How the Minotaur Turned into Ariadne: Ontologies in Web Data Extraction*

Tim Furche, Georg Gottlob, Xiaonan Guo, Christian Schallhart, Andrew Sellers, and Cheng Wang

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

Abstract. Humans require automated support to profit from the wealth of data nowadays available on the web. To that end, the linked open data initiative and others have been asking data providers to publish structured, semantically annotated data. Small data providers, such as most UK real-estate agencies, however, are overburdened with this task—often just starting to move from simple, table- or list-like directories to web applications with rich interfaces.

We argue that fully automated extraction of structured data can help resolve this dilemma. Ironically, automated data extraction has seen a recent revival thanks to ontologies and linked open data to guide data extraction. First results from the DIADEM project illustrate that high quality, fully automated data extraction at a web scale is possible, if we combine domain ontologies with a phenomenology describing the representation of domain concepts. We briefly summarise the DIADEM project and discuss a few preliminary results.

1 Introduction

The web has changed how we do business, search for information, or entertain ourselves to such a degree that saying so has become a platitude. The price for that success is that every business must maintain a website to stay operational. For example, even the smallest real estate agency[1] needs a searchable website and must spend considerable effort to be both found on Google and integrated into the major real estate aggregators. Businesses have reluctantly accepted this cost for doing business as a price for higher visibility—reluctantly, as aggregators present long list of normalised results from all agencies. Thus, agencies have become dependent on dominant aggregators where it is hard to distinguish oneself by reputation, service, or novel features rather than price. Google, on the other hand, is able to pick up on reputation of agencies to some extent, but does very

* 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).

[1] Of which there are over ten thousand in the UK alone, some offering just a handful of properties, e.g., focused on a single burrow of Oxford.

S. Auer, O. Diaz, and G.A. Papadopoulos (Eds.): ICWE 2011, LNCS 6757, pp. 13–27, 2011.

poorly on property searches. This observation holds nowadays for many product domains, with price comparison systems becoming dominant.

One of the goals of the semantic web, linked open data and similar initiatives is to address this dependency on few, dominant aggregators: Let real estate agencies publish their data in a structured format such as RDF with an appropriate ontology describing the schema of that data. Then, search engines can be adapted for object search, i.e., finding the property best fitting specific criteria (such as price range and location). In contrast to aggregators, object search engines should be able to pick up any properly structured data published on the web, just as search engines pick up any properly published HTML page. To publish structured data, an agency has to find a suitable business vocabulary (possibly extending it for novel features of its properties) and mark up its properties according to that schema.

For most agencies, the burden of maintaining and promoting a traditional, HTML web page already incurres considerable cost, requiring expensive training and specialised personnel. Publishing high quality structured data using the most relevant ontologies is a task well beyond the expertise of most agencies: First, there are still few good standardised domain ontologies. In the real estate domain, previous attempts at standardising the vocabulary (e.g., as XML schemata for interchange and use in aggregators) have largely failed, partially as properties are not quite commodities yet. Second, small agencies often manage their properties in an ad-hoc fashion and have little or no knowledge (or need) for proper information management. These concerns are reflected in the fact that the growth of linked open data is certainly driven by adoption in governments, large nonprofits, and major companies.

The availability of structured data from all these agencies promises a fairer, more competitive property market—but with considerable sacrifice. We are stuck in a labyrinth of vocabularies, semantic technologies and standards. Every business will be forced to spend considerable resources on publishing proper structured data, even more than they already do for maintaining their web sites. What begins as a call for everyone to participate, may further harm the competitiveness of small businesses.

We argue that the very same reason that makes the labyrinth scary—the ontologies for annotating structured data—can also direct us out of the labyrinth. What is needed is a **"red thread" program** that automatically finds a path for each given website to relevant domain objects. With such a program, we can analyse the pages that people and businesses are publishing, rather than everyone annotating their objects (and solving the related problems again and again). We consider how to turn such pages into structured data along the following questions:

1. How are these objects represented in web sites (*object phenomenology*)?
2. How do humans find these objects (*search phenomenology*)?
3. How to turn that information into a *"red thread" program.*
4. How to extract *all* the data from relevant pages *at scale.*

In the DIADEM project we are working on answering these questions based on a fundamental observation: If we combine domain with phenomenological (object and search) knowledge in an ontology of a domain's web objects, we can automatically derive an extraction program for nearly any web page in the domain. The resulting program produces high precision data, as we use domain knowledge to improve recognition and alignment and to verify the extraction program based on ontological constraints.

