

# OPAL: A Passe-partout for Web Forms\*

Xiaonan Guo, Jochen Kranzdorf, Tim Furche, Giovanni Grasso, Giorgio Orsi, Christian Schallhart

Department of Computer Science, Oxford University, Wolfson Building, Parks Road, Oxford OX1 3QD  
firstname.lastname@cs.ox.ac.uk

## ABSTRACT

Web forms are the interfaces of the deep web. Though modern web browsers provide facilities to assist in form filling, this assistance is limited to prior form fillings or keyword matching.

Automatic form understanding enables a broad range of applications, including crawlers, meta-search engines, and usability and accessibility support for enhanced web browsing. In this demonstration, we use a novel form understanding approach, OPAL, to assist in form filling even for complex, previously unknown forms. OPAL associates form labels to fields by analyzing structural properties in the HTML encoding and visual features of the page rendering. OPAL interprets this labeling and classifies the fields according to a given domain ontology. The combination of these two properties, allows OPAL to deal effectively with many forms outside of the grasp of existing form filling techniques. In the UK real estate domain, OPAL achieves > 99% accuracy in 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 filling, form understanding, web interfaces, deep web

## 1. INTRODUCTION

Tired of filling web forms with the same contents over and over again? Though browsers increasingly provide assistance for form filling, it is limited to previously filled forms or forms where labels are obviously attached with form fields and use a limited set of keywords. Suppose you want to buy a flat: To get notified of new flats matching your criteria as soon as possible, you need to register with each agency serving your area. Each time you need to provide all of your criteria, in as much as they apply to the registration form provided by the agency. Wouldn't it be great to provide your

\*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 DIADEM, no. 246858. Giorgio Orsi has been supported by the Oxford Martin School, Institute for the Future of Computing.

Copyright is held by the International World Wide Web Conference Committee (IW3C2). Distribution of these papers is limited to classroom use, and personal use by others.

WWW 2012 Companion, April 16–20, 2012, Lyon, France.  
ACM 978-1-4503-1230-1/12/04.

criteria only once, in a passe-partout form, and the browser figures out how fields of a specific form are related to that passe-partout and fills the fields accordingly.

A human can easily do that, recognize most elements and their function, such as date or price fields, by associating textual labels to the corresponding fields and by grouping logically and visually coherent fields together. Once the fields are classified and corresponding values are specified, one can fill the form accordingly. However, automatic approaches to form understanding, see [4], suffer from three main limitations: (i) Domain independence limits them to observations holding across multiple domains, which has been acknowledged in [7, 5]. (ii) Using a single feature class, i.e., visual, textual, or structural properties, compromises the robustness of their results. (iii) Monolithic algorithm design handicaps their adaptability to the continuously changing web. For example, [6, 5] encode specific assumptions on spatial relationships of fields and labels, or [3] works with token classes hard-coded for concepts, such as “min” or “max”. On typical web forms, these approaches therefore identify form fields only with an accuracy around 90% – 92%. Furthermore, they are often limited only to associating fields with labels (form labeling) and do not classify forms with a given schema. For form filling we need such a schema (given by our passe-partout) and near perfect accuracy to minimize manual corrections by the user.

In this demonstration, we therefore present an assisted form filling system based on OPAL (*ontology based web pattern analysis with logic*) [2], a domain-aware form understanding system that achieves near perfect accuracy through a combination of visual, textual, and structural features. In particular, OPAL labels each form in three sequential “scopes”, increasing the size of the page fragment relevant to the analysis of each individual field from a small HTML subtree to a large visual neighborhood: We exploit (i) at *field scope* the structure of the page between fields and labels, (ii) at *segment scope* observations on fields in groups of similar fields, and (iii) at *layout scope* the relative position of fields and texts in the visual rendering of the page. Subsequently, we interpret the form by aligning the form and its labeling with the *domain schema* (containing the concepts of the master form). This interpretation is often imperfect due to missing or misunderstood labels. OPAL addresses this in a repair step, where structural constraints drive the disambiguation and completion of the classification. The necessary constraints are specified in an extension of Datalog to express common patterns as parameterizable templates. A group consisting, e.g., of a minimum and maximum field is a template for generic range specifications.

