Exploring the Web with OXPath[*]

Tim Furche, Georg Gottlob, Giovanni Grasso, Christian Schallhart, Andrew Sellers
Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD
firstname.lastname@comlab.ox.ac.uk

ABSTRACT

OXPath is a careful extension of XPath that facilitates data extraction from the deep web. It is designed to facilitate the large-scale extraction of data from sophisticated modern web interfaces with client-side scripting and asynchronous server communication. Its main characteristics are (1) a minimal extension of XPath to allow page navigation and action execution, (2) a set-theoretic formal semantics for full OXPath, (3) and a sophisticated memory management that minimizes page buffering. In this poster, we briefly review the main features of the language and discuss ongoing and future work.

Categories and Subject Descriptors

H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia—*navigation*

General Terms

Languages, Algorithms

Keywords

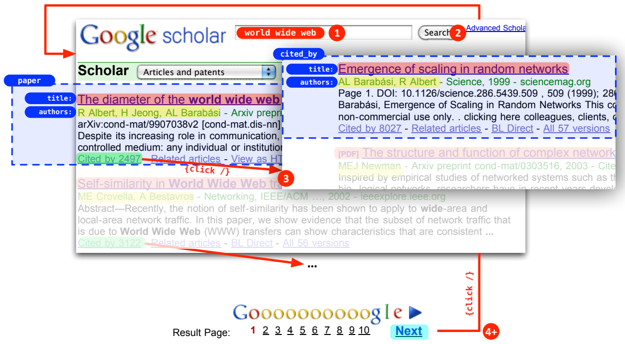
Web extraction, web automation, XPath, AJAX

1. INTRODUCTION

In this paper, we present details of the OXPath language, a formalism for specifying data extraction from modern web applications.

XPath has become the *de facto* standard for querying single XML and HTML documents. However, modern web applications often require navigation, form filling, and other (single or multi-page) interactions to expose the data of interest. OXPath is an extension of XPath to allow page navigation, form filling, interactions, and the extraction of multiple data items in a single expression.

For space reasons, we focus here on presenting the language and point the interested reader to [1] for discussion of OXPath's semantics, evaluation algorithm, and formal properties.



Google scholar
Search
Advanced Schola
Scholar | Articles and patents

The diameter of the world wide web
R Albert, H Jeong, AL Barabási - Arxiv prepri
arXiv:cond-mat/9907038v2 [cond-mat.dis-nn]
Despite its increasing role in communication,
controlled medium: any individual or institutio
Cited by 2497 - Related articles - View as HT

Emergence of scaling in random networks
AL Barabási, R Albert - Science, 1999 - sciencemag.org
Page 1. DOI: 10.1126/science.286.5439.509 , 509 (1999); 28
Barabási, Emergence of Scaling in Random Networks This co
non-commercial use only. . clicking here colleagues, clients,
Cited by 8027 - Related articles - BL Direct - All 57 versions

[PDF] The structure and function of complex network
MEJ Newman - Arxiv preprint cond-mat/0303516, 2003 - Cite
Inspired by empirical studies of networked systems such as th

Self-similarity in World Wide Web tr
ME Crovella, A Bestavros - Networking, IEEE/ACM …, 2002 - ieeexplore.ieee.org
Abstract—Recently, the notion of self-similarity has been shown to apply to **wide**-area and local-area network traffic. In this paper, we show evidence that the subset of network traffic that is due to **World Wide Web** (WWW) transfers can show characteristics that are consistent ...
Cited by 3122 - Related articles - BL Direct - All 56 versions

Gooooooooooogle ▶
Result Page: 1 2 3 4 5 6 7 8 9 10 Next

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

Figure 1: Finding an OXPath through Google Scholar

2. EXAMPLE

Figure 1 shows an OXPath expression for extraction of papers, authors, and citations from Google Scholar: Lines 1–2 fill and submit the search form. Line 3 realizes the iteration over the set of result pages by repeatedly clicking the "Next" link. Lines 4–5 identify a result record and its author and title, lines 6–8 navigate to the cited-by page and extract the papers.

3. OXPATH: THE LANGUAGE

OXPath extends XPath with (1) a new axis for selecting nodes based on visual attributes, (2) a new node-test for selecting visible fields, (3) a new kind of location step for actions and form filling, and (4) a new kind of predicate for marking data to be extracted. For page navigation, we adapt the notion of Kleene star over path expressions from [2]. Nodes and values marked by extraction markers

[*] 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. The views expressed in this article are solely those of the authors.

