SXPath - Extending XPath towards Spatial Querying on Web Documents

Ermelinda Oro
Department of Electronics, Informatics, and Systems
University of Calabria
Altilia s.r.l.
Via. P. Bucci, 41/C, 87036, Rende CS, Italy
linda.oro@deis.unical.it

Massimo Ruffolo
Institute of High Performance Computing and Networking of the Italian CNR
Altilia s.r.l.
Via. P. Bucci, 41/C, 87036, Rende CS, Italy
ruffolo@icar.cnr.it

Steffen Staab
Institute for Computer Science
Institute WeST
University of Koblenz
Universitätsstraße 1, PO Box 201 602 56016
Koblenz, Germany
staab@uni-koblenz.de

ABSTRACT

Querying data from presentation formats like HTML, for purposes such as information extraction, requires the consideration of tree structures as well as the consideration of spatial relationships between laid out elements. The underlying rationale is that frequently the rendering of tree structures is very involved and undergoing more frequent updates than the resulting layout structure. Therefore, in this paper, we present Spatial XPath (SXPath), an extension of XPath 1.0 that allows for inclusion of spatial navigation primitives into the language resulting in conceptually simpler queries on Web documents. The SXPath language is based on a combination of a spatial algebra with formal descriptions of XPath navigation, and maintains polynomial time combined complexity. Practical experiments demonstrate the usability of SXPath.

1. INTRODUCTION

HTML is a presentation-oriented document format conceived for presenting Web documents on screen. Querying data from such a format requires a versatile machinery to abstract from presentation structure into conceptual relationships. Germane to such abstraction is the transformation into structural languages such as XML. Indeed quite often XML languages are used as presentation formats, e.g. HTML 5. Based on these structural languages, one may use well founded and known query formalisms such as XPath 1.0 [25] and XQuery 1.0 [24] in order to turn presentation structure into meaningful relational or logical representations. Typical problems are incurred by the separation of document structure and the ensued spatial layout — whereby the layout often indicates the semantics of data items, e.g. the meaning of a table cell entry is most easily defined by the leftmost cell of the same row and the topmost cell of the same column (in Western languages) [9]. In tree structures such as those arising on real-world Web pages, such spatial arrangements are rarely explicit and frequently hidden in complex nestings of layout elements — corresponding to intricate tree structures that are conceptually difficult to query.

In the literature, approaches aimed at manipulating Web pages by leveraging the visual arrangement of page contents [13, 22], and frameworks for representing and querying multimedia and presentation databases [2, 14] have been proposed. However such approaches and frameworks provide limited capabilities in navigating and querying Web documents for information extraction purposes. Therefore, we propose to extend XPath 1.0 by new spatial navigation primitives, namely: (i) *Spatial Axes*, based on topological [21] and rectangular cardinal relations [18], that allow for selecting document nodes that have a specific spatial relation w.r.t. the context node. (ii) *Spatial Position Functions* that exploit some *spatial orderings* among document nodes, and allow for selecting nodes that are in a given spatial position w.r.t. the context node.

The main contributions of this paper are the following: (i) We have defined the novel *Spatial Document Object Model* (SDOM) that constitutes a mixed spatial and structural model of HTML pages. In our model, we represent the relationships between DOM nodes resulting from the XML tree structures on the one hand and from the spatial relationships between the rendered XML elements on the other hand. (ii) We have defined and implemented the *Spatial XPath* (SXPath) language that allows for navigating the SDOM. It can be used on its own, but also as building block for more expressive languages such as XQuery. (iii) We have defined two fragments of SXPath language, namely: *Core SXPath* that represents the navigational core of SXPath, and *Spatial Wadler Fragment* (SWF) that is a fragment with very interesting practical value, since the vast majority of useful queries fall into it, and that allows for optimized evaluation. (iv) We have defined the formal semantics of the language and the evaluation algorithms that allow for maintaining combined polynomial time complexity [12]. (v) We have performed several empirical evaluations showing that (a) processing time needed is polynomial as predicted by worst case complexity, and that the SXPath language is (b) substantially independent from the adopted browser and visualization parameters variations, (c) easy to learn and easier to use than pure XPath on Web pages, (d) more tolerant to modifications of the internal structure of Web pages, and (e) capable to provide benefit to some current Web contents manipulation and wrapper learning approaches.

The paper is organized as follows. In Sec. 2 we provide motivations that drove us to extend XPath 1.0 with spatial navigation primitives. In Sec. 3 we describe SXPath data model, formal syntax, semantics and complexity issues. Sec. 4 reports notes on the implementation and results of experiments aimed at evaluating processing performances, SXPath usability, and qualitative enhancement in real applications. Sec. 5 briefly describes related works.

2. WHY SPATIAL XPATH?

Web designers plan Web pages contents in order to provide visual patterns that help human readers to make sense of document contents. This aspect is particularly evident in *Deep Web* pages [16], where designers always arrange data records and data items with visual regularity to meet the reading habits of humans.

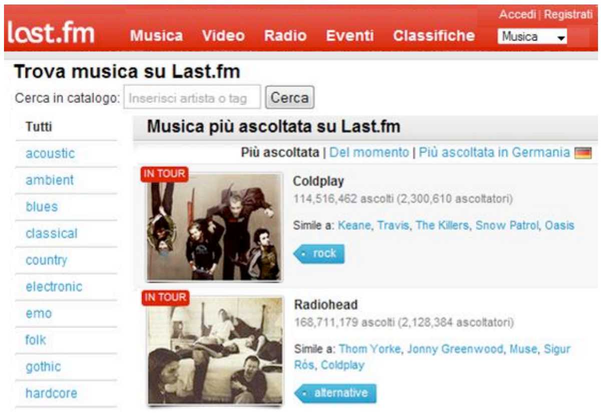

Accedi | Registrati
last.fm
Musica
Video
Radio
Eventi
Classifiche
Musica
Trova musica su Last.fm
Cerca in catalogo:
Inserisci artista o tag
Cerca
Tutti
acoustic
ambient
blues
classical
country
electronic
emo
folk
gothic
hardcore
Musica più ascoltata su Last.fm
Più ascoltata | Del momento | Più ascoltata in Germania
IN TOUR
Coldplay
114,516,462 ascolti (2,300,610 ascoltatori)
Simile a: Keane, Travis, The Killers, Snow Patrol, Oasis
rock
IN TOUR
Radiohead
168,711,179 ascolti (2,128,384 ascoltatori)
Simile a: Thom Yorke, Jonny Greenwood, Muse, Sigur Rós, Coldplay
alternative

Figure 1: A Page of the http://www.lastfm.it/ Web Site

In Figure 1 we show a deep Web page coming from the last.fm social network. For instance, information about the two bands 'Coldplay' and 'Radiohead' in Figure 1 is given using similar layout. In the past, manual wrapper construction (e.g. [22]), or wrapper induction approaches (e.g. [7, 19, 27]) have exploited regularities in the underlying document structures, which led to such similar layout, to translate such information into relational or logical structures. However, surveying a large number of real (Deep) Web pages, we have observed that the document structure of current Web pages has become more complicted than ever implying a large conceptual gap between document structure and layout structure. Thus, it has become very difficult: (i) for human and applications aiming at manipulating Web contents (e.g. [10, 13, 22]), to query the Web by language such as XPath 1.0; (ii) for existing wrapper induction approaches (e.g. [19, 27]) to infer the regularity of the structure of deep Web pages by only analyzing the tag structure. Hence, the effectiveness of manual and automated wrapper construction are limited by the requirement to analyse HTML documents with increasing structural complexity. The SXPath language allows for navigating DOM structures like XPath 1.0 as well as for exploiting the spatial layout of DOM nodes after Web pages rendering. In the following examples we give an intuition about novel features and capabilities of the SXPath language.

Layout engines of Web browsers consider the area of the screen aimed at visualizing a Web page, as a 2-dimensional Cartesian plane. They adopt rendering rules that take into account the page DOM structure and the associated *cascade style sheets* (CSS). In the rendered page each DOM node is visualized in a rectangle having sides parallel to the axes of the Cartesian plane. Figure 2 depicts rectangles created by a layout engine for the page in Figure 1. Some rectangles are annotated by fragments of XPath 1.0 location paths that characterize related DOM nodes.

A human reader can relate information on the page in Figure 1 by considering the spatial arrangement of laid out content elements. He can interpret the spatial proximity of images and nearby strings as a corresponding aggregation of information, namely as the complete record describing the details of a music band profile and one of its photos. Figure 3 depicts a sketch of the DOM related to the part of the document containing the profile of the Radiohead music band. Grey circles and boxes are used for the nodes that are visualized on screen, whereas white circles and boxes depict nodes that belong to the DOM but are not visible on screen.

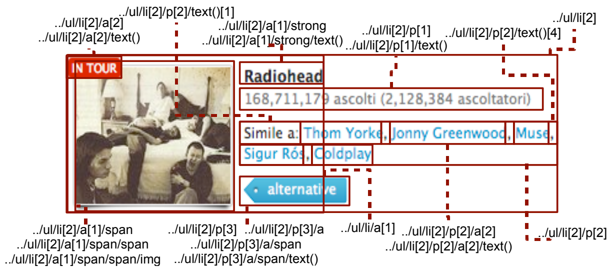


Figure 2: Rectangles that Bound Visualized DOM Nodes

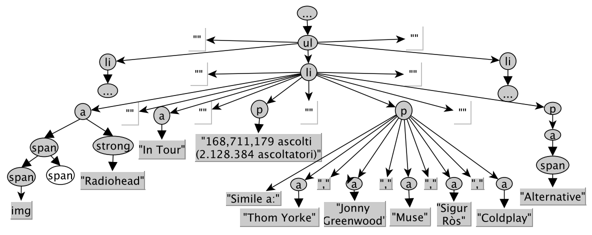


Figure 3: A DOM Portion of the Web Page in Figure 1

The Web page in Figure 1 allows us to show how to extract data about music bands by a corresponding conventional query using existing XQuery 1.0 [24] and XPath 1.0 on the DOM partially depicted in Figure 3.

