case that, for such concepts, there is not a corresponding element in the DOM tree. In these cases, the heuristics of the phenomenological rules generate an artificial bounding box for such elements that are then annotated using the ontology.

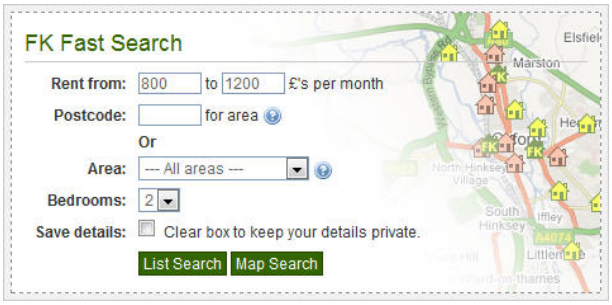

Figure 9: Form on Finders Keepers

5. ANALYSIS AND EVALUATION

We evaluate OPAL with a brief usage study of five real estate web sites. In the second part of this section, we run OPAL on 50 randomly sampled UK web sites and show that OPAL achieves very high precision on these sites.

5.1 Real-Estate Forms: Examples

In this section we discuss some examples of real forms from UK real-estate web sites. Although each form has its own peculiarities, it is possible to capture their logical structure with the few general rules from Section 3 together with the domain ontology from Section 4.

Finders Keepers. The Finders Keepers[2] web form (Figure 9) contains two input fields for the price with interleaved labels, and an input field for restricting the search to a given area determined by a postcode. It is also possible to specify this restriction from a dropdown list. The next input field allows the user to specify the desired number of bedrooms. At the bottom, the form contains two search buttons determining the format of the results page: it is possible to return them as a list of results or as points on an active map.

From a structural point of view the form is organized as a list of HTML `div` elements, one for each row in the form. All these `div` elements are part of another HTML `div` element. Thus, the HTML structure of the form does not quite reflect its logical structure: The rows containing the postcode and area search boxes form together a logical segment as they both represent a way to restrict the search on the basis of the location, but there is not corresponding HTML element. On the other hand, the row containing the term "or" (and represented as a `p` element in the HTML tree) carries the information that the two search boxes are to be used exclusively.

[2] `http://www.finders.co.uk/`.

Figure 10: Buying Form on Vebra

In addition to the logical structure, another interesting aspect that contributes to the understanding of the form's semantics is the analysis of the labels and the content of the drop-lists. For example it is possible to derive that the purpose of this form is to rent a house (and not to buy one) from the terms "rent" and "per month" in the first row. This can also been inferred using the values "800" and "1200" coupled with the pound symbol in the search fields of the first row; it is reasonable to assume that these values cannot represent a sale price for any house. For the price fields we also need to recognize that the user must use the fields to specify a minimum and a maximum price (i.e., a range). This can be done by means of the labels with value "to" and "from" between the two price fields.

Vebra. The Vebra[3] form of Figure 10 is an example of a less sophisticated form. In contrast to Finders Keepers, the location of the property can only be filtered with a free text input field. The price is specified in a similar way to Finders Keepers, but the values are selected from dropdown lists whose content clearly expresses a sale price because it is unlikely to rent a house for 3 million pounds even in London. This can also be easily inferred by the term "for sale" in the header of the form. As on Finders Keepers we have a way to specify the form of the results page (by means of a check box) and the number of bedrooms. The button at the bottom of the form is easily recognized as a search button.

Bell Park Kerridge. The form from Bell Park Kerridge[4] is structurally organized as a grid, where each pair of labels and fields are represented without any HTML element bounding them. As a result the minimum and maximum price fields have the same parent as the location-based search field, the rent/buy selector, the room selector and the submit button. An issue with this web form is to create all the proper logical bounding boxes for the elements in the form, especially for the two price fields that must be grouped together. The purpose (i.e., buy or rent) of this form is not pre-determined but depends on what it is selected by the user in the dropdown list at the upper-left corner of the form.

Harmony Homes. The last web form considered in this section is the one from Harmony Homes[5] and shown in Figure 12. Similarly to the Bell Park Kerridge example, the form has a combined purpose since there is a dropdown

[3] http://www.vebra.com/vebra/.
[4] http://www.bpkestateagents.co.uk/.
[5] http://www.harmony-homes.co.uk/.

Figure 11: Form on Bell Park Kerridge

Figure 12: Form on Harmony Homes

list from which the user can choose whether to search for a house for sale or for rent. But the form is again organized as a list of elements. The only issue with this form is the correct grouping of the minimum and maximum price input fields that are here expressed as dropdown lists whose values change according to what it is selected in the buy/rent dropdown list at the top of the form.