DIADEM's "red thread" program is a large set of analysis rules combined with low-level annotators providing a logical representation of a webpage's DOM, including the visual rendering and of textual annotations (based on GATE). Section 2 gives a brief overview of the DIADEM system and its components, including the ontologies and phenomenologies used for web form (Section 3) and object analysis (Section 4).

We generate an extraction program describing the paths through a web site (and individual pages) to get to the actual data. These programs are formulated in OXPath (Section 5), a careful extension of XPath for simulating user interactions with a web site.

More details on DIADEM are available at `diadem-project.info`.

2 Overview DIADEM Prototype

Figure 1 gives a simplified overview on DIADEM prototype architecture. Every web page is processed in a single sequential pipeline. First we extract the page model from a live rendering of the web page. This model represents logically the DOM, information on the visual rendering, and textual annotations. The textual annotations are generated partially by *domain-specific* gazetteers and rules, but otherwise this part is domain-independent. In the next step, we do an initial classification of web blocks, such as navigation links, advertisements etc. to separate general structures from domain-specific structures and to provide additional clues to object and form analysis. In the third step we use the AMBER prototype, discussed in Section 4 to identify and classify any objects of the domain that may occur on the page. This is done before the form analysis in stage four (using the OPAL prototype from Section 3), as we use the information from the object analysis together with the block classification to decide if navigation links on the page may lead to further data. If the form analysis can identify a form belonging to the target domain, we proceed to fill that form (possibly multiple times). Finally, we extract and crawl links for further exploration of the web site.

Once a site is fully explored, all collected models are passed to the OXPath generator that uses simple heuristics to create a generalised OXPath expression that to be executed with the OXPath prototype for large scale extraction (see Section 5).

This analysis only needs to be repeated if the analyzed site has changed significantly, otherwise the resulting OXPath expression can be used for repeated extraction from the web site.

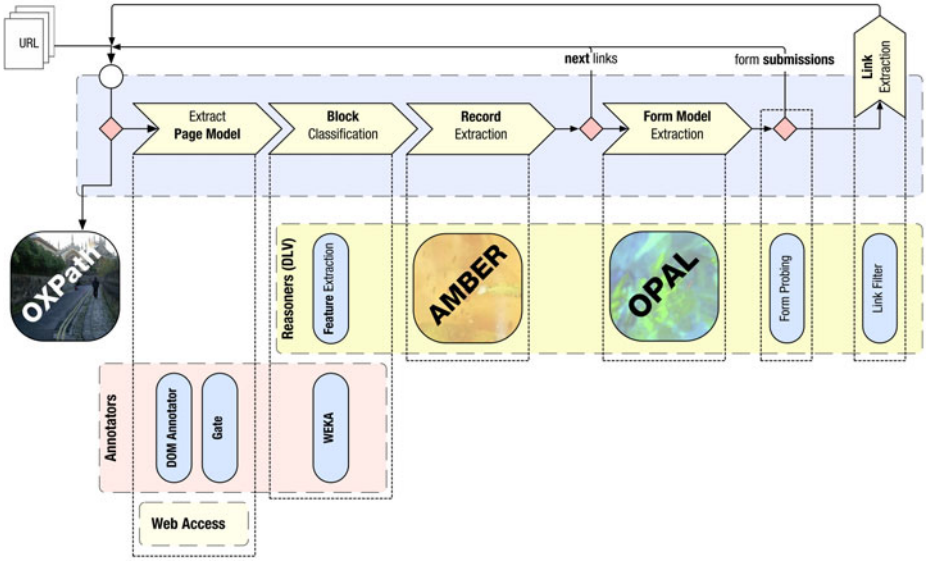

Fig. 1. Overview of the DIADEM 0.1 System

3 Ontologies for Form Analysis

Form understanding, as the gate-way process in automated web data extraction, has been addressed in the context of deep web search [1,2,3,4], web querying [5,6], and web extraction [7]. These approaches focus on observing commonalities of general web forms and exploiting the arising patterns in specifically tailored algorithms and heuristics. However, trying to define a general approach capable of producing high precision results in all domains is not an easy task. Furthermore, by generalizing the assumptions made about web forms, these approaches cannot exploit domain-specific patterns.