Example 1. XQuery 1.0 and XPath 1.0

```
for $li in document ("last-fm.htm")
(1.1) //div[@id='content']//ul/li   return
    <music-band>
(1.2)   <name> {$li/a/strong/text()} </name>
        <similar-bands>
(1.3)          {$li/p[2]//text()}
        </similar-bands>
    </music-band>
```

For writing the query in Example 1 above, a human user must know the intricate DOM structure of the input Web page (a sketch of which is given in Figure 3). The location path (1.1) is the shortest relative path found by performing many attempts. In fact, shorter location paths like `//li` or `//div//ul/li` (that does not use attributes) make the query unsound. In contrast, the following query exploits SXPath that allows for extracting details of music bands by exploiting only the DOM nodes of type `img` and `text`, and their spatial relationships.

Example 2. XQuery 1.0 and SXPath

```
for $img in document ("last-fm.htm")
(2.1) /CD::img[N|S::img]   return
    <music-band>
(2.2)   <name> {$img/E::text[W,1][N,1]} </name>
        <similar-bands>
(2.3)          {$img/E::*[W,1][N,3][max]/CD::text}
        </similar-bands>
    </music-band>
```

The *spatial location path* (`2.1`) returns images that form a vertical sequence. These are exactly the photos of music bands in the page. Such kind of visual patterns (i.e. alignment in a given direction) are very frequent in Deep Web pages, but often hard to recognize in the DOM structure. The spatial location path `$img/E::text` in (`2.2`) returns all nodes labeled by `text` that lie on east (spatial axis `E`) of the context node represented by the variable `$img` (photos of music bands). Among these nodes the predicates `[W,1]` and `[N,1]` select the node that is the first from west and from north, i.e. the name of the bands (e.g. `Coldplay` and `Radiohead`). The spatial location path (`2.3`) selects the nodes that are first from west, third from north, and are not contained in other nodes (predicate `[max]`) among all nodes on east of each photo of music bands (see the node `..ul/li[2]/p[2]` in Figure 2). So, all similar bands are selected.

Example 2 demonstrates that human users can exploit visual patterns that they recognize on the screen. They can define spatial location paths based on layout and DOM structure[1]. We argue that in the future machines will be able to perform a likewise step of exploiting visual patterns for learning extraction rules, because the spatial location paths are conceptually simpler than their XPath 1.0 based counterparts.

3. THE SXPATH LANGUAGE

The SXPath language extends the W3C's XPath 1.0 [25] with spatial capabilities. Intuitive navigational features and querying capabilities of XPath 1.0 are central to most XML-related technologies. For this reason XPath 1.0 has attracted great attention in the computer science research community. Even though XPath 2.0 reached recommendation status, it is not well suited for our objectives. In fact, the XPath Core 2.0 query evaluation is PSpace-complete [23] in contrast to the polynomial time complexity of XPath 1.0 [12, 20] (combined complexity). Furthermore, a strong theoretical background on XPath 1.0 is currently available, and specific subsets of the language with attractive properties and essential language features have been characterized [6]. These investigations lay the foundations for understanding the formal properties of SXPath. The SXPath language adopts the path notation of XPath 1.0 augmented by a user-friendly syntax having a natural semantics that enables spatial querying. In particular, SXPath provides: (i) A new set of *spatial axes* that allow for selecting nodes that have a specific spatial relation w.r.t. *context nodes*; (ii) New node set functions, namely *spatial position functions*, that allow for expressing predicates working on positions of nodes in the plane. As already shown in Example 2, such extensions are very useful in practice. In this section, we first define the SXPath data model and describe its new spatial capabilities. Then we provide the formal syntax and semantics, and the computational complexity of the language.

3.1 Data Model

In this section we present the SXPath data model, namely *Spatial DOM* (SDOM), and some auxiliary functions used in the rest of the paper. The SDOM considers relations existing among the visual representation of DOM nodes defined as follows.

Definition 1. Let n be a node in the DOM of a Web page, the *minimum bounding rectangle* (MBR) of n is the minimum rectangle r that surrounds the contents of n (see Figure 2) and has sides parallel to the axes (x and y) of the Cartesian plane. The function $mbr(n)$ returns the rectangle r assigned to a DOM node by the layout engine of a Web browser. We call r_x and r_y the segments that are obtained as the projection of r on the x-axis and the y-axis respectively. Then, each side of the rectangle is represented by the segments (r_x^-, r_x^+) and (r_y^-, r_y^+), where r_x^- (resp. r_y^-) denote the *infimum* on the x-axis (y-axis) and r_x^+ (resp. r_y^+) denote the *supremum* on the x-axis (y-axis) of the segments r_x and r_y.

Considering the function $mbr(n)$ given in Def. 1, a Web page can be modeled as a DOM enriched by spatial relations existing between MBRs. For representing such spatial relations we adopt the *Rectangle Algebra* (RA) qualitative spatial model [4], which allows for representing all possible relations between rectangles the sides of which are parallel to the axes of some orthogonal basis in a 2-dimensional Euclidean space. RA is a straightforward extension of the standard model for temporal reasoning, the *Interval Algebra* (IA) [3], to the 2-dimensional case. IA models the relative position between pairs of segments by a set of 13 atomic relations (R_{int}), namely $before$ (b), $meet$ (m), $overlap$ (o), $start$ (s), $during$ (d), $finish$ (f), together with theirs inverses $\{bi,\ mi,\ oi,\ si,\ di,\ fi\}$ and the relation $equal$ (e). Let s and s_1 be two segments the IA relation $s\ b\ s_1$ represents that the segment s is preceded by the segment s_1. For a pictorial representation of IA relations see Table 4 in Appendix A. Let a and b be two rectangles, a RA relation between them is written as $a\ \rho\ b$ where $\rho = (\rho_x, \rho_y)$ is a pair of IA relations. The RA relation holds iff the IA relations $a_x\ \rho_x\ b_x$ and $a_y\ \rho_y\ b_y$ hold for segments that are obtained as projections of rectangle sides along x (i.e. $a_x,\ b_x$) and y (i.e. $a_y,\ b_y$) respectively. The expressiveness of RA covers the modeling of all qualitative spatial relations between two MBRs. For a pictorial representation of RA relations see Figure 7 in Appendix A. Furthermore, since the RA model is an algebra, it holds many important properties (e.g. invertibility) that allow optimized query evaluation.

Definition 2. SDOM is a node labeled sibling ordered tree [11, 15, 17] enriched by RA relations. It is described by the following 5-tuple:

$$SDOM = \langle V, R_{\Downarrow}, R_{\Rightarrow}, A, f_s \rangle$$

where:

- V is the set of labeled DOM nodes. $V = V_v \cup V_{nv}$, where V_v is the set of nodes visualized on screen, and V_{nv} is the set of nodes that are not visualized.
- $R_{\Downarrow}$ is the $firstchild$ relation. Let n and n' be two nodes in V, $nR_{\Downarrow}n'$ holds iff n' is the first child of n.
- $R_{\Rightarrow}$ is the $nextsibling$ relation. Let n and n' be two nodes in V, $nR_{\Rightarrow}n'$ holds iff n' is the next sibling of n.
- $A \subseteq V_v \times V_v$ is the set of arcs that represent spatial relations between pairs of nodes visualized on screen.
- Let R_{rec} be the set of RA relations, $f_s : A \to R_{rec}$ is the function that assigns to each element in A a RA relation in R_{rec}. So, let n and n' be two nodes in V_v, we have $a = (n, n') \in A$ holds iff $mbr(n)\ f_s(a)\ mbr(n')$.

Definition 3. Let Σ be the labeling alphabet (i.e., *"tags"*). We define the *node test function* $T : (\Sigma \cup \{*\} \cup \{text\}) \to 2^V$ (*"node test"*) which assigns to each label (XML tag or textual leaf node) the set of nodes labeled with it. Furthemore, $T(*) := V$.

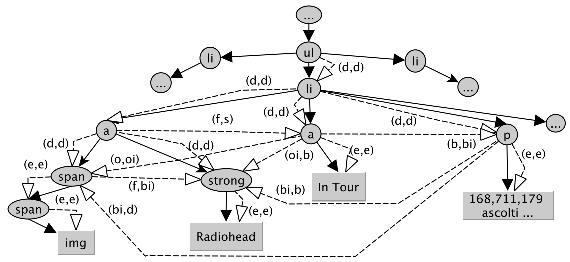


Figure 4: A SDOM Fragment of the Web Page Portion in Figure 2

Figure 4 depicts the SDOM for a portion of Web page shown in Figure 2. Solid arrows represent the classical DOM tree structure that is equivalent to the DOM depicted in Figure 3. Dashed labeled arcs represent the RA relations between pairs of nodes. For instance, the arc (d, d) between nodes `ul` and `li` represents the RA relation $mbr(ul)(d,d)mbr(li)$, i.e. both IA relations $mbr(ul)_x\ d\ mbr(li)_x$ and $mbr(ul)_y\ d\ \ mbr(li)_y$ hold, and means that the rectangle $mbr(li)$ is spatially contained in the rectangle $mbr(ul)$. In order to improve readability of the figure, spatial relations are represented only for a subset of nodes.

