Visual OXPath: Robust Wrapping by Example*

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

ABSTRACT

Good examples are hard to find, particularly in wrapper induction: Picking even one wrong example can spell disaster by yielding overgeneralized or overspecialized wrappers. Such wrappers extract data with low precision or recall, unless adjusted by human experts at significant cost.

Visual OXPath is an open-source, visual wrapper induction system that requires minimal examples and eases wrapper refinement: Often it derives the intended wrapper from a single example through sophisticated heuristics that determine the best set of similar examples. To ease wrapper refinement, it offers a list of wrappers ranked by example similarity and robustness. Visual OXPath offers extensive visual feedback for this refinement which can be performed without any knowledge of the underlying wrapper language. Where further refinement by a human wrapper is needed, Visual OXPath profits from being based on OXPath, a declarative wrapper language that extends XPath with a thin layer of features necessary for extraction and page navigation.

Categories and Subject Descriptors

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

General Terms

Languages, Algorithms

Keywords

Web extraction, Web automation, XPath, AJAX

1. INTRODUCTION

The web has enabled choice on a scale never seen before: We are no longer limited to the products available in local stores, but can choose from products and offers from shops all over the world. However, the "paradox of choice" is that this has actually made it harder for us to make a satisfying choice—never knowing if further searching would have yielded a better offer. A first step towards addressing this paradox is to provide users with a comprehensive, detailed view of the available choices that can be automatically processed further, e.g., to match with user preferences.

Decision support is just one example, where web pages need to be turned into actionable data fit for automatic processing. This *data extraction* process is driven by examples: Human annotators provide examples of data to be extracted, which are used to generate extraction programs (or *wrappers*). Most existing wrapper induction systems, however, face two severe limitations: They need many examples for accurate data extraction and couple wrapper induction and wrapper language tightly. The former significantly increases human effort and the likelihood for errors in the examples. The latter has hampered progress in data extraction, as wrapper languages are created ad hoc as part of wrapper induction systems and often not amenable to further refinement by a human user.

OXPath [1] is a novel, lightweight wrapper language that aims to provide a broad base for wrapper induction systems. It extends XPath with four features necessary for wrappers: navigation between pages through *actions*, *iteration* over many similar pages, e.g., to navigate paginated results, *extraction* of multiple related items, and access to *visual* features of a website. Though OXPath has already been adopted for easy manual development of wrappers, there are, so far, no wrapper induction systems for OXPath.

To that end, we present **Visual OXPath**, a supervised wrapper induction system for OXPath that generates highly robust wrappers from few examples. For most wrapper tasks, the user can define a wrapper in Visual OXPath visually without any knowledge of OXPath: Given only a *single example*, Visual OXPath suggests a list of wrappers among which the user can choose based on visual feedback on what will be extracted. The list is ranked by similarity, coverage, and robustness of the generated wrapper. **(1)** Similarity allows us to extend the single example to all other DOM nodes that are somewhat similar with the given example. **(2)** Coverage biases towards wrappers that extract more examples. **(3)** Robustness promotes wrappers that we judge likely to be robust to minor changes in the layout or structure of the web page. In a user study, we show that Visual OXPath suggests the most desirable wrapper in over 90% of the cases. In nearly all remaining cases, the most desirable wrapper is contained in the list of suggested wrappers, but not top ranked. We also show that over an evaluation period of 6 months high robustness ranking clearly correlates with wrapper robustness.

In this demonstration in particular, we demonstrate Visual OXPath on a set of prepared *examples* as well as pages chosen by the audience. We guide the user through the definition of a wrapper for extraction flights from Skyscanner, as well as publication on Google Scholar that cite one of her papers, see Section 5 for details.

*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.

We demonstrate, how Visual OXPath gives the user full *freedom* in which order to provide examples: For example, to extract a property with its price and location, we could start with one location and price, from which Visual OXPath automatically derives similar locations and prices on the page, as well as which locations and prices are part of the same property. We also show, how to deal with the few cases, where Visual OXPath does not find the desired wrapper, but offers the option to edit the generated expressions. Such edits are almost always in purely XPath parts of an expression, as there is little room for error in the OXPath specific parts.

2. OXPATH IN A NUTSHELL

OXPath is a superset of XPath, which is extended for declarative specification of interactions with web applications for data extraction. To this end, OXPath introduces **(1)** the *action location step* for simulating user interaction such as mouse events, form filling; **(2)** the *style axis* for selecting nodes and fields based on actual visual attributes as rendered by the browser; **(3)** the *extraction marker predicate*, for marking data to be extracted, and **(4)** the *Kleene star operator*, for iterating expressions.