To overcome these limitations, we designed an ontology-assisted approach, OPAL (Ontology-based web Pattern Analysis with Logic), for domain-aware form understanding. OPAL analyzes and manipulates on form elements using both general assumptions and domain ontological knowledge. The former adopts several heuristics to provide segmentation and labeling for a form. Form elements and segments are then annotated, classified, and verified using the domain ontology. The link between general form understanding and logical form representation is referred to as the phenomenology, which describes how ontological concepts appear on the web. We have implemented a prototype system for UK real-estate domain, and conducted extensive evaluation of the system on a sampled domain dataset.

OPAL represents information at three successive levels connected by two mappings. Firstly, the page model represents a rendered DOM of a web page through an embedded browser, enriched with visual properties, e.g., bounding boxes of

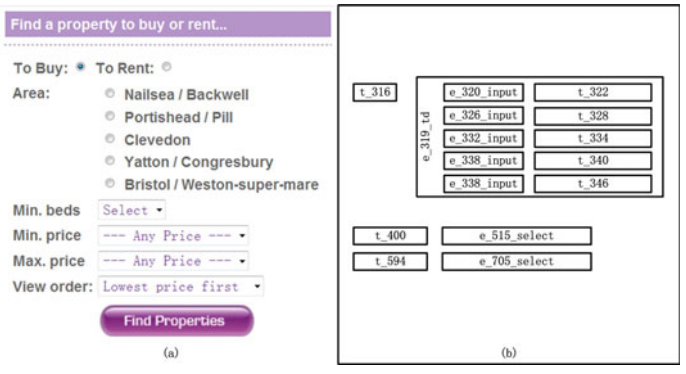

Fig. 2. Form on Heritage with its Page Model

HTML elements. The page model is also expanded with linguistic annotations and machine learning based classifications, relying on domain-specific knowledge. Secondly, the segmentation model describes conceptual relationships between form elements and associates them with texts that are potential labels. This is obtained from the page model through the segmentation mapping, which employs a number of heuristics to build a conceptual segmentation by choosing proper structural and visual relationships among fields and texts. Thirdly, the domain model describes the form as a tree of domain-specific elements, such as price elements. We construct it by joining the domain annotations from the page model with the segmentation results from the segmentation model. The joining process, called phenomenological mapping, is guided by the domain ontology. To adapt OPAL for another domain, one only needs to configure the relevant annotations, define domain specific elements, and instantiate rules for the phenomenological mappings from a set of common templates.

In the following paragraphs, we discuss the OPAL system using `http://www.heritage4homes.co.uk` as our running example (see Figure 2).

3.1 Page Model

The page model represents the structural, textual, and visual properties of the web page as rendered by a browser engine. We represent the DOM tree (including element, text, comment, and attribute) in structural relations and encode the tree structure using the start/end encoding, see Figure 2.

Relying on domain-specific knowledge, the page model is expanded with linguistic annotations and machine learning based classifications on texts appearing on web pages.

3.2 Segmentation Model

Taking the page model, the segmentation mapping labels and groups form elements, such as fields and texts. This mapping exploits heuristics on structural and visual properties and yields the segmentation model. Groups are derived

from similarities in the HTML design and in the visual appearance of form fields. A consecutive list of segments makes a group (parent segment) if they satisfy the similarity conditions and their least common ancestor contains no other segments. We translate this and other similar conditions into rules as shown below (where `Es` refers to a list of segments).

```
group(Es) ⇐ similarFieldSequence(Es) ∧ leastCommonAnc(A,Es) ∧
            not hasAdditionalField(A,Es).
leastCommonAnc(A,Es) ⇐ commonAnc(A,Es) ∧ not(child(C,A) ∧ commonAnc(C,Es)).
partOf(E,A) ⇐ group(Es) ∧ member(E,Es) ∧ leastCommonAnc(A,Es).
```

Labeling, e.g. label assignment to form segments, is achieved through three heuristics, such that `hasLabel(E,L,T)` is true if segment E is assigned with label node L which contains text T. The three heuristics are as follows: **(1)** *HTML label,* which extracts HTML labels from web forms. **(2)** *Greatest unique ancestor,* which finds the greatest unique ancestor of a segment and associates all its text descendants with the segment. **(3)** *Segment alignment,* which does label assignment based on the position of segment and text members. In the last heuristics, we identify the list of all text groups in the parent segment, partitioned by each occurring child segment. If an interleaving situation is encountered, we perform one-to-one assignments between text groups to child segments. We show a fragment of the results produced by the three heuristics below.

```
hasLabel(e_320_input,t_322,"Nailsea / Backwell").
hasLabel(e_326_input,t_328,"Portishead / Pill").
hasLabel(e_515_select,t_400,"Min Price").
hasLabel(e_705_select,t_594,"Max Price").
hasLabel(e_319_td,t_316,"Area").
```