[1]In this example, the DOM structure in use was restricted to XML node types.

The SDOM model provides orders among nodes defined as follows.

Definition 4. Document order relations for a SDOM are:

- the *document total order* $<_{doc}$, already defined in [25]. Let $u, w \in V$ be two nodes, then $u <_{doc} w$ iff the opening tag of u precedes the opening tag of w in the (well-formed version of the) document.
- the document *directional total orders* $\leqslant_{\uparrow}$ (from south), $\leqslant_{\rightarrow}$ (from west), $\leqslant_{\downarrow}$ (from north), and $\leqslant_{\leftarrow}$ (from east). Let $u, w \in V_v$ be two nodes with MBRs $a = mbr(u)$ and $b = mbr(w)$ respectively, we have $u < w$ (or $u = w$) iff:
  - $a_y^- < b_y^-$ $(a_y^- = b_y^-)$ for $\leqslant_{\uparrow}$,
  - $a_x^- < b_x^-$ $(a_x^- = b_x^-)$ for $\leqslant_{\rightarrow}$,
  - $a_y^+ > b_y^+$ $(a_y^+ = b_y^+)$ for $\leqslant_{\downarrow}$,
  - $a_x^+ > b_x^+$ $(a_x^+ = b_x^+)$ for $\leqslant_{\leftarrow}$.
- the *containment partial order* $\leqslant_t$. Let $u, w \in V_v$ be two nodes with $a = mbr(u)$ and $b = mbr(w)$ respectively, we have $u \leq w$ iff $a_x^- \leqslant b_x^- \leqslant b_x^+ \leqslant a_x^+$ and $a_y^- \leqslant b_y^- \leqslant b_y^+ \leqslant a_y^+$. In particular, $u = w$ if $a_x^- = b_x^-$, $b_x^+ = a_x^+$, $a_y^- = b_y^-$, and $b_y^+ = a_y^+$.

In the following we introduce the concept of layering adopted for computing spatial order among nodes of a given node set.

Definition 5. Let $\Gamma \subseteq V_v$ be a set of SDOM nodes, for each spatial order $\leqslant_z$, where $\leqslant_z \in \{\leqslant_{\uparrow}, \leqslant_{\rightarrow}, \leqslant_{\downarrow}, \leqslant_{\leftarrow}, \leqslant_t\}$, the *spatial layering* of nodes in Γ w.r.t. $\leqslant_z$ is an ordered list of non-empty lists $L_{\leqslant_z}(\Gamma) = \{l_{1_{\leqslant_z}}, \cdots, l_{h_{\leqslant_z}}\}$, where let u, w be two nodes in Γ:

- if $u = w$ w.r.t. $\leqslant_z$, then $u, w \in l_{i_{\leqslant_z}}$ for some $i_{\leqslant_z} \in \{1, \cdots, h_{\leqslant_z}\}$.
- if $u < w$ w.r.t. $\leqslant_z$, then $u \in l_{i_{\leqslant_z}}$ and $w \in l_{j_{\leqslant_z}}$ where $i_{\leqslant_z}$ and $j_{\leqslant_z}$ are the smallest numbers such that $i_{\leqslant_z}, j_{\leqslant_z} \in \{1, \cdots, h_{\leqslant_z}\}$ and $i_{\leqslant_z} < j_{\leqslant_z}$.

Example 3. Considering Figure 5 and the spatial order $\leqslant_{\uparrow}$, we obtain the layering $L_{\leqslant_{\uparrow}}(\{n_1, \ldots, n_6\}) = \{l_{1_{\leqslant_{\uparrow}}}, l_{2_{\leqslant_{\uparrow}}}, l_{3_{\leqslant_{\uparrow}}}\}$ where $l_{1_{\leqslant_{\uparrow}}} = \{n_1\}$, $l_{2_{\leqslant_{\uparrow}}} = \{n_2, n_4, n_6\}$, and $l_{3_{\leqslant_{\uparrow}}} = \{n_3, n_5\}$. The layering corresponds to the following directional total order from south: $n_1 \leqslant_{\uparrow} n_2 =_{\uparrow} n_4 =_{\uparrow} n_6 \leqslant_{\uparrow} n_3 =_{\uparrow} n_5$.

In order to facilitate the definition of SXPath language semantics we introduce the concepts of *spatial index function* and *last index function* as follows.

Definition 6. Let $\Gamma \subseteq V_v$ be a set of SDOM nodes, and let $L_{\leqslant_z}(\Gamma) = \{l_{1_{\leqslant_z}}, \cdots, l_{h_{\leqslant_z}}\}$ be the spatial layering of nodes in Γ w.r.t. $\leqslant_z$, where $\leqslant_z \in \{\leqslant_{\uparrow}, \leqslant_{\rightarrow}, \leqslant_{\downarrow}, \leqslant_{\leftarrow}, \leqslant_t\}$. The *spatial index function* $pidx_{\leqslant_z}(u, \Gamma)$ returns the index of the layer which the node u belongs to (where 1 is the smallest index). The *last directional index function* $plast_{\leqslant_z}(\Gamma)$ returns the index $h_{\leqslant_z}$.

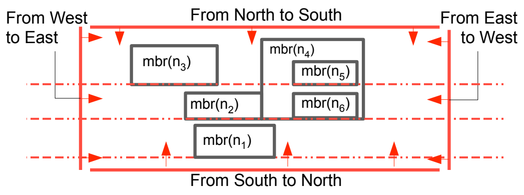


Figure 5: Ordering directions

3.2 Spatial Axes

RA relations, stored in the SDOM, represent all qualitative spatial relations between MBRs, but they are too fine grained, verbose and not intuitive for querying. Therefore, for defining SXPath spatial axes we consider the more synthetic and intuitive *Rectangular Cardinal Relation* (RCR) [18] and *Rectangular Connection Calculus* (RCC) [21] models. In particular, RCRs express spatial axes that represent directional relations between MBRs. RCRs are computed by analyzing the 9 regions (cardinal tiles) formed, as shown in Figure 6, by the projections of the sides of the reference MBR (i.e. r). The *atomic* RCRs are: *belongs to* (B), *South* (S), *SouthWest* (SW), *West* (W), *NorthWest* (NW), *North* (N), *NorthEast* (NE), *East* (E), and *SouthEast* (SE). Using the symbol ":" it is possible to express *conjunction* of atomic RCRs. For instance, by considering Figure 6, r `E:NE` r_1 means that the rectangle r_1 lies on east and (symbol ":") north-east of the rectangle r. Moreover, the RCR model allows for expressing uncertain (disjunction of) directional relations: for example r `E|E:NE` r_1 means that r_1 lies on `E` or (symbol "|") on `E:NE` of r. Furthermore, the three relations inspired by the RCC calculus, namely: *contained* (CD), *container* (CR), and *equivalent* (EQ), allow for expressing spatial axes that represent topological relations between MBRs. For instance, $r\ CD\ r_2$ means that the rectangle r_2 is spatially contained in the rectangle r.

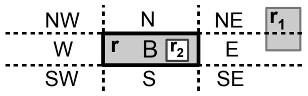


Figure 6: Cardinal tiles

Each spatial axes (expressed by a RCR or a topological relation) corresponds to a set of RA relations expressed by the Cartesian product shown in tables 1 and 2. Formally:

Definition 7. Let R_{card}, R_{topo} and R_{rec} be the set of (atomic and conjunctive) RCRs, topological and RA relations respectively, then the *mapping function* $\mu : 2^{R_{card}} \cup R_{topo} \rightarrow 2^{R_{rec}}$ that maps RCRs and topological relations in terms of RA relations is:

$$\mu(R) = \begin{cases} \text{as in table 1} & \text{if } R \in R_{card} \\ \text{as in table 2} & \text{if } R \in R_{topo} \\ \mu(R_1) \cup \ldots \cup \mu(R_k) & \text{if } R = R_1|\ldots|R_k \end{cases}$$

Example 4. For the relation NW reported in Tab. 1 we have $\mu(NW) = \{b, m\} \times \{bi, mi\} = \{(b, bi), (b, mi), (m, bi), (m, mi)\}$.

×	{b,m}	{o,fi}	{di}	{e,s,d,f}	{si,oi}	{bi,mi}
{b,m}	SW	W:SW	SW:W:NW	W	W:NW	NW
{o,fi}	S:SW	B:S:SW: W	B:S:SW: W:NW:N	B:W	B:W: NW:N	N:NW
{di}	S:SW:SE	B:S:SW: W:E:SE	All	B:W:E	B:W:NW: N:NE:E	NW:N:NE
{e,s,d,f}	S	B:S	B:S:N	B	B:N	N
{si,oi}	S:SE	B:S:E:SE	B:S:N:NE:E:SE	B:E	B:N:NE:E	N:NE
{bi,mi}	SE	E:SE	NE:E:SE	E	E:NE	NE

Table 1: RCRs as cartesian products of RA Relations

Top. rel.		**RA-relations**		
EQ	:=	{ (e,e) }		
CR	:=	{{ e, si, di, fi }	×	{ e, si, di, fi }} − {(e,e)}
CD	:=	{{ e, s, d, f }	×	{ e, s, d, f }} − {(e,e)}

Table 2: Topological relations mapped into RA Relations

Like in XPath 1.0, SXPath axes are interpreted binary relations $\chi \subseteq V \times V$. Let $self := \{\langle u, u \rangle | u \in V\}$ be the reflexive axis, remaining SXPath axes are partitioned in two sets: Δ_t and Δ_s. The set Δ_t contains *traditional* XPath 1.0 axes (*forward*, e.g. child, descendant, and *reverse*, e.g. parent, ancestor) that allow for navigating along the tree structure. They are encoded in terms of their *primitive* relations (i.e. *firstchild*, *nextsibling* and their inverses), as shown in [11, 12]. The set Δ_s contains the novel (directional and topological) *spatial axes* corresponding to the RCRs and Topological Relations that allow for navigating along the spatial RA relations. In the following we formally define spatial axes in terms of their primitive RA relations stored in the SDOM.