The demonstration showcases OPAL's analysis and the automatic form filling driven by OPAL. In the OPAL GUI, the user provides form fillings in the master form (passe-partout) for a domain such

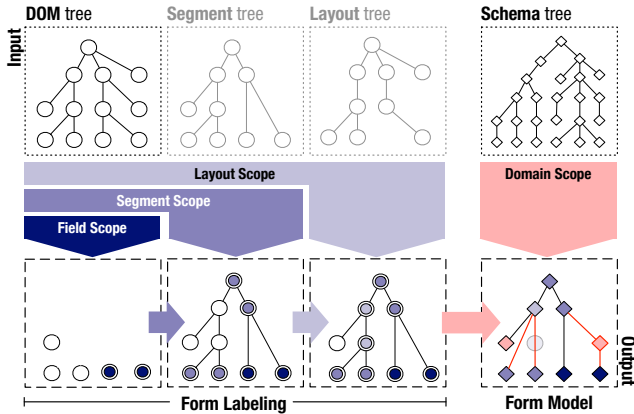


Figure 1: OPAL Overview

as real estate. The system then automatically analyzes any given form and, if recognized as a real estate form, automatically fills it with the values of the master form. To illustrate the inner workings of OPAL, the user can visualize how form labels are associated with form fields in the three scopes above, and where additional adjustments are necessary based on the domain constraints.

## 2. THE OPAL APPROACH

OPAL operates on the *live DOM* of a web page, i.e., the DOM tree computed by the rendering engine of a browser. This implies that all the CSS rules have been applied and that all the client-side scripting code has been executed. The browser also provides the CSS box model, i.e., the bounding boxes of all DOM elements.

When humans interact with forms, they have a clear understanding of the form elements such as buttons, input fields, and the input values to be provided to correctly perform a search. To mimic such an understanding in a software program, the necessary background knowledge must be provided formally.

In OPAL, the needed background knowledge is formalized as a *domain schema*  $\Sigma : \langle \mathcal{T}, \mathcal{C}_{\mathcal{T}} \rangle$  where: (i)  $\mathcal{T}$  is a set of *types*, such as PRICE-FIELD or AREA-MAP, representing logical entities in the form, while (ii)  $\mathcal{C}_{\mathcal{T}}$  consists of (first-order) *constraints* encoding structural relationships among types. The schema depends on the domain and is assumed to be an input of the filling process.

Given a DOM tree  $P$ , a *form model* for a domain schema  $\Sigma$  is a tree  $\mathcal{M} : \langle F, \lambda, \tau \rangle$  where (i)  $F$  is the tree of fields and field segments, (ii)  $\lambda : F \times P$  relates form fields with their labels in  $P$ , (iii)  $\tau : F \rightarrow \mathcal{T}$  types each node in  $F$ . E.g., a form segment is typed as GEOGRAPHIC-SEGMENT if all its children are of type GEOGRAPHIC-FIELD.

*Automated form filling* consists of three tasks: *labeling*, *interpretation*, and *filling* (see Figure 1). The labeling task associates textual labels in the form to the corresponding (groups of) fields, while the interpretation tasks annotates the fields in the form with types from the schema and verifies that such annotation is consistent with the constraints. Finally, the *filling* task, automatically fills the form fields with values and then submits the form.

**Labeling.** OPAL associates labels to fields using a multi-scope approach that simulates the human exploration of a form.

At *field scope*, OPAL considers a single field  $f$  and tries to locate a suitable label  $l$  among the immediate neighbors of  $f$  using only the structure of the DOM tree. In particular, OPAL considers explicit references (e.g., using the *for* attribute) and common DOM ancestors for  $f$  and  $l$  which has no other field descendant.

If a suitable label for  $f$  is not found, OPAL considers at *segment scope* groups of structurally related fields (“segments”) and tries to locate a suitable label for the entire group as well as repeated patterns of interleaved fields and labels. OPAL derives the segments

from the HTML structure, based on similarity between fields, with elimination of segments that are likely to have no semantic relevance and are only introduced, e.g., for formatting reasons.