3.3 Domain Model

This model describes conceptual entities on forms as defined in the domain ontology. The ontology provides a reference description of how such forms occur in practice on the considered websites. Figure 3 presents three relevant fragments of our ontology for the UK real estate domain. (A) the top concepts defining how a real-estate web form is constructed, (B) the price segments, modeling the structure of a price input/selection facility, and (C) the area/branch segment describing a search facility based on location information, i.e. a geographic area (e.g. London). In the ontology, classes (or concepts) are represented as unary first-order predicates, denoting the type of an object in the form. The "part of" relation encodes the hierarchical structure of the form segments. Attributes are represented as binary relations between concepts and DOM nodes. For example, it may 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. Furthermore, additional constraints are modeled for attributes (defining domains for attribute values) and relations (cardinality and compatibility constraints). To discuss the ontology, we explain (B) as an example. A price segment is composed by an optional currency element and one or more price elements. The price-Type

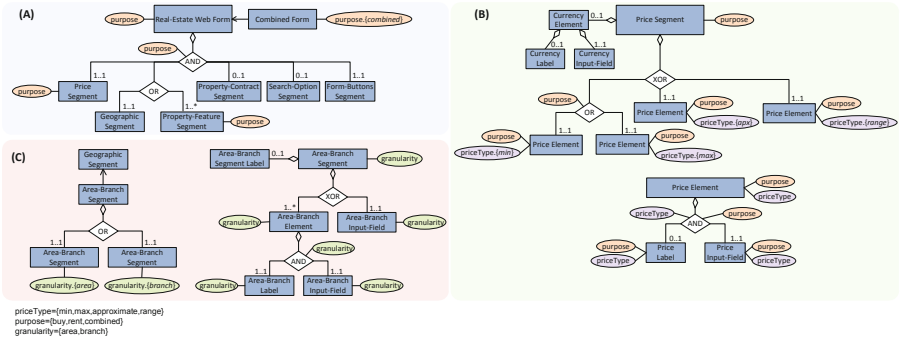

Fig. 3. The Real-Estate Web-Form Ontology (fragment)

attribute is used to denote different price-input facilities occurring in real-estate web forms, e.g. price range selection, a pair composed by a minimum and a maximum price, etc. In case of such a range, the price elements must agree on the value of the purpose attribute (the compatibility constraint). Each price element consists of a label and a price input field.

From the ontology, we derive a phenomenological mapping to classify the form elements and derive the domain form model. This mapping is called *phenomenological* as it connects the abstract concepts of the ontology (e.g. price element) with observable phenomena on the web pages, e.g., a select box with a text label. For example, we annotated "Nailsea" as a location in the extended page model and obtain the result that "Nailsea / Backwell" is associated with the first radio button in the "Area" segment. Hence we conclude that this radio button should be classified as an area/branch element. In this fashion, we associate an area/branch element to the five radio buttons. The second and third dropdown lists are identified as price elements with minimum and maximum for price-type value. It is interesting to note that in some cases, the annotations for form elements can be associated with existing DOM nodes, while in other cases—when a corresponding HTML element does not exist—the phenomenological rules generate an artificial bounding box for such elements which is then annotated using the ontology.

Experiments. We conducted experiments for OPAL up to the segmentation model on a publicly available dataset (ICQ dataset) to test its domain independence and the complete OPAL on a sampled UK real-estate dataset to show the enhancement of ontology. In the former case, we covered 100 query interfaces from 5 domains and achieved 94% F1-score for field labeling and 93% for correct segmentation. For the latter case, where there are 50 randomly selected UK real-estate web sites, we achieved over 97% in field labeling and 95% for the segmentation.

4 Ontologies for Object Recognition and Analysis

We introduce AMBER (*"Adaptable Model-based Extraction of Result Pages"*) to identify result page records and extract them as objects. AMBER is parameterized with a domain ontology which models knowledge on **(i)** records and attributes of the domain, **(ii)** low-level (textual) representations of these concepts, and **(iii)** constraints linking representations to records and attributes. Parametrized with these constraints, domain-independent heuristics exploit the repeated structure of a result page to derive attributes and records. AMBER is implemented to allow an explicit formulation of the heuristics and easy adaptation to different domains.