Definition 8. SXPath *spatial axes* are interpreted binary relations $\chi_s \subseteq V_v \times V_v$ of the following form $\chi_s = \{\langle u, w \rangle | u, w \in V_v \wedge mbr(u)\ \rho\ mbr(w) \wedge \rho \in \mu(R)\}$. Here, R is the RCR or topological relation that names the spatial axis relation and μ is the mapping function.

In order to define the semantics of SXPath we give a trivial generalization of the document total order w.r.t. an axis [25] and define the relative index function.

Definition 9. The *document total order w.r.t. the axis* χ written as $<_{doc,\chi}$ is: (i) the reverse document order $>_{doc}$ when χ is a traditional reverse axis [25]; (ii) the document total order $<_{doc}$ (Def. 4) otherwise.

Definition 10. Let $\Gamma \subseteq V$ be a set of nodes, the *index function w.r.t. the axis* χ written as $idx_\chi(u, \Gamma)$ returns the index of the node u in Γ w.r.t. $<_{doc,\chi}$ (where 1 is the smallest index).

3.3 Syntax

In this section we present the formal syntax of two important fragments of SXPath, namely: *Core SXPath* and *Spatial Wadler Fragment*. The grammar of their unabbreviated version is incrementally provided. Like XPath 1.0, SXPath has also a number of syntactic abbreviations, used in Example 2 of Sec. 2. But they are just syntactic sugar and are not considered in the following.

We define the fragment of *Core SXPath* as the navigational core of SXPath. It is obtained extending Core XPath [11] (the navigational core of XPath 1.0) by spatial axes introduced in Sec. 3.2.

Definition 11. The EBNF grammar of Core SXPath is:

```
locpath     ::= ‘/’ locpath | locpath ‘/’ locpath |
                locpath ‘|’ locpath | locstep
locstep     ::= axis ‘::’  t | locstep ‘[’ bexpr ‘]’
bexpr       ::= bexpr ‘and’ bexpr | bexpr ‘or’ bexpr|
                ‘not(’ bexpr ‘)’ | locpath.
axis        ::= xpathAxis | spatialAxis
xpathAxis   ::= ‘self’ | ‘child’ | ‘parent’ | ···
spatialAxis::= topAxis | dirAxis
topAxis     ::= ‘EQ’ | ‘CD’ | ‘CR’
dirAxis     ::= ‘B’ | ··· | ‘U’
```

where:

- `locpath` is the start symbol.
- `axis` denotes axis relations that are traditional XPath axis (`xpathAxis`) and spatial axes (`spatialAxis`).
- `t` is the node test.

Since Core SXPath lacks the ability to exploit spatial position of nodes, we define the Spatial Wadler Fragment (SWF). SWF is the spatial extension of the *Extended Wadler Fragment* (EWF) [12]. It allows positional, logical and arithmetic features.

Definition 12. The syntax of the SWF-Queries is defined by the Core SXPath grammar with the following extensions.

```
expr        ::= locpath | bexpr | nexpr
dirAxis     ::= ‘B’ | ··· | ‘U’ | disjDirAxis
bexpr       ::= bexpr ‘and’ bexpr | bexpr ‘or’ bexpr|
                ‘not(’ bexpr ‘)’ | nexpr relop nexpr |
                sexpr relop sexpr | locpath |
                locpath relop sexpr | locpath relop number
nexpr       ::= number | nexpr arithop nexpr |
                ‘position()’ | ‘last()’ | ‘posFromS()’ |
                ‘lastFromS()’ | ‘posFromN()’ | ‘lastFromN()’|
                ‘posFromW()’ | ‘lastFromW()’ | ‘posFromE()’ |
                ‘lastFromE()’ | ‘posSpatialNesting()’
sexpr       ::= string
arithop     ::= ‘+’ | ‘-’ | ‘*’ | ‘div’ | ‘mod’
relop       ::= ‘=’ | ‘!=’ | ‘<’ | ‘<=’ | ‘>’ | ‘>=’
```

where:

- `expr` (instead of `locpath`) is the start symbol.
- `dirAxis` considers disjunctive directional relation `disjDirAxis`[2].
- `nexpr` extends traditional XPath numerical expressions with spatial position functions. `number` and `string` denote constant real-valued numbers and strings respectively.

We don't give here the syntax of the full SXPath language. The reason for this is lack of space. However, by considering [25] extending the syntax is an easy exercise.

[2]Core SXPath does not allow for querying spatial orders, so spatial axes corresponding to disjunction of *RCRs* do not add expressiveness.

3.4 Semantics

In this section we define the semantics of SXPath by adopting the denotational semantics proposed in [26]. Like in XPath 1.0 the main structural feature of SXPath are *expressions*, that return a value from one of the following types: *node set*, *number*, *string*, or *Boolean*. Every expression evaluates relative to a *context*. Context and domain of context are defined in the following as extension of the definition given in [26].

Definition 13. The *Context* is the following 12-tuple:

$$\vec{c} = \langle n, p_{<_{doc}}, s_{<_{doc}}, p_{\leqslant_\uparrow}, s_{\leqslant_\uparrow}, p_{\leqslant_\rightarrow}, s_{\leqslant_\rightarrow}, p_{\leqslant_\downarrow}, s_{\leqslant_\downarrow}, p_{\leqslant_\leftarrow}, s_{\leqslant_\leftarrow}, p_{\leqslant_t} \rangle$$

where: (i) n is a *context node*. (ii) $p_{\leqslant_z}$ are natural numbers that indicate the *context positions* w.r.t. orders $\leqslant_z \in \{<_{doc}, \leqslant_\uparrow, \leqslant_\rightarrow, \leqslant_\downarrow, \leqslant_\leftarrow, \leqslant_t\}$. (iii) $s_{\leqslant_z}$ are natural numbers that indicate the *context sizes* w.r.t. orders in $\{<_{doc}, \leqslant_\uparrow, \leqslant_\rightarrow, \leqslant_\downarrow, \leqslant_\leftarrow\}$.

Definition 14. The *Domain of Contexts* is the following set:

$$\mathbf{C} = \{\langle n, p_{<_{doc}}, s_{<_{doc}}, p_{\leqslant_\uparrow}, s_{\leqslant_\uparrow}, p_{\leqslant_\rightarrow}, s_{\leqslant_\rightarrow}, p_{\leqslant_\downarrow}, s_{\leqslant_\downarrow}, p_{\leqslant_\leftarrow}, s_{\leqslant_\leftarrow}, p_{\leqslant_t} \rangle$$
$$\mid n \in V \wedge 1 \leqslant p_{<_{doc}} \leqslant s_{<_{doc}} \leqslant |V| \wedge 1 \leqslant p_{\leqslant_z} \leqslant s_{\leqslant_z} \leqslant |V_v| \;\wedge 1 \leqslant p_{\leqslant_t} \leqslant |V_v|\}$$

where $\leqslant_z \in \{\leqslant_\uparrow, \leqslant_\rightarrow, \leqslant_\downarrow, \leqslant_\leftarrow\}$.

In the following we first define the auxiliary semantic function for location paths then we give the semantics of SXPath.

Definition 15. Location path semantics. Considering the grammar defined in the Sec. 3.3. Let π, π_1, π_2 be location paths, let $locstep$ be a location step over an axis χ, let $bexpr$ be a boolean expression and let n be a context node, then the *semantics function of SXPath location paths* $P : LocationPath \rightarrow node \rightarrow nodeset$ is defined as follows:

$$P\llbracket /\pi \rrbracket(n) := P\llbracket \pi \rrbracket(root)$$
$$P\llbracket \pi_1/\pi_2 \rrbracket(n) := \{n_2 | n_1 \in P\llbracket \pi_1 \rrbracket(n) \wedge n_2 \in P\llbracket \pi_2 \rrbracket(n_1)\}$$
$$P\llbracket \pi_1 | \pi_2 \rrbracket(n) := P\llbracket \pi_1 \rrbracket(n) \cup P\llbracket \pi_2 \rrbracket(n)$$
$$P\llbracket axis :: t \rrbracket(n) := \{n' \mid \llbracket axis \rrbracket(n, n')\} \cap T(t)$$
$$P\llbracket locstep[bexpr] \rrbracket(n) := \{n' \mid \vec{W} = P\llbracket locstep \rrbracket(n) \wedge n' \in \vec{W} \wedge$$
$$\varepsilon\llbracket bexpr \rrbracket(\vec{c_{n'}}) = true \wedge \vec{c_{n'}} := \langle w, idx_\chi(n', \vec{W}), |\vec{W}|, pidx_{\leqslant_\uparrow}(n', \vec{W})$$
$$plast_{\leqslant_\uparrow}(\vec{W}), pidx_{\leqslant_\rightarrow}(n', \vec{W}), plast_{\leqslant_\rightarrow}(\vec{W}), pidx_{\leqslant_\downarrow}(n', \vec{W}),$$
$$plast_{\leqslant_\downarrow}(\vec{W}), pidx_{\leqslant_\leftarrow}(n', \vec{W}), plast_{\leqslant_\leftarrow}(\vec{W}), pidx_{\leqslant_t}(n', \vec{W})\rangle\}$$

where T is the node test function defined in Def. 3, and the semantics of an `axis` ($\llbracket axis \rrbracket$) is: (i) $\llbracket spatialAxis \rrbracket := \{(n, n') \mid mbr(n)\ \rho\ mbr(n') \wedge \rho = \mu(spatialAxis)\}$ for spatial axes, (ii) the semantic of traditional axis already described in [17].