5.2 Real-Estate Forms: Analysis

In the remainder of this section we report on our preliminary statistical analysis of the current state of web forms. We illustrate the result of an *experimental evaluation* of the OPAL approach, in which a random sampling of 50 UK real-estate web sites is considered (this sample has little overlap with the pages used for creating the ontology). In addition, we show that OPAL's runtime for analysing web pages is dominated by the page rendering time.

In order to evaluate the precision of our technique, we annotate by hand all 50 pages our benchmark is composed of. For each of these, we note how many form fields and form segments these pages contain, respectively.

Table 1 summarizes the result for each of the web sites we consider: The URL is given in the first column. The second column contains how many fields the page actually contains, whereas the third and the fourth show, respectively, the percentage of fields and labels OPAL correctly retrieves. On these tasks, our approach obtains an average precision of 97.77% and 96.82%, respectively. While this result itself shows the effectiveness of OPAL, it is worth noting that, only for very few web sites of our benchmark we are not able to perfectly identity fields and labels.

As far as form segmentation is concerned, the fifth column in Table 1 indicates how many segments are present on the corresponding page, whereas the last column reports

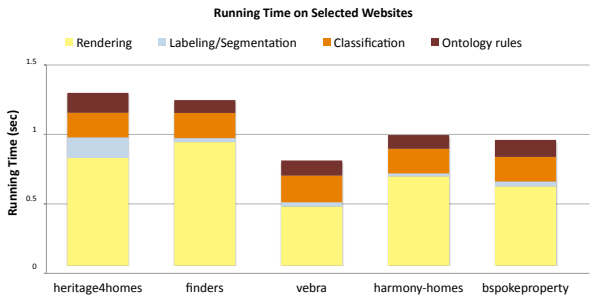

Figure 13: Running Time on Selected Websites

whether OPAL did perform a correct segmentation (✓), otherwise the number of missed segments. We compute the correctness for the segmentation task, as the percentage of found segments out of the total number. We achieved a high rate of 93.33%, which again confirms the overall quality of our approach.

In addition to the above experiments, we conducted an evaluation of the runtime OPAL needs for analyzing a given web page. As Figure 13 shows, processing time is by far dominated by the page rendering time of the underlying browser. However, processing time is yet sensibly small.

6. CONCLUSION AND FUTURE WORK

To the best of our knowledge, OPAL is the first system for automated form understanding that exposes its assumptions explicitly as rules. It demonstrates that a purely declarative specification of the involved heuristics is possible and can yield competitive precision for typical form understanding tasks such as label assignment and segmentation. It also demonstrates the efficacy of analysis rules specialized for a concrete domain.

However, OPAL is only a first-step towards a domain-centric analysis of web pages using declarative rules. We believe that it is a strong indication that such an approach is very promising for web page analysis in general.

There is, however, no parcity of research challenges yet to be addressed:

- Although we present OPAL here for the real-estate domain, we are confident that it can be applied with similar success to other domains. The main effort is the identification of web patterns in that domain as well as the annotation of domain concepts with labels and value specification appropriate for the domain. We plan to investigate the use of machine learning techniques for supporting the creation of the ontology for a new domain.
- OPAL is focused on form understanding. The employed methodology—explicit heuristics in rules based on knowledge of web patterns in a concrete domain—can likely be applied to other types of web objects, e.g., for analyzing result pages or the navigation structure of web sites.
- In form understanding the major challenge is that the segmentation strategy adopted in OPAL is too naive for some web sites. We plan to investigate a segmentation strategy guided by the ontology that also integrates more visual knowledge than used in the current version of OPAL.
- Background knowledge beyond the web patterns represented in the ontology is currently not used in OPAL.
- OPAL generates, for the most part, a single model of the form. In some cases, however, there may not be one clearly preferable model of the form. We plan to investigate probabilistic logic for representing several alternative form models and selecting the one that best matches the constraints expressed in our ontology.

7. REFERENCES

[1] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner. MonetDB/XQuery: a fast XQuery Processor powered by a Relational Engine. In *Proc. ACM Symp. on Management of Data (SIGMOD)*, pages 479–490

[2] E. C. Dragut, T. Kabisch, C. Yu, and U. Leser. A Hierarchical Approach to Model Web Query Interfaces for Web Source Integration. In *Proc. Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 325–336, 2009.

[3] Gargoyle Software Inc. HtmlUnit. Online only, 2010. Retrieved at 14/09/2010.

[4] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[5] H. He, W. Meng, Y. Lu, C. Yu, and Z. Wu. Towards Deeper Unterstanding of the Search Interfaces of the Deep Web. *Word Wide Web*, 10:133–155, 2007.

[6] O. Kalijuvee, O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Efficient Web Form Entry on PDAs. In *Proc. Int'l. World Wide Web Conf. (WWW)*, pages 663–672, 2001.

[7] R. Khare, Y. An, and I.-Y. Song. Understanding Deep Web Search Interfaces: A Survey. *Sigmod Records*, 39(1):33–40, 2010.