4.1 Background

There have been a number of approaches for extracting records from result pages, but they were mostly either semi-automated or domain-independent, as surveyed e.g. in [8,9]. In contrast, and as in case of web form analysis, we follow a domain-aware approach: Based on domain-specific annotations in the result page, e.g. marking all occurring rent prices, we identify the occurring data areas, segment the records, and align the attributes of the found records.

Our approach works in four steps: During the **(i)** *retrieval* and **(ii)** *annotation stage,* the *page* and *annotation model* are obtained to represent the DOM of a live browser and relevant domain-specific markup. **(iii)** The **phenomenological mapping** constructs an *attribute model* which summarizes the annotations into potential record attributes occurring on the analyzed web page. **(iv)** The **segmentation mapping** uses the structural and visual information from the browser model and the attributes identified in the attribute model to locate data areas and segment these areas into individual data records. As a result, we obtain the *result page model* for the given page.

4.2 Algorithm Description

Due to different representations for the same content on different web sites, automatic data extraction usually results in a complex and time-consuming task. Existing approaches mostly try to detect those repeated structural patterns in the DOM tree that represent data records. This approach has the advantage of being domain independent because it relies only on the structural similarities between different records (within the same or among different pages). However, we can safely say that all past *domain-independent* attempts describing *all* possibly occurring page structures have failed.

In our approach, we combine the detection of repeated structures with background knowledge of the domain. We provide the analysis process with a semantic description of the data that we expect to find on the web page, plus a set of "constraints" that are known to hold in the domain. Our experiments show that this combination results in a much more precise analysis and enables a simple consistency check over the extracted data.

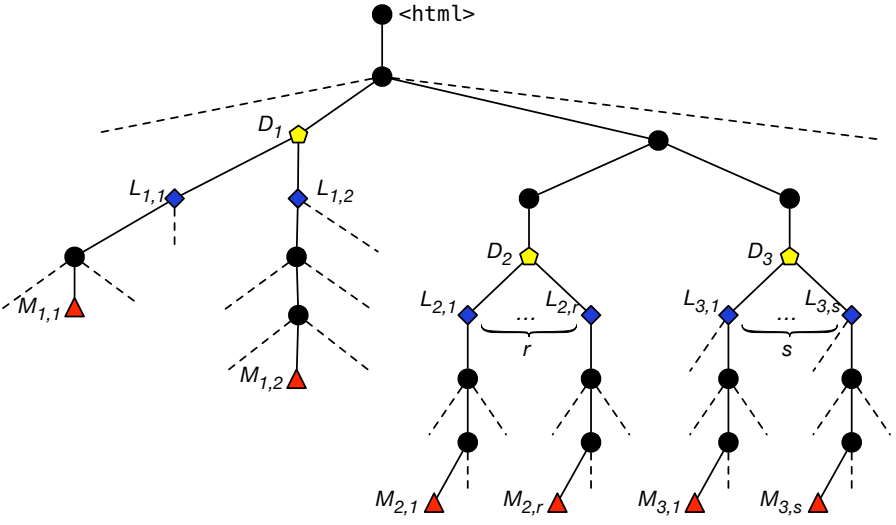

Fig. 4. Results on Zoopla

Data Area Identification. A data area in a result page is identified by leveraging *mandatory elements*. A mandatory element is a domain concept that appears in all records of a given data area, e.g., the location in a real-estate website. Since in this phase the records are yet to be discovered, the mandatory elements are identified by matching the content of the text nodes with a domain ontology.

Since the matching process is intrinsically imperfect, we allow false-positives during the identification of mandatory nodes. To reduce the false-positives among the matched mandatory nodes (MD-nodes) , we group MD-nodes with same (or similar) *depth* in the DOM and similar *relative position* among their siblings. We then consider only MD-nodes belonging to the largest group and discard other nodes. The least common ancestor in the DOM tree of all identified MD-nodes is considered the data area root.

Because it is possible that a result page contains several data areas, we repeat the data area identification process and choose the largest group as a data area until the largest group contains only one record (probably noise) or we eliminate all groups. In Figure 4, D_1 and D_2 are data areas.