If the analysis at the first two scopes fails, OPAL locates at *layout scope* labels that are visually related to  $f$  using the CSS box model. We observe a strong preference for placing labels in the *w-nw-n* (i.e., west, north-west, north) visual neighborhood of a field. However, forms often have fields interspersed with field and segment labels, therefore OPAL considers also *overshadowing*. Intuitively, we collect all the labels  $l$  in the *w-nw-n* region of  $f$  and, if they are not overshadowed by another field and contained only in segments that are ancestors of  $f$ , then we consider them labels for  $f$ .

Each of the three labeling scopes considers information not considered in the prior scopes. Their order reflects the higher confidence in earlier stages. In addition, each scope builds on the partial form labeling of the previous scope.

**Interpretation.** The goal of the interpretation task is to turn the form labeling into a model of the form consistent with the domain schema. Intuitively, it represents a further enlargement of the labeling scopes from the given form to the class of all the forms of a domain (*domain scope*). To this end, the textual content of the labels in the form model must be interpreted so that the corresponding fields can be associated with types of the schema. This is done in OPAL by annotating the labels with an entity recognizer such as GATE configured for a given domain (e.g., real estate).

If  $X$  is a node in the form model  $F$  and  $A(X)$  the set of annotations carried by its labels, then the following constraint types  $X$  as PRICE-FIELD if  $X$  has a label annotated with class *price*.

$$\text{PRICE-FIELD}(X) \Leftarrow \text{price} \in A(X)$$

Once the fields have been associated with types, OPAL uses the constraints in the domain schema to (i) verify that the types of the fields and their relationships satisfy the constraints, and (ii) complete a partially specified form model.

As an example, the following constraint checks whether there exist two sibling nodes  $X, Y$  in the form model with a common parent  $Z$  such that one of the two ( $X$ ) is already typed as MINFIELD, i.e.,  $X$  represents a minimum quantity, e.g., for a price or an item. If this is true, the constraint gives to the untyped node  $Y$  the type MAX-FIELD, i.e.,  $Y$  represents a maximum price for the same quantity.

$$\text{MAX-PRICE}(Y) \Leftarrow \text{MIN-PRICE}(X), \text{sibl}(X, Y), \text{PRICE-FIELD}(Y)$$

**Filling.** If the model produced by the interpretation step is consistent with the schema, we proceed to the actual filling of the form. The values provided in the *master form* (the “*pass-partout*”) are automatically filled into the corresponding fields of the current web form. The main challenges in doing so are mapping the free text inputs from the master form to the field types used in the actual form. E.g., if we can not fill the exact value in a select box that does not include the value from the master form, we try to find the closest (for continuous) or most similar (for discrete) value. Failing either we keep the default value and notify the user (by highlighting the field). Furthermore, we apply normalization on the values, e.g., in case of different representations such as “1k” vs. “1000”.

Adapting OPAL to a new domain is a fairly easy task. For the used car domain, e.g., we reused the real estate templates, instantiated with used car concepts. For gazetteers we can, in most cases, exploit existing data sources.

## 3. THE OPAL DEMO

In the demonstration, we showcase OPAL on forms from the real estate and the used car domain. OPAL’s domain independent part is very successful on forms of any domain, but for form filling a