Actions such as clicks or mouse-overs can be explicitly executed in OXPath on a set of DOM nodes. To enter "OXPath" into Google Scholar's search form and click the search button:

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

OXPath allows two types of action steps, namely *contextual actions steps*, such as `{click}` or `{OXPath}`, and *absolute actions steps* with a trailing slash, as in `{click /}`. An absolute action step returns the DOM root of the page after action execution, while contextual actions continue from the same context as the action, if possible.

The Style Axis allows selecting elements based on either CSS properties, e.g., only the visible fields (denoted as `field()`). The `style` axis uses the computed CSS properties and can not be expressed in XPath. To select all paper links on Google Scholar:

```
//a[style::color="blue"][style::font-size="16px"]
```

Extraction markers are used in OXPath to allow the extraction of many related data items and of nested data items. In contrast, XPath returns a single node set. To extract from Google Scholar each paper with its title and authors:

```
..//div[@class='gs_r']:<publication>[.//h3:<title=string(.)>]
                       [.//span[@class='gs_a']:<authors=string(.)>]
```

OXPath supports many output formats, as XML this produces:

```
<publication><title>OXPath: A Language for ...</title>
    <authors>Tim Furche, ...</authors> ... </publication>
```

Kleene stars are borrowed from Regular XPath [2] to repeat an expression. By including an action that triggers a page change into the expression, OXPath can navigate over an unbounded number of pages. To traverse all result pages on Google scholar and extract the publication titles:

```
.../(//a[contains(string(.),'Next')]/{click/})*//h3:<title>
```

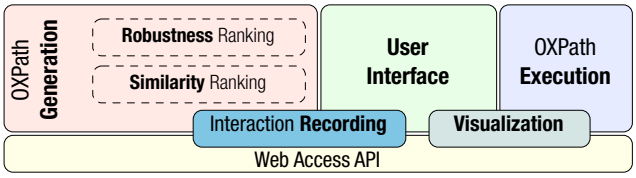

Figure 1: Architecture

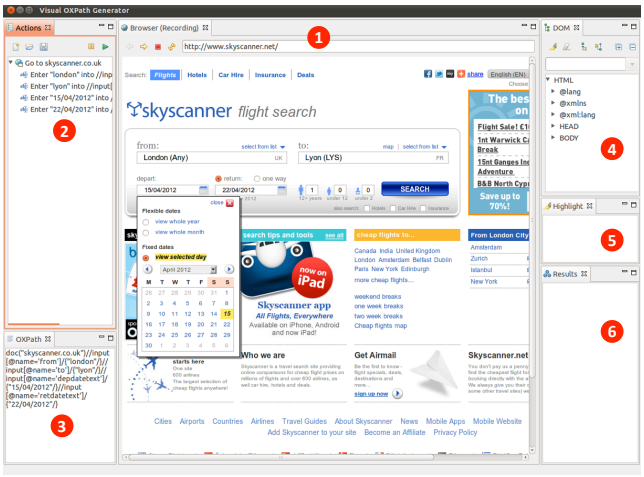

Figure 2: Visual OXPath UI

3. VISUAL OXPATH

Visual OXPath combines frameworks for similarity, robustness and visualization with browser interaction recording and OXPath expression generation. Figure 1 shows the architecture of Visual OXPath. The front-end is based on the Eclipse Rich Client Platform framework and is shown in Figure 2. It is build around a life browser **(1)** where the user can interact with web pages (recorded by the system) and pick examples of data to be extracted. The browser is surrounded by views for

(2) automatically recorded actions that can be refined or grouped into Kleene stars for repetition at any time. Here we also display extraction markers, subtly distinguished through different icons, to avoid confronting the user with two dependent lists.
(3) the generated OXPath expression (updated live).
(4) the DOM structure of the page for finding and highlighting specific elements, useful when manually refining expressions.
(5) the list of currently highlighted elements.
(6) the list of records that will be extracted from this page.