[8] N. Kushmerick. Learning to invoke web forms. In *CoopIS/DOA/ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 997–1013, 2003.

[9] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's Deep Web Crawl. In *Proc. of the VLDB Endowment (PVLDB)*, pages 1241–1252, 2008.

[10] H. Nguyen, T. Nguyen, and J. Freire. Learning to Extract From Labels. In *Proc. of the VLDB Endowment (PVLDB)*, pages 684–694, 2008.

[11] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In *Proc. Int'l. Conf. on Very Large Data Bases (VLDB)*, pages 129–138, 2001.

[12] N. Shadbolt, W. Hall, and T. Berners-Lee. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[13] W. Su, J. Wang, and F. H. Lochovsky. ODE: Ontology-Assisted Data Extraction. *ACM Transactions on Database Systems*, 34(2), 2009.

[14] W. Wu, A. Doan, C. Yu, and W. Meng. Modeling and Extracting Deep-Web Query Interfaces. In *Advances in Information & Intelligent Systems*, pages 65–90, 2009.

[15] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the Deep Web. In *Proc. ACM Symp. on Management of Data (SIGMOD)*, 2004.

[16] K. C.-C. C. Zhen Zhang, Bin He. Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. In *Proc. ACM Symp. on Management of Data (SIGMOD)*, 2004.

Evaluation					
	form elements			form segments	
url	total	found(%)	labeled(%)	total	found
rodgersestates.com	8	100.00	100.00	1	✓
bspokeproperty.com	4	100.00	100.00	0	✓
dreweryandwheeldon.co.uk	5	100.00	100.00	0	✓
nicholasestates.co.uk	6	100.00	100.00	0	✓
wilkie.co.uk/main.htm	7	100.00	100.00	0	✓
harveyrobinson.co.uk	6	83.33	83.33	0	✓
henrygeorgeestates.co.uk	9	100.00	100.00	3	✓
dawsonsproperty.co.uk	23	100.00	100.00	6	✓
knightfrank.com	2	100.00	100.00	0	✓
all-about-homes.co.uk	5	100.00	100.00	0	✓
carlisleandborder.com	6	100.00	100.00	0	✓
bernadetteharris.co.uk	7	100.00	100.00	2	✓
harmony-homes.co.uk	6	100.00	100.00	0	✓
kippencampbell.co.uk	3	100.00	100.00	0	✓
jimmcmillan.co.uk	10	100.00	100.00	5	✓
stokesestateagents.co.uk	5	100.00	100.00	0	✓
wrightschurchstretton.co.uk	3	100.00	100.00	0	✓
huwtudor.co.uk	17	100.00	100.00	2	✓
tspc.co.uk	8	100.00	100.00	0	✓
stewartwatson.co.uk	12	100.00	83.33	2	2 missed
morganyork.co.uk	2	100.00	100.00	0	✓
robsoncarter.co.uk	3	100.00	100.00	0	✓
clearwateruk.net	9	100.00	100.00	0	✓
johnhoole.co.uk	8	100.00	100.00	2	✓
hi-m.co.uk	6	100.00	100.00	1	1 missed
qualityhomes.co.uk	13	100.00	100.00	5	✓
bychoice.co.uk	7	100.00	100.00	0	✓
rowelluk.com	9	100.00	77.78	2	1 missed
nicktart.com	17	100.00	100.00	4	✓
lawsonsestateagents.co.uk	5	100.00	100.00	1	✓
christopherbice.co.uk	7	100.00	100.00	1	✓
finders.co.uk	8	100.00	100.00	2	✓
andrewsonline.co.uk	7	100.00	100.00	0	✓
vebra.com	6	100.00	100.00	1	✓
ankerandpartners.co.uk	4	75.00	75.00	0	✓
babingtons.co.uk	5	100.00	100.00	0	✓
bairstoweves.co.uk	2	50.00	50.00	0	✓
cjhole.co.uk	7	100.00	100.00	3	✓
heritage4homes.co.uk	11	100.00	100.00	1	✓
besleyhill.co.uk	5	100.00	100.00	1	✓
countryproperty.co.uk	8	100.00	100.00	0	✓
chestertonhumberts.com	15	100.00	100.00	3	✓
edisonfordproperty.co.uk	5	100.00	100.00	0	✓
edwards-online.co.uk	7	100.00	100.00	1	✓
bruntandfussell.co.uk	5	100.00	100.00	1	✓
geoffreysmith.org	5	80.00	80.00	0	✓
sequencehome.co.uk	7	100.00	100.00	1	✓
hootons.co.uk	14	100.00	100.00	3	✓
lettingzed.co.uk	7	100.00	100.00	0	✓
houseandco.co.uk	13	92.31	84.62	6	✓
		97.61%	96.68%	93.33%	
		average precision		correct segmentation	

Table 1: Precision of labeling and segmentation in OPAL for 50 UK real-estate sites