Definition 16. Given an SXPath expression e and a context $\vec{c}$, the semantics function $\varepsilon : SXPathExpression \rightarrow \mathbf{C} \rightarrow SXPathType$ returns number, string, boolean, or node set values:

$$\varepsilon\llbracket \pi \rrbracket(\vec{c}) := P\llbracket \pi \rrbracket(n) \qquad \varepsilon\llbracket posSpatialNesting() \rrbracket(\vec{c}) := p_t$$
$$\varepsilon\llbracket position() \rrbracket(\vec{c}) := p_{<_{doc}} \qquad \varepsilon\llbracket last() \rrbracket(\vec{c}) := s_{<_{doc}}$$
$$\varepsilon\llbracket posFromN() \rrbracket(\vec{c}) := p_{\leqslant_\downarrow} \qquad \varepsilon\llbracket lastFromN() \rrbracket(\vec{c}) := s_{\leqslant_\downarrow}$$
$$\varepsilon\llbracket posFromS() \rrbracket(\vec{c}) := p_{\leqslant_\uparrow} \qquad \varepsilon\llbracket lastFromS() \rrbracket(\vec{c}) := s_{\leqslant_\uparrow}$$
$$\varepsilon\llbracket posFromW() \rrbracket(\vec{c}) := p_{\leqslant_\rightarrow} \qquad \varepsilon\llbracket lastFromW() \rrbracket(\vec{c}) := s_{\leqslant_\rightarrow}$$
$$\varepsilon\llbracket posFromE() \rrbracket(\vec{c}) := p_{\leqslant_\leftarrow} \qquad \varepsilon\llbracket lastFromE() \rrbracket(\vec{c}) := s_{\leqslant_\leftarrow}$$
$$\varepsilon\llbracket Op(e_1, ..., e_m) \rrbracket(\vec{c}) := F\llbracket Op \rrbracket(\varepsilon\llbracket e_1 \rrbracket(\vec{c}), \ldots, \varepsilon\llbracket e_m \rrbracket(\vec{c}))$$

where: (i) P is the function in Def. 15. (ii) $F\llbracket Op \rrbracket : O_1 \times ... \times O_m \rightarrow O$, is defined in Tab. 3 (for a complete list of functions we refer to [11, 25]). The meaning of each expression is given by the meaning of its subexpressions.

$F\llbracket$ const. number i: $\rightarrow$ num $\rrbracket()$	::= i
$F\llbracket$ ArithOp: num × num → num $\rrbracket$ (i_1, i_2)	::= i_1 ArithOp i_2, where ArithOp∈ $\{+,-,*,/,\%\}$
$F\llbracket$ constant string s: $\rightarrow$ str $\rrbracket()$	::= s
$F\llbracket$ RelOp: num × num → bool $\rrbracket$ (i_1, i_2)	::= i_1 RelOp i_2, where RelOp ∈ $\{=, \neq, \leqslant, <, \geqslant, >\}$
$F\llbracket$ RelOp: str × str → bool $\rrbracket$ (s_1, s_2)	::= s_1 RelOp s_2
$F\llbracket$ RelOp: nset × const. string → bool $\rrbracket$ (Γ, s)	::= $\exists u \in \Gamma : strval(u)$ RelOp s
$F\llbracket$ RelOp: nset × bool → bool $\rrbracket$ (Γ, b)	::= $F\llbracket$ boolean $\rrbracket(\Gamma)$RelOp b
. . .	

Table 3: Semantics of Functions in SXPath

3.5 Complexity Issues

This section summarizes the computational complexity results of the SXPath query evaluation problem. We show that *Core SXpath*, *Spatial Wadler Fragment* (SWF), and *Full SXpath* allow polynomial time combined complexity query evaluation with increasing degree of the polynomial. In the following theorems we denote by D the XML document, which has size $\Theta(|V|)$, where $|V|$ is the number of nodes of its SDOM representation. It is noteworthy that the SDOM (see Sec. 3.1) has size $O(|V|^2)$.

Theorem 1. Core SXPath queries can be evaluated in time $O(|D|^2 * |Q|)$, where $|D|$ is the size of the XML document, and $|Q|$ is the size of the query Q.

Proof Sketch 1. In [12] it has been shown that Core XPath 1.0 has linear time combined complexity. We extend Core XPath 1.0 evaluation algorithm, proposed in [12], by spatial axes function. For evaluating spatial axes, spatial relations between any pair of visualized nodes must be considered. Since in the SDOM any visible node is spatially related to any other visible node, there are $O(|V_v|^2)$ many spatial relations to be considered in addition to the $O(|V|)$ many relations of the DOM incurring a higher polynomial worst case complexity. The detailed description of the proof is found in the Appendix B.1.

Considering the SWF syntax presented in Sec. 3.3, we can give the following complexity result.

Theorem 2. SXPath queries that fall in the *Spatial Wadler Fragment* can be evaluated in time $O(max(|D|^3 * |Q|, |D|^2 * |Q|^2))$, where D is the XML document, and Q is a *SWF* query.

Proof Sketch 2. The basic idea for proving this theorem is to adopt the top-down/bottom-up evaluation strategy proposed in [12] for EWF queries. The most costly operation in SWF is the evaluation of spatial location paths that require the construction of the full context. This operation, which requires the computation of the layering introduced in Def. 5, generates a higher polynomial worst case complexity w.r.t. EWF [12]. The detailed description of the proof is found in the Appendix B.2.

Theorem 3. Full SXPath queries can be evaluated in time $O(|D|^4 * |Q|^2)$ and space $O(|D|^2 * |Q|^2)$, where D is the XML document, and Q is a *Full SXPath* query.

Proof Sketch 3. Full SXPath combined complexity bound can be proved by adopting a seamless extension of Full XPath queries evaluation strategy proposed in [12] by spatial operators. The most costly operation in SXPath still remains the evaluation of XPath operations on any input context [12] resulting in the same combined complexity bound of XPath 1.0. The detailed description of the proof is found in the Appendix B.2.

4. IMPLEMENTATION AND EXPERIMENTS

We have implemented the language in a system that embeds the Mozilla browser and computes the SDOM in real time at each variation of visualization parameters (i.e. screen resolution, browser window size, font type and dimension). In this way for each Web page and visualization condition there is a unique corresponding SDOM that enables the user to query the Web page by considering what s/he sees on the screen. For computing the SDOM the system exploits the Mozilla XULRunner[3] application framework that allows for implementing the function mbr (see Def. 1) that acquires coordinates of the MBRs assigned by the layout engine to each visualized DOM node. Queries posed by users are parsed and evaluated on the SDOM by the *query evaluator* that returns query answers, and visualizes returned SDOM nodes on the Web page by rectangles with highlighted borders.

[3]https://developer.mozilla.org/en/XULRunner_1.9.2_Release_Notes

4.1 System Efficiency

For evaluating the SXPath system efficiency we have performed experiments that evidence the practical system behavior for both increasing document and query sizes. Due to lack of space, we illustrate and compare results only for queries falling in SWF. For evaluating data efficiency of the query evaluator, which is independent from SDOM construction, we have used a fixed SWF query having size $|Q|$=167. Then, we have computed the needed query time for increasing documents sizes from $|D|$=50 to double the maximum size we found on real-world Web pages, i.e. $|D|$=7500 nodes. Figure 8, in the Appendix C.1, depicts obtained curves (in log log scale) indicating polynomial time efficiency w.r.t. document size. In particular, curves slopes are all 1, which indicates linear time behavior. The time needed for constructing the SDOM is depicted as a straight line on the log log scale (see Figure 9 in the Appendix C.1). In particular, the line slope is 2 indicating quadratic runtime behavior — as expected. Thus, the runtime requirements for the integrated system with SDOM construction and query evaluation remains polynomial. For evaluating query efficiency we tested the whole system with fixed document sizes ($|D|$=1000, $|D|$=3000, $|D|$=6000) and increasing query sizes. Figure 10, in the Appendix C.1, depicts obtained curves on a log log scale. For this experiment time grows linearly with the query size (curves slopes are all 1). Thus, empirical behavior of the system is better than expected from the SWF worst case upper bound. Details and rationales about Web pages and queries adopted in the experiments are given in Appendix C.1.