Record Segmentation. The records within a data area are identified as sub-trees rooted at children of the data area root. The segmentation process uses *record separators*, i.e., sub-trees interleaved with records, typically containing neither text nor a URL. As a first step, each subtree in the DOM containing a single MD-node and rooted at a direct child of the data area root is considered a *candidate record*. A subsequent step tries to "expand" the candidate record to adjacent subtrees in the DOM. We therefore consider the siblings of the candidate record. If they are record separators, we consider each candidate record as a proper record; otherwise we apply the following steps: **(1)** Compute the *distance* l between two candidate records as the number of siblings between their root

nodes. **(2)** Consider all possible $2 \times (l-1)$ offsets for record boundaries, compute the similarity between the identified records, and choose the one with highest similarity. **(3)** Whenever several expansions have the same similarity, we choose the one with the highest structural similarity among records. To break ties, we pick the one delimited by the largest number of record separators.

Data Alignment. After the segmentation phase, it is possible to have inconsistent mappings between the intended record structure and the DOM tree. For example, we might have multiple assignments for functional attributes (e.g., two prices for the same apartment) or we might miss the assignment for a mandatory attribute (e.g., apartments 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. Since data cleaning and reconciliation is out of the scope of this work, we rely on state-of-the-art techniques for these tasks [10]. In particular, since we already exploit domain knowledge during the analysis, we will leverage on the constraints of the result-page model and of the domain ontology to disambiguate multiple assignments and identify missing attributes.

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 the records (considering the DOM document order) and nodes with synthetic content (i.e., the text-node length). The reason is that meaningful content usually appears in the top left corner of the records (i.e., the corresponding nodes will appear early in document order) and they appear in the form of synthetic text.

4.3 Evaluation

We report on our preliminary statistical analysis of the current state of result pages in this section. We illustrate the result of an *experimental evaluation* of the AMBER approach. AMBER has been evaluated on 50 randomly selected UK real-estate web sites from 2,810 UK real-estate web sites from yellow page. In order to evaluate both precision and recall of our technique, after randomly chosen 50 web sites, we chose one or two result pages from each site (some sites have only one result page), and annotated them manually. For each result page, we annotated both the position and content of data area, the position and content of each record and the position and content of each data attributes.

AMBER reaches 100% on both precision and recall of 126 data areas, and 100% precision and 99.8% recall for 2101 records, 99.4% precision and 99.0% recall for 6733 data attributes. While this result itself shows the effectiveness of AMBER, it is worth noting that our program could not perfectly identify all records on only 2 web sites. Although the extraction of data attributes is based on a raw, incomplete ontology, we still achieve a very high rate of 99.4% precision and 99.0% recall for 6733 data attributes.

5 Web Scale Extraction with OXPath

OXPath is the DIADEM formalism for automating web extraction tasks, capable of scaling to the size of the web. A properly specified OXPath expression outputs *structured web objects* adhering to knowledge encoded in domain-specific ontologies consistent with a known phenomenology.

Extracting and aggregating web information is not a new challenge. Previous approaches, in the overwhelming majority, either (1) require service providers to deliver their data in a structured fashion (e.g. the Semantic Web); or, (2) "wrap" unstructured information sources to extract and aggregate relevant data. The first case levies requirements that service providers have little incentive to adopt, which leaves us with wrapping. Wrapping a website, however, is often tedious, since many AJAX-enabled web applications reveal the relevant data only through user interactions. Previous work does not adequately address web page scripting. Even when scripting is considered, the simulation of user actions is neither declarative nor succinct, but rather relies on imperative scripts.

5.1 Language

OXPath extends XPath 1.0 with four conceptual extensions: Actions to navigate the user interface of web applications, exposure to rendered visual information, extraction markers to specify data to extract, and the Kleene star to facilitate iteration over a set of pages with an unknown extent.

Actions. For simulating user actions such as clicks or mouse-overs, OXPath introduces *contextual*, as in `{click}`, and *absolute action steps* with a trailing slash, as in `{click /}`. Since actions may modify or replace the DOM, we assume that they always return a new DOM. Absolute actions return DOM roots, contextual actions return the nodes in the new DOM matched by the *action-free prefix* of the performed action, which is obtained from the segment starting at the previous absolute action by dropping all intermediate contextual actions and extraction markers.

Style Axis and Visible Field Access. We introduce two extensions for lightweight visual navigation: a new axis for accessing CSS DOM node properties and a new node test for selecting only visible form fields. The `style` axis navigates the actual CSS properties as returned by the DOM *style* object. For example, it is possible to select nodes based on their (rendered) color or font size.

To ease field navigation, OXPath introduces the node-test `field()`, which relies on the `style` axis to access the computed CSS style to exclude fields that are not visible, e.g., `/descendant::field()[1]` selects the first visible field in document order.