The user interface interacts with the three other primary components, the *web access API* for controlling the embedded browser; the *OXPath execution* engine; and the *OXPath generation* framework. In the OXPath generation framework we generate OXPath wrappers from user provided examples. To identify, e.g., the list of prices on a result page from a single example, Visual OXPath needs to find a set of possible XPath expressions that describe a suitable list of DOM nodes containing the example. "Suitable" is defined by the similarity and robustness ranking: In the similarity phase, we adapt tree pattern generalization techniques (see, e.g., [3]). These involve manipulating or removing predicates, deleting node tests, relaxing the child axis or the deletion of entire steps. All heuristics try to optimize similarity while also maximizing coverage. This ranking is refined by the *robustness ranking* where we identify alternative, but more robust means to address the same nodes. Visual OXPath prefers, e.g., nodes identifiable only through a unique attribute over those using lengthy location paths with positional se-

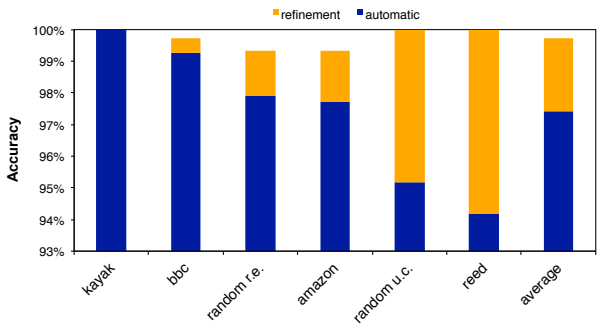

Figure 3: User Study on Expression Similarity

lection. Robustness is defined recursively, i.e., we also prefer nodes that are in a simple relative position to a robust anchor node [4].

4. USER STUDY

To demonstrate the effectiveness of the similarity and robustness heuristics, we have conducted a user with a group of 9 novice and experience users, each extracting 4 fields, from a list of six website, four of them fixed and two randomly chosen by each user in the real estate and used car domains. As shown in Figure 3, the similarity ranking captures user intent correctly in most cases: Of all 216 OXPath expressions suggested by Visual OXPath, 91% of the cases yield perfect accuracy. Overall, we obtain 97.4% average accuracy (depicted to the last bar on the right side) with a minimum of 94%, where accuracy is the F_1-score on the extracted vs. desired data items. For the remaining 9%, users needed to refine the expression, but mostly by selecting another generated OXPath expression. This increases the average accuracy to 99.7% and the overall minimum to 99.3%. There is no significant difference in expression quality between novice and experienced users. All users reported that the tool is easy or very easy to use. Even users without prior exposure to OXPath can create a wrapper in less than 3 minutes on average.

Robustness is automatically evaluated over a 6 month time-frame: Figure 3 shows the accuracy drop over time. The canonical XPath (in red), i.e., a unique path of child steps with position, fails quickly. The wrapper ranked best by Visual OXPath (in green), however, remains at 100% for nearly the entire period and drops only slightly at the end. None of the lower ranked expressions performs as well, their average shown in the blue line.

5. DEMO DESCRIPTION

In the demonstration we focus on highlighting the main contributions of Visual OXPath: **(1)** wrapper induction from a single example through *similarity*; **(2)** *ease of use* of Visual OXPath