4.2 Language Usability

In this section we present experiments aimed at assessing the usability of our approach and the enhancements provided by SXPath language over XPath 1.0. For the evaluation we have considered the situation of an expert user aiming at manually developing Web wrappers for Deep and Social Web sites. We assume that this expert has a good command of XPath. S/he visualizes and explores Web pages by using the SXPath system where the embedded Mozilla browser is configured using a typical setting for which Web pages are generally developed (i.e. screen resolution of 1280x800 pixels, browser window size of 1024x768 pixels, standard font type and dimension). We investigate the following questions in our evaluation: (1) How robust is the SXPath language w.r.t. Web browser and visualization parameters variations? (2) How "easy" is it for this expert to understand and apply SXPath? (3) How suitable is SXPath for manual wrapper construction, giving the expert the possibility to look only at the visualized Web page, in comparison to XPath? (4) How suitable is SXPath for manual wrapper construction, giving the expert full insight into the SDOM/DOM, when compared against XPath? (5) How transportable is SXPath across multiple Web sites in comparison to XPath? Experiments described in the following explore these questions. In the experiments we have used a dataset of 125 pages obtained by collecting 5 pages per site from 25 Deep Web sites already exploited for testing wrapper learning approaches [10, 19, 27] (see Tab. 7 in Appendix C.2). Experiments have involved ten users who were students well trained in XPath with no experience in SXPath.

Experiment 1. In this experiment we have explored question (1) by evaluating the robustness of SXPath w.r.t. Web browser and visualization parameters variations. Firstly, considering the dataset of 25 Web sites listed in Tab. 7 in Appendix C.2, we have varied values for screen resolution (i.e. 1280x800, 1024x768, 800x600 pixels), browser window size (i.e. 1024x768, 800x600, 640x480 pixels), font dimension (i.e. small, standard, large, huge), and font type (i.e. times, times new roman, tahoma, arial) and tested if the spatial relations between node change. We have observed that modifications in: (i) Screen resolution and font type do not cause changes in the spatial arrangement of visualized nodes. (ii) Browser window size does not affect 20 Web sites (highlighted by "†" in Tab. 7 in Appendix C.2) because they are based on absolute positioning properties, whereas for 5 sites we have changes in the spatial arrangement of visualized nodes. (iii) Font dimension affects the spatial arrangement of visualized nodes. In particular, by using large and huge fonts, visual appearance of Web pages strongly changes w.r.t. standard font dimension. Thus, modifications in screen resolution and font type do not affect query results, whereas changes in browser window size and font dimension could affect the query result. However, this aspect does not impact SXPath usability because the SXPath system: (i) Embeds the browser and computes the SDOM real time (i.e. at each changing of visualization parameters). So, users can query what they see on the screen at each moment. (ii) SXPath queries are stored with visualization parameter settings adopted by the user during the query design process. Thus, when a query is reused on a Web page the embedded browser is set with visualization parameters for which the query has been designed.

Secondly, we have constructed a dataset of 200 Deep Web pages, presenting either data records or tables, randomly selected from the *www.completeplanet.com* Web site (the largest portal of Deep Web sites). We have visualized pages by the four most used Web browsers, i.e. Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Safari set with default visualization parameters and standard window size (i.e. 1024x768 pixels). We have observed that, even if coordinates returned by different browsers for rectangles bounding visualized nodes were slightly different, the qualitative spatial arrangement of visualized SDOM nodes remained stable. Such a result was expected because rendering rules of most diffused layout engines[4] have been strongly standardized as it is possible to verify by tests available in [1]. So, SXPath queries do not depend on the browser for fixed visualization parameters.

Experiment 2. In this experiment we have investigated question (2) by evaluating the effort needed for learning SXPath and the feeling perceived by users in applying the language. We have defined the user task "identify product data records and extract product names and prices" from the Web site *www.bol.de*. We have asked users to learn the SXPath language and complete the task by writing a sound and complete SXPath query looking only at the visualized Web page. For learning the language, we have provided users with a short manual explaining spatial axis and position function behavior. We have computed both the number of attempts and the time needed by users for defining the query by using the SXPath system. We have observed that users have required an average number of 4.3 and 3.6 attempts for recognizing name and price respectively, and an average time equal to 57 minutes for learning the SXPath language and accomplishing the assigned task. For evaluating the level of "easiness" and "satisfaction" perceived by users in learning and applying the SXPath language, at the end of the experiment, we have asked users to answer a questionnaire based on the seven-item Likert scale (described in Appendix C.2) where -3 corresponds to "very difficult/unsatisfactory", and +3 corresponds to "very easy/satisfactory". The language was assessed as easy to learn and quite satisfactory to use.

Experiment 3. In this experiment we have explored question (3) by assessing whether spatial information is helpful for a human aiming at manually writing Web wrappers. We have asked users to perform the same extraction task of Experiment 2 by identifying, for each Web site in the dataset and only by looking at the displayed Web pages, the best XPath and SXPath queries they could achieve by using at the most 5 attempts. Results (shown in Tab. 7 in the Appendix C.2) indicate average precision of 99% and recall of 100% for SXPath and average precision of 42% and recall of 99% for pure XPath 1.0. Users were able to define a good SXPath query by using 2 attempts on average, whereas all the 5 available attempts were not enough for finding a good pure XPath query for all Web pages in the dataset. The experiment clearly shows the advantage of SXPath over pure XPath. In fact, only SXPath makes spatial layout information explicitly available for querying.

Experiment 4. In this experiment we have investigated question (4). We have asked users to perform the same extraction task of Experiment 2 by identifying, for each Web site in the dataset and looking at both visualized Web pages and internal page structures (i.e. DOM and SDOM), sound and coplete XPath and SXPath queries by using at the most 10 minutes. Results (shown in Tab. 7 in the Appendix C.2) indicate that: (i) For writing queries users needed 4.2 and 2.7 average attempts for pure XPath and SXPath respectively. Since the number of attempts is proportional to needed time, a greater number of attempts indicate that to write queries in pure XPath requires more time in comparison to SXPath. (ii) In defining pure XPath queries all users have shown the tendency to write absolute location paths by deeply analyzing the DOM structure, hence the average number of location steps for pure XPath and SXPath has been 18.9, and 5.3 respectively. Hence, we asked users to write queries by using relative pure XPath location paths by using at most 5 minutes. We observed that further attempts needed by users are proportional to the complexity of the DOM structure. Such an experiment clearly shows that manually writing queries in pure XPath is more "complex" in comparison to SXPath because XPath requires the navigation of very intricate DOM structures, whereas SXPath mainly requires to look at the displayed Web page.

Experiment 5. In this experiment we have aimed at answering question (5) by evaluating whether SXPath queries are more general and abstract than pure XPath queries given different Web pages and the same extraction task. Firstly, we have chosen the subset of Web sites in the dataset having product names presented by the same visual pattern (such sites, highlighted by "⊙" in Tab. 7 of Appendix C.2, have different internal representations). We have observed that it is possible to use the same sound and complete spatial location path for extracting product names on all these Web sites. Instead, different XPath location paths are needed. Secondly, we have asked users to write a SXPath query aimed at extracting the list of friends in a social network randomly chosen among those listed in Tab. 8 in Appendix C.2. Then we have asked users to try to apply the same query to other remaining social networks in the list. Even though the internal tag structure of various social networks differ strongly (so different pure XPath queries are needed), all users have been able to use almost the same SXPath query for all social network Web sites. This experiment points out that SXPath allows for more general and abstract queries, that are independent from the internal structure of Web pages, in comparison to XPath.

Experiments provide a strong evidence for believing that humans aiming at manually defining Web wrappers and manipulating Web pages, may benefit from using SXPath navigation instead of pure XPath navigation. Moreover, the transportability of SXPath queries from one Web site to the next simplify manual definition of Web wrappers and can also support wrapper induction from sparsely annotated data, while the lack of such transportability observed for pure XPath is detrimental for both manual wrapper definition and wrapper induction. A full investigation of SXPath implications for wrapper induction effectiveness, however, goes beyond space and scope of this paper. Details and rationales about all previous experiments are given in Appendix C.2.

[4]Trident for Microsoft Internet Explorer, Gecko for Firefox, WebKit for Safari and Google Chrome

5. RELATED WORK

Related work broadly falls in the following three areas. *Languages aimed at manipulating visual information*. Kong et al. presented [13] a formalism that uses the visual appearance of Web pages in order to extract relevant information from them. Such a formalism, named spatial graph grammars (SGGs), extends graph grammars by incorporating spatial notions into language constructs. SGG productions are used to describe parts of Web pages by a graph representation. The representation includes spatial relations of the following types: direction relations that describe an order in the space, topological relations that express neighborhood and incidence, and distance relations such as near and far. The system allows for extracting page contents of interest for the user, but it is quite complex in term of both usability and efficiency. *Web wrapper induction exploiting visual interfaces*. Sahuguet et al presented W4F system [22] that uses a SQL-like language named HEL. In W4F parts of the query in HEL can be generated using a visual wizard which returns the full XPath location path of DOM nodes. So the HEL language is unable to recognize when information in a Web page is presented by the same visual pattern but is represented by different tag structures. So, the W4F system may benefit from using SXPath as more expressive basis for defining extraction rules. *Query languages for multimedia database and presentation*. Lee et al. [14] and Adah et al. [2] represent multimedia and presentation objects as direct acyclic graphs with spatiotemporal relations. They adopt algebraic language augmented with ad hoc operators for querying spatial relations among objects. Unlike SXPath, these languages are too complex and not suitable for querying Web pages.

6. CONCLUSION AND FUTURE WORK

In this paper we have extended XPath to include spatial navigation into the query mechanism. We have used spatial algebras to define new navigational primitives and mapped them for query evaluation onto an extension of the XML document object model (DOM), i.e. the SDOM. Thus, we have given a formal model of the extended query language and have evaluated theoretical complexity. The theory has been implemented in a SXPath tool. Empirical evaluation has been performed for testing its performances and functionality. Results on representative real Web pages have evidenced practical applicability of SXPath. The language can still be handled efficiently, yet it is easier to use and allows for more general queries than pure XPath. The exploitation of spatial relations among data items perceived from the visual rendering allows for shifting parts of the information extraction problem from low level internal tag structures to the more abstract levels of visual patterns.

