Effective Web Scraping with OXPath*

Giovanni Grasso, Tim Furche, and Christian Schallhart
Department of Computer Science, Oxford University, Wolfson Building, Parks Road, Oxford OX1 3QD
firstname.lastname@cs.ox.ac.uk

ABSTRACT

Even in the third decade of the Web, scraping web sites remains a challenging task: Most scraping programs are still developed as ad-hoc solutions using a complex stack of languages and tools. Where comprehensive extraction solutions exist, they are expensive, heavyweight, and proprietary.

OXPath is a minimalistic wrapping language that is nevertheless expressive and versatile enough for a wide range of scraping tasks. In this presentation, we want to introduce you to a new paradigm of scraping: declarative navigation—instead of complex scripting or heavyweight, limited visual tools, OXPath turns scraping into a simple two step process: pick the relevant nodes through an XPath expression and then specify which action to apply to those nodes. OXPath takes care of browser synchronisation, page and state management, making scraping as easy as node selection with XPath. To achieve this, OXPath does not require a complex or heavyweight infrastructure. OXPath is an open source project and has seen first adoption in a wide variety of scraping tasks.

Categories and Subject Descriptors

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

General Terms

Languages, Experimentation

Keywords

web extraction, web automation, XPath, AJAX, deep web

1. INTRODUCTION

Web scraping and extraction has long suffered from a dearth of off-the-shelf tools readily available to web programmers. All too often scraping programs ("web wrappers") are still implemented ad-hoc using a complex stack of languages and tools. Integrated solutions, of which [3] and [2] are prominent examples, are proprietary, heavyweight, and costly.

Therefore, we have introduced OXPath [6] as a light-weight, open source web scraping language and tool. OXPath is designed as a minimalistic extension of XPath that adds *just four* features to XPath. Nevertheless, it is sufficient for solving many web scraping tasks, as demonstrated in Section 3. By building on XPath, an established technology widely used for web development, OXPath's users can leverage existing tools (such as as Firebug[1] or FirePath[2]) for inspecting a webpage's structure and finding suitable XPath expressions for selecting the relevant nodes. OXPath extends XPath with just four concise extensions:

(1) OXPath extends XPath with selection based on *HTML specifics and visual* features by exposing all *CSS properties*. For example, `//span.price[style::color='red']` selects all `span`'s with class "price", if they are also red.

(2) OXPath allows the *execution of user actions* (e.g., click, form filling) on selected nodes to interact with the scripted interfaces of web applications. For example, `//input#submit/{click }` clicks on the `input` element with id "submit".

(3) OXPath deals with navigation through data paginated over many web sites through bounded or unbounded *navigation sequences*: `(//a#next/{click /})*//h1` selects all `h1`'s on a page reached by clicking on the a with id "next" on the current page P_0, the page P_1 reached by such a click on P_0, ..., the page P_k reached by such a click on P_{k-1}.

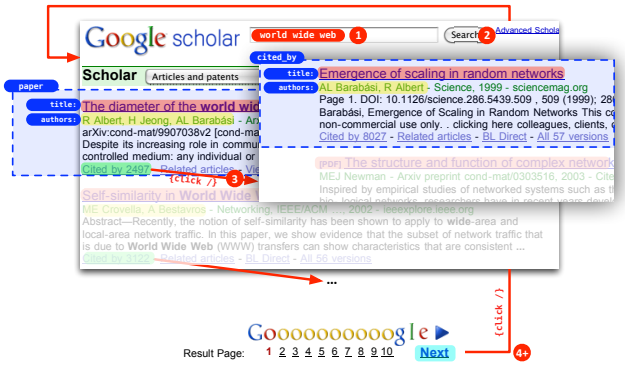
(4) OXPath enables the identification of *data for extraction,* which can be assembled into (hierarchical) records, regardless of its original HTML structure. These records can be stored in a relational database, as XML or RDF, or processed through a streaming API. E.g., `//div.result/span[1]:<price=string(.)>` extracts the string-value of each selected node as a "price".