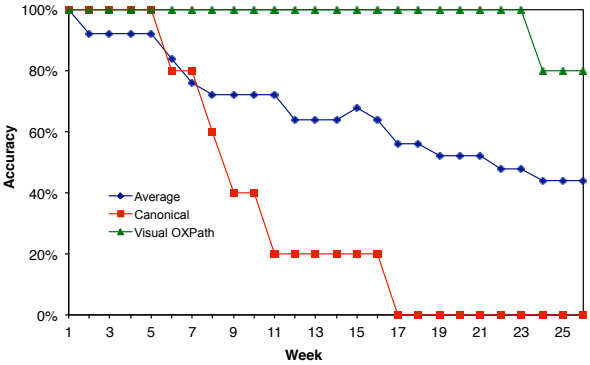

Figure 4: User Study on Expression Robustness

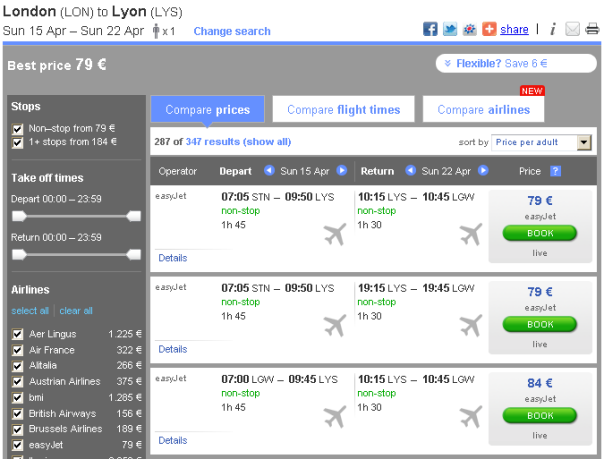

Figure 5: Skyscanner Result Page

through extensive visual feedback and ranked wrapper suggestions; **(3)** *freedom* in which order to provide examples for multiple entities; **(4)** ability to deal with *complex scripted pages*. We also discuss the robustness of the generated wrappers and how they can be evaluated using OXPath in a cloud environment.

A typical use of Visual OXPath comprises the navigation to a webpage, usually by filling form fields with values, and the extraction of data from one or multiple result pages. Users simply navigate through the web and mark the data they want to extract directly on the page. The tool records the user interaction with all visited pages and generates an OXPath expression. Elements selected by this expression are highlighted on the webpage, which allows users to verify if the tool interpreted their intent correctly. If undesired elements are selected, users are visually guided to refine the expression.

Specifically, we demonstrate Visual OXPath along a set of prepared examples as well as with suggestions from the audience. To give an impression of the demonstration assume *you want to fly from London to Lyon*. What would be the best flight to take? Flight price are quickly fluctuating, even on daily basis, so that you might want to constantly monitoring your preferred travel web sites, to catch the best offer. Here, we focus on Skyscanner(`skyscanner.com`) as one of these sites.

Figure 2 depicts the homepage with the search criteria as entered by the user loaded into Visual OXPath. Visual OXPath automatically generates the following expression from the user interaction:

```
doc("skyscanner.com")//input[@name='from']/{"london"/}
                     //input[@name='to']/{"lyon"/}
               //input[@name='depdatetext']/{"15/04/2012"/}
               //input[@name='retdatetext']/{"22/04/2012"/}
  //button#sc_search/{click/}
```

On result pages such as the one shown in Figure 5, users specify which elements should be extracted. In contrast to other web data extraction tools [5, 6], this happens on the rendered webpage itself rather than in the HTML parse tree or source code. Moving the mouse over the webpage highlights the currently hovered element with a grey overlay. To extract a list of results, users hover the mouse over any result and select “Extract Item” in the context menu. Figure 6 shows this step for extracting the price. It also shows on the right hand the extracted data for a partial wrapper, highlighting matched elements on the web page in the same color.

When the user selects “Extract Item” Visual OXPath finds similar nodes on the page and a list of suitable OXPath expression to the user, ranked by similarity, coverage, and robustness. For the given

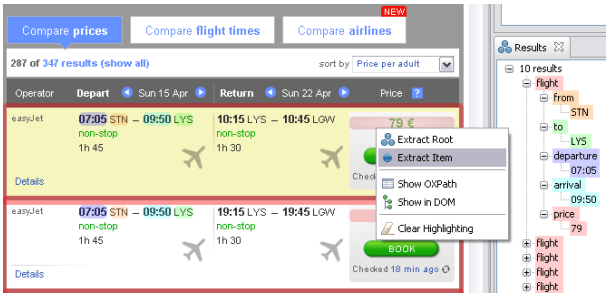

Figure 6: Selecting Data for Extraction

example, the user only needs to indicate just one example for each flight attribute. The order in which those attributes are chosen is up to the user and it is not necessary to first (or at all) define the boundaries as a single flight, as Visual OXPath can identify those from the repeated structure on the page. In the given example, the `div` representing a flight is a suitable common ancestor of the attributes and thus Visual OXPath generates the following expression:

```
//div.sorted-px/div:<flight>
  [.//p.ileg-o/span[2]:<from=string(.)>]
  [.//p.ileg-o/span[5]:<to=string(.)>]
  [.//p.ileg-o/span[1]:<departure=string(.)>]
  [.//p.ileg-o/span[4]:<arrival=string(.)>]
  [.//span.EUR:<price=substring-before(.,'
      {\color{darkorange}\euro}')>]
```

The extraction markers, such as "price" define the output records. They are derived from the DOM tree, for instance from the class of the extracted element. In summary, one simple click per extracted data item—origin and destination airport, departure and arrival time, and the price—suffices to specify the complete OXPath expression.