The SXPath query language that we have defined will be a stepping stone for our future work on extracting information from Web pages. We argue that SXPath could improve both human and automatic capabilities in querying and extracting information. For instance, the popular LIXTO system [10], which allows users to visually define Web wrappers, uses XPath patterns and depends on internal HTML tag structure. It could exploit SXPath in order to leverage visual patterns as desired by authors of [5].

7. REFERENCES

[1] Acid Tests, http://www.acidtests.org. *Web Standards Project*.

[2] S. Adali, M. L. Sapino, and V. S. Subrahmanian. An algebra for creating and querying multimedia presentations. *Multimedia Syst.*, 8(3):212–230, 2000.

[3] J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[4] P. Balbiani, J.-F. Condotta, and L. F. d. Cerro. A new tractable subclass of the rectangle algebra. In *IJCAI*, pages 442–447, 1999.

[5] R. Baumgartner, G. Gottlob, and M. Herzog. Scalable web data extraction for online market intelligence. *VLDB*, 2(2):1512–1523, 2009.

[6] M. Benedikt and C. Koch. Xpath leashed. *ACM Computational Survey*, 41(1):1–54, 2008.

[7] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A survey of web information extraction systems. *TKDE*, 18(10):1411–1428, 2006.

[8] P. Eades and K. Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4):424–437, 1990.

[9] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak. Towards domain-independent information extraction from web tables. In *WWW*, pages 71–80, 2007.

[10] G. Gottlob, C. Koch, R. Baumgartner, M. Herzog, and S. Flesca. The lixto data extraction project: back and forth between theory and practice. In *PODS*, pages 1–12, 2004.

[11] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing xpath queries. In *VLDB*, pages 95–106, 2002.

[12] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing xpath queries. *TODS*, 30(2):444–491, 2005.

[13] J. Kong, K. Zhang, and X. Zeng. Spatial graph grammars for graphical user interfaces. *TOCHI*, 13(2):268–307, 2006.

[14] T. Lee, L. Sheng, T. Bozkaya, N. H. Balkir, Z. M. Özsoyoglu, and G. Özsoyoglu. Querying multimedia presentations based on content. *TKDE*, 11(3):361–385, 1999.

[15] L. Libkin. *Elements Of Finite Model Theory*. SpringerVerlag, 2004.

[16] J. Madhavan, S. R. Jeffery, S. Cohen, X. . Dong, D. Ko, C. Yu, A. Halevy, and G. Inc. Web-scale data integration: You can only afford to pay as you go. In *CIDR*, 2007.

[17] M. Marx and M. de Rijke. Semantic characterizations of navigational xpath. *SIGMOD Rec.*, 34(2):41–46, 2005.

[18] I. Navarrete and G. Sciavicco. Spatial reasoning with rectangular cardinal direction relations. In *ECAI*, pages 1–9, 2006.

[19] N. K. Papadakis, D. Skoutas, K. Raftopoulos, and T. A. Varvarigou. Stavies: A system for information extraction from unknown web data sources through automatic web wrapper generation using clustering techniques. *TKDE*, 17(12):1638–1652, 2005.

[20] P. Parys. Xpath evaluation in linear time with polynomial combined complexity. In *PODS*, pages 55–64. ACM, 2009.

[21] J. Renz. *Qualitative spatial reasoning with topological information*. Springer, 2002.

[22] A. Sahuguet and F. Azavant. Building intelligent web applications using lightweight wrappers. *DKE*, 36(3):283–316, 2001.

[23] B. ten Cate and M. Marx. Axiomatizing the logical core of xpath 2.0. *Theory of Computing Systems*, 44(4):561–589, 2009.

[24] W3C, http://www.w3.org/XML/Query/. *XML Query (XQuery)*, 1.0 edition.

[25] W3C, http://www.w3.org/TR/xpath. *XML Path Language (XPath) Version 1.0*, 1.0 edition, November 1999.

[26] P. Wadler. Two semantics for xpath. Draft: http://homepages .inf.ed.ac.uk/~wadler/papers/xpath-semantics, 2000.

[27] Y. Zhai and B. Liu. Structured data extraction from the web based on partial tree alignment. *TKDE*, 18(12):1614–1628, 2006.

APPENDIX

A. SPATIAL REASONING MODELS

The rectangle algebra model (RA) [4] has been obtained by extending Allen's temporal interval algebra (IA) [3] to the 2-dimensional case. IA models the spatial position between two segments and it is based on 13 atomic spatial relations $R_{int} = \{b, m, o, s, d, f, bi, mi, oi, si, di, fi, e\}$, see Tab. 4. For instance, the relation $s\ b\ s_1$ means that the segment s is preceded by the segment s_1.

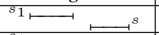
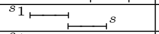
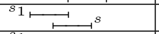
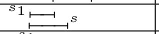
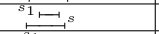
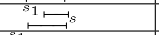
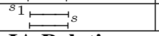
Relation	Symbol	Meaning	Inverse
before	b		bi
meets	m		mi
overlaps	o		oi
starts	s		si
during	d		di
finish	f		fi
equals	e		e

Table 4: IA Relations.

The set of possible atomic and conjuntive RA relations is called R_{rec}. It is obtained by combining R_{int} relations on x and y-axes. Thus, $R_{rec} = R^2_{int}$ contains $13^2 = 169$ elements. Figure 7 shows that RA relations are invertible and represents a pictorial representation of the Mapping Function μ (Def. 7).

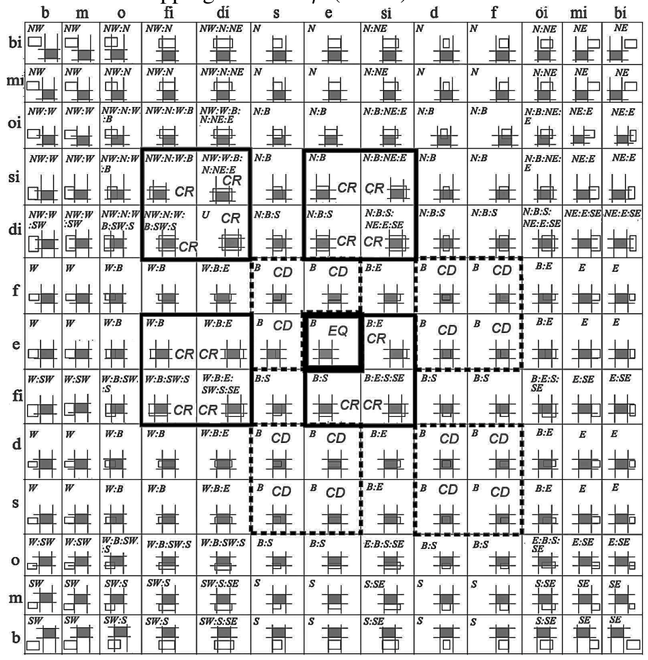


Figure 7: Pictorial Representation of the RA relations

B. PROOFS

B.1 Proof of Theorem 1

In this section we prove that Core SXPath queries Q can be evaluated in $O(|D|^2 * |Q|)$ combined complexity (Theorem 1), where D is the XML document. In order to prove the Theorem 1 we first define: (i) the *axis function* by extending the definition presented in [11] to both traditional and spatial axes; (ii) the *RA function*; (iii) the algorithm which compute the axis function.

Definition 17. Let χ denote an axis in Δ. The related *axis function*, which overloads the relation name $\chi : 2^V \to 2^V$ is defined as $\chi(\Gamma) := \{u | \exists c \in \Gamma : c\chi u\}$, where $\Gamma \subseteq V$ is a set of nodes. Moreover, the inverse axis function $\chi^{-1} : 2^V \to 2^V$ is defined as $\chi^{-1}(\Gamma) := \{c \in V | \chi(\{c\}) \cap \Gamma \neq \varnothing\}$.

Definition 18. Let ρ denote a RA relation in R_{rec}. The related *RA function* $f_\rho : V_v \to 2^{V_v}$ is defined as $f_\rho(u) := \{v \mid u\rho v \wedge u, v \in V_v \wedge \rho \in R_{rec}\}$

The function f_ρ works on the SDOM (Def. 2) represented as a nested direct-access data structure. Such data structure for SDOM can be computed in a preprocessing step, which runs in $O(|V_v|^2)$ and requires $O(|V_v|^2)$ more space than a standard DOM. It is noteworthy that f_ρ takes as input a single node $u \in V_v$, accesses in constant time the hash set representing the SDOM, and returns the node set associated to u by the RA relation ρ in constant time.

Lemma 1. Let ρ be an RA relation. For each pair of nodes $u, v \in V_v$, $u\ \rho\ v$ iff $v\ \rho^{-1}\ u$.

Let $\Gamma \subseteq V$ be a set of nodes of a SDOM, let $\chi_t \in \Delta_t$ be a traditional axis other than *self*, let $\chi_s \in \Delta_s$ be a spatial axis, let ϱ be a set of RA relations, and let $E(\chi_t)$ denote a regular expression based on the primitives *firstchild* and *nextsibling* as presented in [11, 12] that defines $\chi_t \in \Delta_t$.