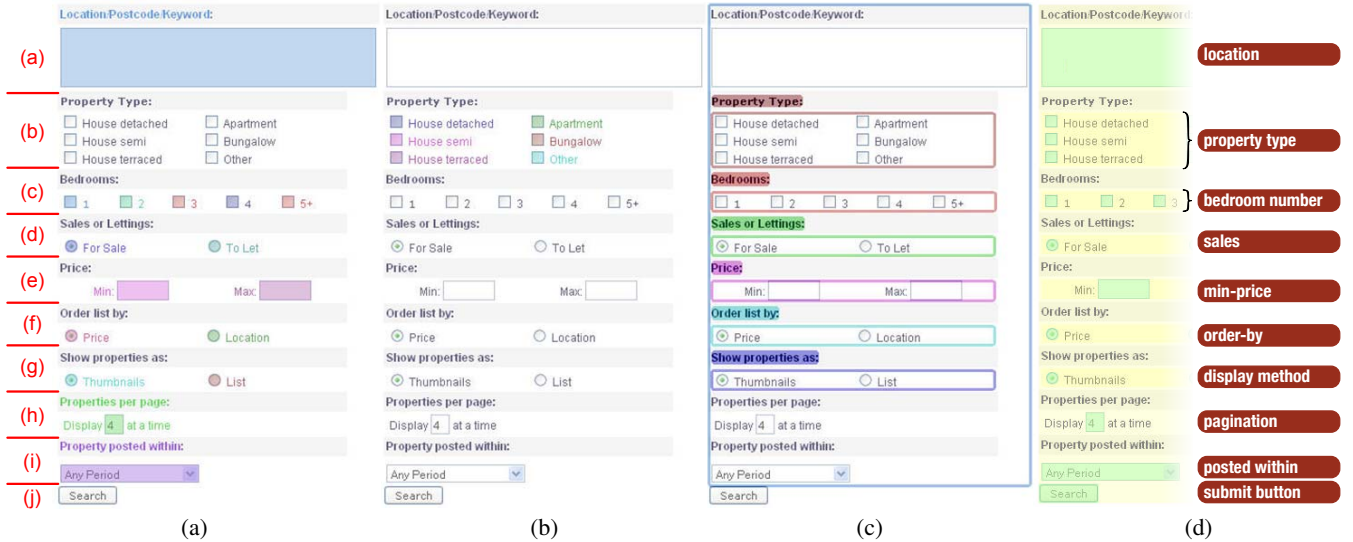


Figure 2: Dawsons multi-scope labeling and typing

domain schema is indispensable. Once the form is interpreted by OPAL we fill it using the values from the master form.

We now discuss step-by-step OPAL’s form understanding and form filling on three real estate forms.

**Dawsons Estate Agents.** Our first example is the form on the site of Dawsons Estate Agents ([dawsonsproperty.co.uk/properties\\_for\\_sale.php](http://dawsonsproperty.co.uk/properties_for_sale.php)) as it appears fairly simple, yet requires most of OPAL’s scopes.

OPAL locates the form on the page and performs its analysis, producing the annotations that are then visualized by the GUI. Dawsons’ original form is shown in Figure 2. Each of the components (a)-(j), each of the fields (c)-(g) and the two groups of checkboxes in (b) are enclosed in a table, tr, or td element. Labels for each of the components such as “Bedrooms:” appear in separate tr’s.

**Field scope.** In our example, OPAL’s field scope correctly assigns all labels to their fields except those of the checkboxes in (b). The reason is that all these labels have a common ancestor in the DOM with their field that is not shared with any other field. In Figure 2a we show this initial form labeling using same color for fields and their labels.

**Segment scope.** The demonstration proceeds by visualizing the output of the segment scope. In our example, the groups (b)-(g) become segments, with (b) further divided into two segments for each of the vertical checkbox groups. Here, OPAL identifies a repeated pattern and each checkbox in (b) is labeled with the text appearing after it as shown in Figure 2b. OPAL also assigns the text in bold face appearing atop each segment as the label for the entire segment (see Figure 2c).

**Layout scope.** On this form, visual features do not produce further label assignments.

**Form interpretation.** Finally, OPAL uses constraints specified in the domain schema to annotate fields and segments and to repair and verify the form interpretation obtained so far. E.g., the first field in (e) is classified as *MIN-PRICE* as we recognize this segment as an instance of a price range pattern. These constraints also disambiguate between multiple annotations, e.g., fields in (f) are annotated with *order-by* and *price*, but the *price* annotation is disregarded due to the group label. For the two checkbox segments in (b), OPAL collapses the two as they are the only children of their

parent segment and both are of the same type. Figure 2d shows the final field classification as produced by OPAL.