Extraction Marker. In OXPath, we introduce a new kind of qualifier, the *extraction marker*, to identify nodes as representatives for records as well as to form attributes for these records. For example,

```
doc("news.google.com")//div[contains(@class,"story")]:<story>
    [.//h2:<title=string(.)>]
    [.//span[style:color="#767676"]:<source=string(.)>]
```

extracts a `story` element for each current story on Google News, along with its title and sources (as strings), producing:

```
<story><title >Tax cuts ...</title>
    <source>Washington Post</source>
    <source>Wall Street Journal</source> ...  </story>
```

The nesting in the result mirrors the structure of the OXPath expression: extraction markers in a predicate (`title`, `source`) represent attributes to the last marker outside the predicate (`story`).

Kleene Star Finally, we add the Kleene star, as in [11]. For example, the following expression queries Google for "Oxford", traverses all accessible result pages and extracts all links.

```
doc("google.com")/descendant::field()[1]/{"Oxford"}
                 /following::field()[1]/{click /}
    /( /descendant::a:<Link=(@href)>[.#="Next"]/{click /} )*
```

To limit the range of the Kleene star, one can specify upper and lower bounds on the multiplicity, e.g., `(...)*{3,8}`.

5.2 Example Expression

The majority of OXPath notation is familiar to XPath users. In the previous section, we carefully extend XPath to achieve desired automation and extraction features. Consider now a full expression, shown in Figure 5. In this example, we define an OXPath expression that extracts prices of properties from `rightmove.co.uk`. We begin in line 1 by retrieving the first HTML page via the `doc(url)` function, which OXPath borrows from XQuery. We continue through line 3 by *simulating* user action for many different form input fields, spread over multiple HTML pages. Note here that OXPath allows the use of the CSS selectors `#` and `.`, which allows selection of nodes based on their `id` and `class` attributes, respectively. Line 4 uses the Kleene star to specify extraction from all possible result pages, which are traversed by clicking on hyperlinks containing the word "next". Finally, line 5 identifies all relevant properties and extracts their corresponding prices. This example could be extended to incapsulate all attributes relevant to each found web object, which in this example are all rentable properties from this site that satisfy our criteria.

5.3 System

OXPath uses a three layer architecture as shown in Figure 6:

(1) The **Web Access Layer** enables programmatic access to a page's DOM as rendered by a modern web browser. This is required to evaluate OXPath expressions which interact with web applications or access computed CSS style


```
doc("rightmove.co.uk")/descendant::field()[1]/{"Oxford"}
/following::input#rent/{click/}//select#minBedrooms/{"2"/}
//select#maxPrice/{"1,750 PCM"/}//input#submit/{click/}
///(//a[contains(.,"next")])/{click/})*
//ol#summaries/li:<property>[//p.price:<price=string(.)>];
```

Fig. 5. OXPath for Rental Properties in Oxford

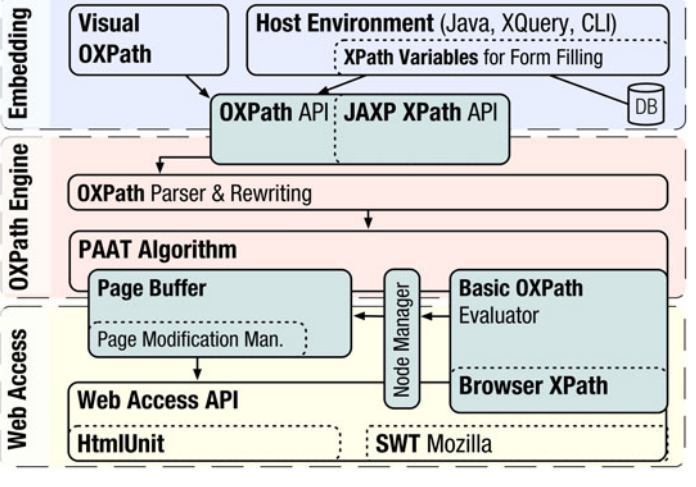

Fig. 6. OXPath System Architecture

information. The web layer is implemented as a facade which promotes interchangeability of the underlying browser engine (Firefox and HTMLUnit).

(2) The **Engine Layer** evaluates OXPath expressions.Basic OXPath steps, i.e., subexpressions without actions, extraction markers and Kleene stars, are directly handled by the browser's XPath engine.

(3) The **Embedding Layer** facilitates the integration of OXPath within other systems and provides a host environment to instantiate OXPath expressions. The host environment provides variable bindings from databases, files, or even other OXPath expressions for use within OXPath. To facilitate OXPath integration, we slightly extend the JAXP API to provide an output stream for extracted data.