Like XPath, OXPath manages most aspects of the actual evaluation of these expressions automatically, without requiring the user to consider page or state management, browser synchronisation, or network timing. For rare cases, where OXPath's built in strategies do not suffice, the user can provide additional hints in form of timeouts or waits after executing an action (similar to WebDriver's explicit and implicit waits). All these features are achieved without sacrificing performance: OXPath scales very well both in time and in memory and uses very little resources compared to other web data extraction tools (for an extensive evaluation, see [5]). Specifically, its *memory requirements are independent* of the number of pages visited. This is achieved by aggressive page buffering during a depth-first traversal of pages, combined with memoization of intermediate expression results. To the best of our knowledge, OXPath is the first web extraction tool with such a guarantee. Low resource usage is crucial for use as an open information access tool, as it translates into low cost when deployed in the cloud.

*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.

WWW 2013 Companion, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

[1] `https://getfirebug.com/`
[2] `http://code.google.com/p/firepath/`




```
doc("scholar.google.com")//
descendant::field()[1]/{"world wide web"}❶
 /following::field()[1]/{click/}❷
 //(//a[contains(string(.),'Next')]/{click/})*❹
 //div.gs_r:<paper>
  [.//h3:<title=string(.)>]
  [.//*.gs_a:<authors=substring-before(.,' - ')>]
  [.//a[.~'Cited by']/{click /}❸
 //div.gs_r:<cited_by>[.//h3:<title=.>]]
```

Figure 1: OXPath on Google Scholar

Following this brief introduction of OXPath, we demonstrate the ease of use by two examples in Section 2. OXPath is starting to see use in a number of scraping tasks by a growing group of users. In Section 3, we briefly sketch some of these user stories to give an idea of the versatility, but also the limits of OXPath. We conclude in Section 4 with an overview of current development plans.

2. OXPATH BY EXAMPLE

OXPath and its features are best illustrated by means of two examples. Figure 1 depicts the actions necessary to **(1)** fill the form on Google Scholar, **(2)** navigate through the paginated results, **(3)** extract bibliographic information from these pages. At line 2 and 3, the expression locates the text input and search button fields respectively. On the former, a typing action is performed to fill the value "world wide web" (❶), while on the latter a click is executed to submit the form (❷). Note the use of OXPath's node-test `field()`, which is meant to ease navigation among visible fields only. OXPath supports all kind of actions necessary to fill forms, e.g., selection of options, checking check-boxes, selecting radio buttons.

Figure 1 continues on the result page thus reached. OXPath inherits from XPath the capability to precisely identify nodes on a page, and extends it to specify data for extraction. This is achieved by *extraction markers* such as `:<paper>` in line 5, which creates a record for every paper on that page, each consisting of title and authors attributes specified by attribute extraction markers (e.g., `:<title=string(.)>`). An important feature of OXPath is the support for extracting nested records. For example, at line 9 a new record marker `:<cited_by>` is nested in order to extract all citing papers of the `<paper>` at hand, together with their own attributes (only `<title>` shown for brevity). Note that, in order to collect all citing papers for the paper at hand, the OXPath expression `clicks` on the "Cited by" link (❸). Finally, we are able to repeat the actions aforementioned on all result pages, by means of OXPath's iteration support. In line 4 (❹), the expression is performed until matched, hence a click action is executed upon the "next" link of each result page.

This simple OXPath expression is thus powerful enough to scrape thousands of results from Google Scholar. OXPath natively supports output as XML, RDF, and into a relational database. As a second example, we show how OXPath can be used to turn web-

```
doc("http://wwagency.co.uk/")//input[@name='search']/{click/}
/(descendant::a[@class='pagenum']/{click/})*
  /descendant::div.proplist_wrap:<(gr:Offering)>
  [.//span.prop_price:<dd:hasPrice(xsd:double)=
    substring-after(.,'£')>]
  [.//a[@class='link_fulldetails']:<foaf:page=string(@href)>]
  [.:<gr:includes(dd:House)>
    [.//h2:<gr:name=string(.)>]
    [.//h2:<vcard:street_address=string(.)>]
    [.//div.prop_info//strong[1]:<dd:bedrooms=string(.)>]
    [.//img:<foaf:depiction=string(@src)>]
```

Figure 2: OXPath RDF wrapper

sites into RDF triples. Figure 2 illustrates an expression that extracts property offerings from `http://wwagency.co.uk`, a real estate website in UK.