**Form filling.** Using the form interpretation, OPAL then fills each field according to the value provided in the master form. This is fairly straightforward in this case (with the exception of the checkbox for bedroom) and OPAL fills the form with 100% success.

**PrimeLocation.** To further detail the form filling process, we use the form [primelocation.com/uk-property-for-sale](http://primelocation.com/uk-property-for-sale), see Figure 3. Here we show the entire OPAL GUI: The top panel allows the user to switch on or off the visualization of the results of OPAL’s scopes. In particular, form fields and associated labels are highlighted with the same colors. Form segments are shown as unfilled boxes with their labels in the same color. The bottom panel shows the master form (OPAL’s *passee-partout*), where the user provides her search requirements. The user can switch between the UK real estate or used car domains and is presented the corresponding fields. Note, that we use free text fields for the values.

In the middle panel of Figure 3, we show [primelocation](http://primelocation.com) with the results of field and segment scope highlighted. For example, “price range” is assigned as segment label for the group containing both price fields which are labeled “minimum” and “maximum” respectively. The screenshot actually shows [primelocation](http://primelocation.com) after OPAL has filled it according to the values from the master form. Notice, how for the three select boxes for minimum and maximum price, as well as bedroom number, OPAL picks the closest value to the one specified in the master form. OPAL can also easily handle variations in the value representation such as “3 bedrooms” (vs. “3” in the master form).

**Holbrook Moran Estate Agents.** Consider the form taken from Holbrook Moran Estate Agents ([holbrookmoran.co.uk](http://holbrookmoran.co.uk)), Figure 4a. At a first glance, this form appears to be simpler than the previous one. Nevertheless, all the four labeling scopes must be used to complete its analysis.

First, at field scope, OPAL correctly labels fields in area (d) and (e). Next, at segment scope, we successfully find the segments for areas (a), (c), and (e). For (a), by recognizing the interleaving pattern between the radio buttons and texts, OPAL associates these texts

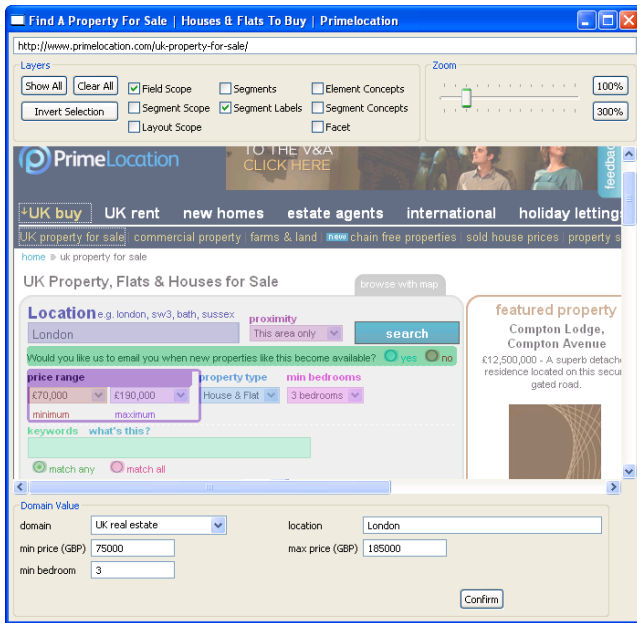


Figure 3: OPAL Interface

correctly with the respective radio buttons. However, in (c), no clear text-field pattern arises to guide the label assignment.

Thus, the first two scopes leave the fields in areas (b) and (c) unassigned. OPAL continues with its analysis by exploiting visual information. At the layout scope (see Figure 4b), “Town / City” in area (b) is the only visible text for the drop down list, since it appears in the top left region of the field and does not belong to the visible region of any other field. Similarly, in area (c) we assign “Price” to both drop down lists.

Finally at domain scope, OPAL classifies the elements and verifies that the obtained results comply with our domain specific form model. There are no ambiguous or superfluous segments in this case.