Refinement. For easy refinement, Visual OXPath lists all recorded actions, as depicted in the left hand-side on Figure 7. Existing actions can be moved, deleted, modified and highlighted on the web page, via a contextual menu. In addition, users can create new actions, such as a Kleene star, which repeats other actions with a specified cardinality. For instance, multiple result pages can be crawled by clicking on the "Next" button and surrounding the recorded action with a Kleene star. The expression

```
((//button.next)[1]/{click/})*
```

is automatically generated.

For Skyscanner, we are done after this step, but we will also show pages such as `autotrader.co.uk`, where Visual OXPath's heuristics fail to suggest the best wrapper due to a high level of noise.

In such cases, selecting "Properties" in the context menu opens the dialog shown in Figure 7 (right hand-side), which allows users to refine every step of the OXPath expression. This includes the extraction marker and function. By default, the function `string(.)` extracts the text content of the subtree below the addressed node. Other options in the dropdown menu include the HTML code, any attribute of the node, such as `src` for `img` elements, or a user defined OXPath function. Furthermore, the Visual OXPath recognizes substrings common to the whole result set and suggests to adjust the extraction function. In the given example, all prices end with the € character, so that the tool suggests a `substring-before` function.

The last field in the dialog contains the OXPath to address the node in the DOM tree. Multiple OXPath expressions are generated, scored according to their resilience to future page changes, and ranked with the most robust expression pre-selected.

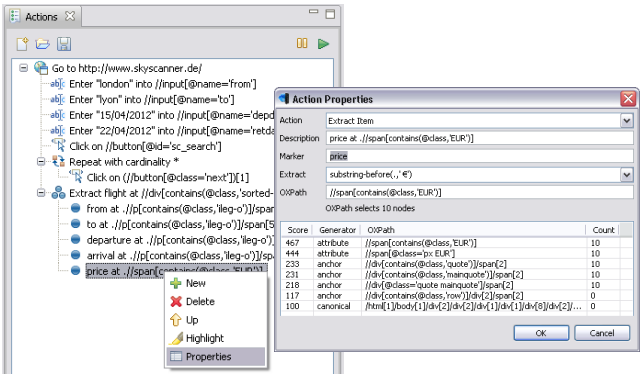

Figure 7: List of Recorded Actions and Extractions

For the given example, the expression `//span[@class='px EUR']` is considered more robust than `//div[@class='quote mainquote']/span[2]` because it addresses the node directly through an attribute instead of using its parent as anchor.

Users can select another expression in the list that appears to be more suitable. They can also adapt the generated OXPath or define their own expression manually. For assistance, the syntactic correctness of the expression and the number of nodes it selects is updated during typing. In addition, Visual OXPath highlights the selected items on the webpage and displays the extracted content each time the user selects or adjusts an expression.

For advanced users, the tool offers DOM tree navigation and manipulation. The context menu of the embedded browser allows any clicked element to be shown in the DOM tree. Any attribute value can be edited and the page rendering is updated instantly. Furthermore, arbitrary OXPath expressions can be evaluated and highlighted on the webpage. This helps users to understand the webpage intricacies and assess the applicability of manually created OXPaths expressions.

Finally, the OXPath expression can be replayed with the extracted data being highlighted on the webpage for verification. The extraction project can be saved to disk for future refinement and saved as OXPath expression for execution with the standard OXPath engine.

A screencast of Visual OXPath is available at `diadem-project.info/oxpath/visual`.

6. REFERENCES

[1] T. Furche, G. Gottlob, G. Grasso, C. Schallhart, and A. Sellers, "OXPath: A language for scalable, memory-efficient data extraction from web applications," *Proceedings of the VLDB Endowment*, 2011.

[2] M. Marx, "Conditional XPath," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 4, pp. 929–959, 2005.

[3] B. Fazzinga, S. Flesca, and A. Tagarelli, "Schema-based web wrapping," *Knowledge and Information Systems*, pp. 1–47, 2011.

[4] M. Abe and M. Hori, "Robust pointing by XPath language: Authoring support and empirical evaluation," in *IEEE Symposium on Applications and the Internet*, 2003, pp. 156–165.

[5] A. Laender, B. Ribeiro-Neto, A. da Silva, and J. Teixeira, "A brief survey of web data extraction tools," *ACM SIGMOD Record*, vol. 31, no. 2, pp. 84–93, 2002.

[6] C. Chang, M. Kayed, M. Girgis, and K. Shaalan, "A survey of web information extraction systems," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1411–1428, 2006.