LWDM 2011, March 25, 2011, Uppsala, Sweden.
Copyright 2011 ACM XXXX ...$10.00

are streamed out as records of the result tables. For efficient processing, we cannot fix an apriori order on nodes from different pages. Therefore, we do not allow access to the order of nodes in sets that contain nodes from multiple pages.XPath expressions are also OXPath expressions and retain their same semantics, computing sets of nodes, integers, strings or Booleans.

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 is similar to the **`attribute`** axis, but navigates dynamic CSS properties instead of static HTML properties. For example, the following expression selects the sources for the top story on Google News based on visual information only:

```
doc("news.google.com")//*[style::color="#767676"]
```

The **`style`** axis provides access to the actual CSS properties (as returned by the DOM `style` object), rather than only to inline styles.

An essential application of the **`style`** axis is the navigation of *visible fields*. This excludes fields which have type or visibility `hidden`, or have display property `none` set for either themselves or an ancestor. To ease field navigation, we introduce the node-test `field()` as an abbreviation. In the above Google search for “Oxford”, we rely on the order of the visible fields selected with `descendant::field()[1]` and `following::field()[1]`. Such an expression is not only easier to write, it is also far more robust against changes on the web site. For it to fail, either the order or set of visible form fields has to change.

Actions. For explicitly simulating user actions, such as clicks or mouse-overs, OXPath introduces *contextual action steps*, as in **`{click}`**, and *absolute action steps* with a trailing slash, as in **`{click /}`** . Since actions may modify or replace the entire DOM, OXPath’s semantics assumes that they produce a new DOM. Absolute actions return DOM roots, while contextual actions return those nodes in the resulting DOMs which are matched by the action-free prefix of the performed action: The *action-free prefix* AFP(*action*) of *action* is constructed by removing all intermediate contextual actions and extraction markers from the segment starting at the previous absolute action. Thus, the action-free prefix selects nodes on the new page, if there are any. For instance, the following expression enters “Oxford” into Google’s search form using a contextual action—thereby maintaining the position on the page—and clicks its search button using an absolute action.

```
doc("google.com")/descendant::field()[1]/{"Oxford"}
                 /following::field()[1]/{click /}
```

Extraction Marker. Navigation and form filling are often means to data extraction: While data extraction requires records with many related attributes, XPath only computes a single node set. Hence, we introduce a new kind of qualifier, the *extraction marker*, to identify nodes as representatives for records and to form attributes from extracted data. For example, `:<story>` identifies the context nodes as `story` records. To select the text of a node as title, we use `:<title=string(.)>`. Therefore,

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

extracts from Google News a `story` element for each current story, containing its title and its sources, as in:

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

The nesting in the result above mirrors the structure of the OXPath expression: An extraction marker in a predicate represents an attribute to the (last) extraction marker outside the predicate.

Kleene Star. Finally, we add the Kleene star, as in [2], to OXPath. For example, we use the following expression to query Google for “Oxford”, traverse all accessible result pages, and to extract all contained links.

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

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

Also note that OXPath adopts the class selector from CSS.

4. FUTURE WORK

To the best of our knowledge OXPath is the first web extraction system with strict memory guarantees. These memory guarantees reflect strongly in our experimental evaluation. Just as important, OXPath is built on standard web technology, such as XPath and DOM, so that it is familiar and easy to learn for web developers. We believe that it has the potential to become an important part of the toolset of developers interacting with the web. To further simply OXPath expressions and enhance their robustness, we plan to investigate additional features, such as more expressive visual language constructs and multi-property axes.

A strength of OXPath is that it is focused and easily embeddable. We want to exploit that potential by realizing OXPath in a variety of contexts, but will next focus on deploying OXPath in a cloud for large-scale extraction.

OXPath is designed for highly parallel execution: the host language can assign different bindings for the same variable to create multiple OXPath queries. These queries can be processed with separate web sessions hosted on separate computing instances. We think this approach to parallel decomposition is ideally suited for the share-nothing nature of computing instances in elastic computing environments. The design of a host language for the cloud requires careful consideration: in particular, most existing web programming languages, such as XQuery, do not provide access to a dynamic DOM. Beyond variable bindings, any useful host language almost certainly requires aggregation and subquerying capabilities.

Adapting OXPath to a cloud-based environment raises the question of how to optimize the evaluation of several parallel OXPath expressions in order to minimize expensive browser instantiations, unnecessary replication of common action sequences, and to best consolidate output extracted by OXPath expressions running in multiple instances.

5. REFERENCES

[1] T. Furche, G. Gottlob, G. Grasso, C. Schallhart, and A. Sellers. Finding an OXPath to Cherries Hidden in the Scripted Web. Tech. rep., `diadem-project.info/oxpath`.

[2] M. Marx. Conditional XPath. *TODS*, 2005.