By leveraging form interpretation, the filling phase requires OPAL to pick the closest matches from all the select boxes. If no match is found, such as for the town/city here (as Holbrook Moran does not serve the London area), the wildcard value is selected and the user is notified.

**Evaluation.** The demonstration proceeds with several other examples from the UK real estate and used car domain. Figure 5a shows precision, recall, and F-score (accuracy) of OPAL for 100 forms from each of these domains. The contribution of each analysis scopes in this experiment is depicted in Figure 5b. In both cases, OPAL achieves nearly perfect form understanding with F-scores close to 99%. We evaluate the form filling on top of this evaluation and observed no case where OPAL understands a form correctly, but does not fill it successfully, except for a few cases where the form changes dynamically or uses heavily scripted UI elements. Figure 5b emphasizes that the combination of the three form labeling scopes and the domain dependent form interpretation are indeed necessary to achieve such a high accuracy. In particular, it is worth pointing out that though it may appear that we achieve high accuracy with the simple form scope only (> 80% in the used car domain), that observation overlooks that the hard task in form understanding are the last 10%. To underline this, we also evaluated OPAL on two publicly available benchmarks, ICQ and

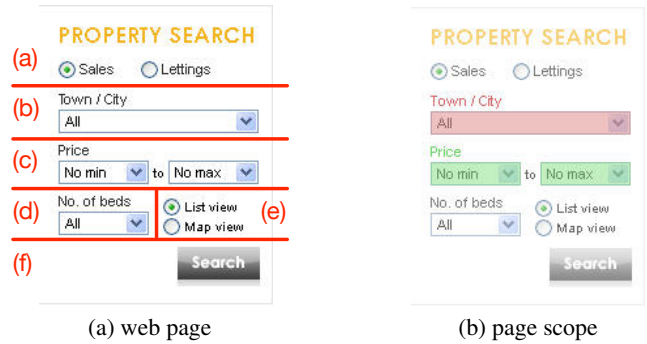


Figure 4: Holbrook Moran form and page-scope labeling

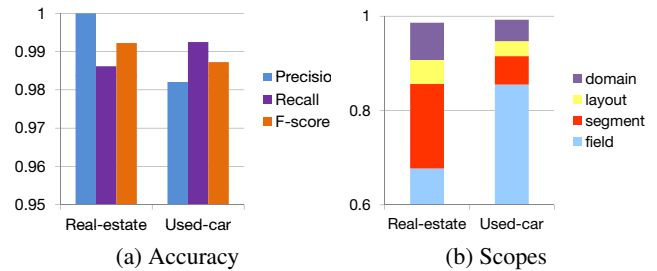


Figure 5: OPAL evaluation

TEL-8 (<http://metaquerier.cs.uiuc.edu/repository/>), using only OPAL’s domain independent form labeling. Even in this case, OPAL easily outperforms existing form understanding systems with > 95% on average for ICQ (where existing systems such as [1] achieve at best 92%).

A screencast of this demonstration is available at [diadem-project.info/opal](http://diadem-project.info/opal).

## 4. REFERENCES

- [1] E. C. Dragut, T. Kabisch, C. Yu, and U. Leser. A hierarchical approach to model web query interfaces for web source integration. *Proc. VLDB Endow.*, 2:325–336, 2009.
- [2] T. Furche, G. Gottlob, G. Grasso, X. Guo, G. Orsi, and C. Schallhart. Real Understanding for Real Estate Forms. In *Proc. of WIMS*, 2011.
- [3] R. Khare and Y. An. An empirical study on using hidden markov model for search interface segmentation. In *Proc. of CIKM*, pages 17–26, 2009.
- [4] R. Khare, Y. An, and I.-Y. Song. Understanding Deep Web Search Interfaces: A Survey. *SIGMOD Record*, 39(1):33–40, 2010.
- [5] H. Nguyen, T. Nguyen, and J. Freire. Learning to extract form labels. *Proc. VLDB Endow.*, 1:684–694, 2008.
- [6] 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.
- [7] K. C.-C. C. Zhen Zhang, Bin He. Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. In *Proc of SIGMOD*, 2004.