Though OXPath can be used in any number of host languages such as Java, XSLT, or Ruby, we designed a lightweight host language for large-scale data extraction. It is a Pig Latin[12]-like query language with OXPath subqueries, grouping, and aggregation. We separate these tasks, as well as the provision of variable bindings, from the core language to preserve the declarative nature of OXPath and to guarantee efficient evaluation of the core features.

OXPath is complemented by a visual user interface, a Firefox add-on that records mouse clicks, keystrokes, and other actions in sequence to construct an equivalent OXPath expression. It also allows the selection and annotation of nodes used to construct a generalised extraction expression. We are actively improving the visual interface and developing a visual debugger for OXPath.

5.4 Further Reading

We have limited the scope of the discussion here to fundamental aspects of the OXPath formalism. For further details, please see [13]. In particular, this work introduces the PAAT (page-at-a-time) algorithm that evaluates OXPath expressions with intelligent page caching without sacrificing the efficiency of regular XPath. In this way, PAAT guarantees polynomial evaluation time and memory use independent of the number of visited pages. Further, this work highlights experimental results of the current prototype. These experimental results validate our strong theoretical time and memory guarantees. OXPath performs faster than comparable systems by at least an order of magnitude in experiments where a constant memory footprint for OXPath can be empirically observed. No observed competitor managed memory as intelligently as PAAT: either all target pages were cached (requiring linear memory w.r.t. pages visited) or a fixed number of pages were cached (requiring pages to be revisited in the general case). For example applications of OXPath over real-world web data, please see [14,15].

References

1. Nguyen, H., Nguyen, T., Freire, J.: Learning to Extract From Labels. In: Proc. of the VLDB Endowment (PVLDB), pp. 684–694 (2008)
2. Su, W., Wang, J., Lochovsky, F.H.: ODE: Ontology-Assisted Data Extraction. ACM Transactions on Database Systems 34(2) (2009)

3. Kushmerick, N.: Learning to invoke web forms. In: Chung, S., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 997–1013. Springer, Heidelberg (2003)
4. Shadbolt, N., Hall, W., Berners-Lee, T.: The Semantic Web Revisited. IEEE Intelligent Systems 21(3), 96–101 (2006)
5. Wu, W., Doan, A., Yu, C., Meng, W.: Modeling and Extracting Deep-Web Query Interfaces. In: Advances in Information & Intelligent Systems, pp. 65–90 (2009)
6. Dragut, E.C., Kabisch, T., Yu, C., Leser, U.: A Hierarchical Approach to Model Web Query Interfaces for Web Source Integration. In: Proc. Int'l. Conf. on Very Large Data Bases (VLDB), pp. 325–336 (2009)
7. Raghavan, S., Garcia-Molina, H.: Crawling the Hidden Web. In: Proc. Int'l. Conf. on Very Large Data Bases (VLDB), pp. 129–138 (2001)
8. Chang, C.-H., Kayed, M., Girgis, M.R., Shaalan, K.F.: A survey of web information extraction systems. IEEE Trans. Knowl. Data Eng. 18(10), 1411–1428 (2006)
9. Laender, A.H.F., Ribeiro-Neto, B.A., da Silva, A.S., Teixeira, J.S.: A brief survey of web data extraction tools. SIGMOD Record 31(2), 84–93 (2002)
10. Batini, C., Scannapieco, M.: Data Quality: Concepts, Methodologies and Techniques. Springer, Heidelberg (2006)
11. Marx, M.: Conditional xpath. ACM Trans. Database Syst. 30(4), 929–959 (2005)
12. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 1099–1110. ACM, New York (2008)
13. Furche, T., Gottlob, G., Grasso, G., Schallhart, C., Sellers, A.J.: Oxpath: A language for scalable, memory-efficient data extraction from web applications. In: Proc. of the VLDB Endowment PVLDB (2011) (to appear)
14. Sellers, A., Furche, T., Gottlob, G., Grasso, G., Schallhart, C.: Taking the oxpath down the deep web. In: Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT 2011, pp. 542–545. ACM, New York (2011)
15. Sellers, A.J., Furche, T., Gottlob, G., Grasso, G., Schallhart, C.: Oxpath: little language, little memory, great value. In: Proceedings of the 20th International Conference Companion on World Wide Web, WWW 2011, pp. 261–264. ACM, New York (2011)