Lines 1 and 2 are similar to the previous example: form submission and iteration on all result pages. This expression employs *RDF extraction markers*, that allow to generate both data and object properties including proper type information and object identities. At line 3, for instance, each match of the extraction marker `<:(gr:Offering>` produces an object instance of the class `Offering` in the *GoodRelations*[3] ontology. Lines 4 and 6 specifies data properties `hasPrice` (in our own namespace `dd`) and `foaf:page` for this object. At line 7, the nested marker `<gr:includes(dd:House)>` has the double effect of creating an object instance of the class `dd:House` which is related to the current offering object via the specified object property `gr:includes`. In the next lines, this object is in turn related with several data properties. Namespaces are provided by the environment through a namespace resolver as in XPath.

This wrapper produces RDF triples as below, describing two instances, the first one `dd:g31g111` representing a house with 4 bedrooms in Oxford, and the second one `dd:g31g109` representing an offer on this house at GBP 475000.

```
dd:g31g111
      a dd:House ;        dd:bedrooms 4 ;
      gr:name "William Street,  Oxford OX3" ;
      vcard:street-address "William Street,  Marston OX3" ;
      foaf:depiction "http://www.wwagency...99510028.jpg" .
dd:g31g109
      a gr:offering ;     dd:hasPrice "475000"^^xsd:double ;
      gr:includes dd:g31g111 .
```

3. OXPATH USER STORIES

OXPath has been employed in several projects, in which it has proved its versatility. In this section we briefly report on ongoing applications, and discuss OXPath's strengths and weaknesses in the specific context.

DIADEM. DIADEM [4] is a project at Oxford University on unsupervised domain-specific web object extraction. Its goal is to transform unstructured web information into highly structured data without human supervision. DIADEM replaces human annotators in traditional wrapper induction systems by an knowledge about entities (and relations) in the domain, and about the representation of these entities in the textual, structural, and visual language of a website of this domain. With this knowledge at hand, DIADEM needs to analyse only a small fraction of a web site to automatically generate (induce) an OXPath wrapper that is then executed in the second stage to extract all the relevant data on the site. It is worth mentioning that these wrappers are fairly uniform: They usually start from (multi-stage) web forms and pass through result pages to the individual real-estate listings. We are collecting the OXPath wrappers automatically induced in DIADEM for the UK

[3] `http://purl.org/goodrelations/v1`

real-estate domain, available on our project page. ***Lessons learned.*** OXPath is the language of choice for wrappers in DIADEM, as they can be executed at high speed and low cost. However, the experiments carried out so far show that extracting specific parts of a node's text is necessary more often as expected, e.g., to parse a location into components such as street address, town, and postcode. XPath's (and hence OXPath's) lack of support for such extraction is limiting the precision of the obtained wrappers.

DEQA. DEQA [7] is a project at Leipzig University on a comprehensive deep web question answering system for entire domains, that can answer most of natural language questions on objects only available in form of plain, old HTML websites. In DEQA, OXPath plays the critical role of turning websites into structured RDF data. This RDF data is then enriched, e.g., via linking, schema enrichment, geo-coding. E.g., in the UK real estate domain, over 100k triples for 2400 properties were extracted and enriched by over 100k links to the Linked Open Data. This allows users to ask questions in a more natural way (e.g., using landmarks rather than coordinates). Finally, DEQA first tries to convert a natural language query to SPARQL, yet can fall back to standard information retrieval, where this fails. ***Lessons learned.*** OXPath has been positively received in the Linked Open Data community as a valuable tool to quickly produce RDF data. In particular, OXPath proves to be perfectly complemented by recently developed tools for link discovery and enrichment.

ARCOMEM. Arcomem [8] is an European project that aims at developing methods and tools for turning digital archives into community memories. An important part of Arcomem is developed at Telecom ParisTech (France), and focuses on a new application-aware approach to archival Web crawling. The idea is to use a knowledge base of known web applications (e.g., vBullettin, WordPress) and their publishing templates, in order to archive only valuable information (e.g., posts, authors, comments) avoiding duplicates, uninteresting URLs, and presentational templates. This is particularly true for web forums, blogs, and social networks which are the main target of Arcomem. Here, OXPath is employed to **(1)** detecting web application types mainly by patterns rules for specific characteristics of the underlying template, e.g., "powered by wordpress" nodes, **(2)** reaching hidden content by performing actions such as clicking on "show all comments" or "read more" links on posts, and **(3)** extracting the relevant data. ***Lessons learned.*** This project has provided interesting feedback for improvement. Firstly, OXPath's evaluation speed could be drastically increased in all those (frequent) cases where the target pages are almost plain HTML, for which the overhead of a real browser and its rendering can be avoided. Secondly, for evidence preservation, archiving necessitates taking screenshots of web pages which is currently not supported in OXPath.

Complex (Web)Events Processing. At University of Linz, OXPath is used to realize an automatic agent for Ebay, based on event-condition-action rules in a complex event processor (CEP). This agent is capable of determining auctions of interest, monitoring them, and bid if certain conditions, based on rules, are satisfied. Conditions can be complex: for a product auctioned, shopping aggregator websites are queried to determine where to get the product at which price. If the current auction bid is cheaper then the lowest price retrieved the agent places a bid for the product. OXPath has proved to fit well in this context: it is used to detected events on the web (action found, bid placed, bid won, price found elsewhere) and to perform the actions (e.g., bidding) the CEP produces as reactions. ***Lessons learned.*** While CEP are designed to process events at very fast rates of thousands per second, web event recognition introduces a significant latency. Indeed, it is often not practicable to issue web requests at similar pace, which would overtax the target server or lock out the extraction engine.

4. OXPATH: WHERE NEXT?

OXPath is continued to be actively developed at Oxford University together with collaborators from all over Europe. We are working on a number of improvements and extensions to OXPath driven by the user stories from Section 3:

(1) The current public release of OXPath is based on HtmlUnit, a pure and efficient Java browser engine. However, its rendering differs from current browsers and fails for certain heavily scripted cases. Thus, we are currently porting OXPath to Selenium WebDriver, which enables OXPath to access the actual rendering of a page in current browsers. This will also allow to introduce a new action for taking webpage screenshots.

(2) OXPath inherits and improves on XPath's strong node selection support. However, the text matching and processing capabilities are limited to those of XPath and not sufficient for more complex cases, such as contextual or approximate matching in product descriptions. We are considering adopting some of the extensions from XPath Full Text 1.0 [1].

(3) OXPath's output schemata are currently (possibly nested) relations of untyped attributes. We plan to improve the support extracting typed data such as numbers, dates or prices.

(4) Every web page reached by an OXPath expression must be rendered in the existing OXPath engine. As shown in [6], the rendering time dominates all other aspects of OXPath by a wide margin. Though OXPath is already faster than existing extraction engines, it may still be too slow for large-scale archiving or mirroring. To address these scenarios, we are investigating automatic or semi-automatic means of determining when the expression may be evaluated on an unrendered DOM.

All these extensions will be contributed to the open source implementation of OXPath implementation available at `http://oxpath.org`. There we also present an extended list of examples, as well as tutorials on how to use OXPath and how to get involved into developing OXPath further.

5. REFERENCES

[1] S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, D. McBeath, M. Rys, and J. Shanmugasundaram. XQuery 1.0 and XPath 2.0 Full-Text. Recommendation, W3C, 2011.

[2] Arcomem EU project. `http://www.arcomem.eu/`, 2013.

[3] R. Baumgartner, S. Flesca, and G. Gottlob. Visual Web Information Extraction with Lixto. In *VLDB*, 2001.

[4] T. Furche, G. Gottlob, G. Grasso, O. Gunes, X. Guo, A. Kravchenko, G. Orsi, C. Schallhart, A. Sellers, and C. Wang. DIADEM: Domain-centric, Intelligent, Automated Data Extraction Methodology. In *WWW*, 2012.

[5] T. Furche, G. Gottlob, G. Grasso, C. Schallhart, and A. Sellers. OXPath: A Language for Scalable, Memory-efficient Data Extraction from Web Applications. *PVLDB*, 2011.

[6] T. Furche, G. Gottlob, G. Grasso, C. Schallhart, and A. Sellers. OXPath: A Language for Scalable Data Extraction, Automation, and Crawling on the Deep Web. *VLDB J.*, 2013.

[7] J. Lehmann, T. Furche, G. Grasso, A.-C. N. Ngomo, C. Schallhart, A. Sellers, C. Unger, L. Bühmann, D. Gerber, K. Höffner, D. Liu, and S. Auer. DEQA: Deep Web Extraction for Question Answering. In *ISWC'12*, LNCS, 2012.

[8] Mozenda. `http://www.mozenda.com/`, 2013.