Algorithm 1. (*Axis evaluation*)

```
Input: A set of nodes $\Gamma$ and an axis $\chi \in \Delta$
Output: $\chi(\Gamma)$
Method: $eval_\chi(\Gamma)$
(1.1)  function $eval_{self}(\Gamma) := \Gamma$.
(1.2)  function $eval_{\chi_t}(\Gamma) := eval_{E(\chi_t)}(\Gamma)$.
(1.3)  function $eval_{\chi_s}(\Gamma) := eval_{\{\rho_i | \rho_i \in \mu(\chi_s)\}}(\Gamma)$.
(1.4)  function $eval_{\chi_s^{-1}}(\Gamma) := eval_{\{\rho_i^{-1} | \rho_i \in \mu(\chi_s)\}}(\Gamma)$.
(1.5)  function $eval_\varrho(\Gamma)$ begin
(1.6)    $\Gamma' := \varnothing$;
(1.7)    foreach $u \in \Gamma \cap u \in V_v$ do
(1.8)        foreach $\rho_i \in \varrho$ do
(1.9)            $\Gamma' := \Gamma' \cup_{set} f_{\rho_i}(u)$ od od
(1.10) return $\Gamma'$ end.
```

The Algorithm 1 computes the set of nodes reached from a set of nodes Γ by means of an axis χ. For traditional axes in (1.1) and (1.2) the algorithm requires time $O(|V|)$ [11]. Whereas, for spatial axis in (1.3) and (1.4), we have to consider the mapping function μ that runs in constant time (see Def. 7) and returns a set of atomic/conjunctive RA relations ϱ representing the axis given as input. The function $eval_\varrho$ (1.5)-(1.10), for each input node u and for each RA relation $\rho \in \varrho$ takes the reached nodes using the function f_ρ. The union of the resulting set of nodes (1.9) runs in time $O(|V_v|)$, hence the total time required is $O(|V|^2)$. Now, we are ready to prove the theorem 1.

PROOF. In [11] it is was shown that Core XPath 1.0 fragment have linear combined complexity by mapping its queries to an algebraic expression. Likewise for Core XPath 1.0 and its extension, every query that falls in the Core SXPath can be mapped in time $O(|Q|)$ to an algebraic expression Φ over the set of operations $\cap, \cup, \neg, \chi$ (the axis function) and its inverse χ^{-1} and $\frac{V}{root}(\Gamma) := \{x \in V | root \in \Gamma\}$ (that is, returns V if $root \in \Gamma$ and $\varnothing$ otherwise). In the query Q there are at most $O(|Q|)$ of such operations. Each operation in our algebra can be computed in time $O(|V|)$ except for the axis function that is computable in $O(|V_v|^2)$ time bound in the spatial case as shown in Algorithm 1. Hence, the computation has time bound $O(|V|^2 * |Q|)$. □

B.2 Proof of Theorem 2 and 3

In this section we prove first that Full SXPath can be evaluated in time $O(|D|^4 * |Q|^2)$ and space $O(|D|^2 * |Q|^2)$, where D is the XML document, and Q is a *Full SXPath* query (Theorem 3). Then we exploit such proof in order to prove that queries falling in the *SWF* can be evaluated in better time than full SXPath queries.

In order to obtain a polynomial-time combined complexity bound for Full SXPath query evaluation we use dynamic programming adopting the *Context-Value Table* (CV-Table) principle proposed in [11]. Given an expression e belonging to an input query, the CV-Table of e specifies which value is obtained given a valid context $\vec{c}$: $(\vec{c}, \varepsilon[\![e]\!](\vec{c}))$. The CV-Table of each expression is obtained combining the values of its subexpressions. Moreover, we adopt the simple idea [26] that for evaluating each expression just the necessary information of the context (*relevant context*) can be taken into account. The relevant contexts of any expression e associated to a node q of Q can be computed in a preprocessing step as follows: (i) If q is leaf node of the query parse tree, the relevant context depends on e. If e is a constant, then its evaluation does not depends on the input context and thus the relevant context is empty. If the expression is a location path or a positional function, then the relevant context corresponds to the i^{th} value of the context[5] that defines the semantics of the expression as defined in Def. 16. (ii) Otherwise, if e is a location path, then the relevant context of q is the context node. In the other cases, the relevant context of q is given by the union of the relevant context of children parse tree nodes $(q_1, ..., q_k)$ of q: $RelevContext(q) := \cup_{i=1}^{k} RelevContext(q_i)$.

The CV-Table principle avoids exponential time complexity because it guarantees that no evaluation of the same subexpression for the same context is done more than once. So it allows simultaneous evaluation for all possible contexts (node). As in [12] where position and size are computed on demand, we compute all spatial position functions in a loop for all pairs of previous\current nodes. For evalauting SXPath location steps we use the *min context algorithm* presented in [12] with the substantial difference being the computation of location step and spatial position functions. The complete context will be needed only for predicates of location steps that use spatial position functions. In the following, for lack of space, we describe only the most costly part of the *location step evaluation algorithm* that computes complete contexts.

Algorithm 2. (*Location step evaluation algorithm*)

```
**Input**: A set of nodes $\Gamma$ and a location step $e = \chi :: \tau[e_1]\dots[e_m]$
**Output**: $P[\![e]\!](\Gamma)$
**Method**: $eval(e, \Gamma)$ **begin**
(2.1) $Res := \varnothing$
(2.2) $W := \chi(\Gamma) \cap T(\tau)$;
(2.3) for each $u \in \Gamma$ do
(2.4)     $W' := \{w \mid w \in W \wedge u \; \chi \; w\}$
(2.5)     for each $e_i$ with $1 \leqslant i \leqslant m$ (in ascending order) do
(2.6)         $\vec{W} := layering(W')$
(2.7)         $W' := \{w \mid w \in \vec{W} \wedge \varepsilon[\![e_i]\!](\vec{c_w}) = true \wedge$
                  $\vec{c_w} := \langle w, idx_{\chi}(w, \vec{W}), |\vec{W}|, pidx_{\leqslant_{\uparrow}}(w, \vec{W}), plast_{\leqslant_{\uparrow}}(\vec{W}),$
              $pidx_{\leqslant_{\rightarrow}}(w, \vec{W}), plast_{\leqslant_{\rightarrow}}(\vec{W}), pidx_{\leqslant_{\downarrow}}(w, \vec{W}), plast_{\leqslant_{\downarrow}}(\vec{W}),$
              $pidx_{\leqslant_{\leftarrow}}(w, \vec{W}), plast_{\leqslant_{\leftarrow}}(\vec{W}), pidx_{\leqslant_{t}}(w, \vec{W})\rangle\}$
          od
(2.8)     $Res := Res \cup W'$
      od
(2.9) return $Res$ end;
```

The Algorithm 2 corresponds to the semantic definition of location step presented in Def. 15. But it considers only the most expensive case that requires the computation of the complete context. Such a case is sufficient for proving the combined complexity of Full SXPath. We need to compute complete context when all expressions $e_1 \dots e_m$, in the input location step, require the evaluation of positions w.r.t. document and spatial orders. As detailed in [12], when expressions do not require evaluation of positions we can pre-compute the CV-Tables because the relevant context is empty or corresponds to a single node. In the instruction (2.2) we obtain the nodes reachable via $\chi :: \tau$. In (2.3)-(2.7) we select nodes that satisfy the predicates $e_1 \dots e_m$. We have to compute the complete context $\vec{c}$ corresponding to any pair of previous and current nodes u and w, respectively. The instruction (2.7) builds the context $\vec{c}$ that considers the position of the node w in $\vec{W}$ w.r.t. the document order and all spatial orders (see Def. 13). For obtaining spatial ordering we need to apply the $layering$ function (instruction (2.6)) to $\vec{W}$ (i.e. the set of nodes reached from u). Such a function assigns to each node a layer in order to allow the functions $pidx_{\leqslant_z}(u, \Gamma)$ and $plast_{\leqslant_z}(u, \Gamma)$ (see Def. 6) for computing spatial position of w in $\vec{W}$. The layering for each spatial order (directional or topological) can be obtained by applying a topological layering algorithm [8]. For lack of space and because the layering for the other orders can be obtained in similar way, only the layering for the topological order is shown in the following.

Definition 19. Let $\Gamma \subseteq V_v$ be a set of SDOM nodes, the *topological directed acyclic graph* (TDAG) $G_t = (\Gamma, A_t)$ is the graph obtained from the SDOM by considering RA relations that express containment among nodes. So for each pair of nodes n, n' in Γ, an arc is added to A_t iff n' is spatially contained in n.

Definition 20. Let $\Gamma \subseteq V_v$ be a set of SDOM nodes, and let $G_t = (\Gamma, A_t)$ be the corresponding TDAG, the *topological layering* $L_t(G_t) = \{l_1, \cdots, l_{h_t}\}$ (Def. 5) is obtained applying to G_t a topological layering algorithm [8].

Layering algorithm runs in time $O(|\Gamma| + |A_t|)$ by using appropriate data structures with minor modifications to the standard topological sorting algorithm. Topological layers allow for defining a topological ordering among nodes in Γ based on their spatial nesting. So, for example, the first layer represents nodes in Γ that are not contained in other nodes. The second layer represents nodes that are directly contained in nodes in the first layer at first level of nesting, and so on. The layering for each directional order can be obtained also by using optimized methods that work on a prelayered version of the SDOM, not explained here for lack of space.

Having obtained the complete context $\vec{c}$, the instruction (2.7) allows for computing the set of nodes W' reached from the current node u and that satisfy the predicates $e_1 \dots e_i$. For the current node w, the value of $\varepsilon[\![e_i]\!](\vec{c_w})$ is looked up from the table if $\vec{c_w}$ exists in the CV-Table of e_i, computed otherwise. The resulting nodeset (instruction (2.9)) is the union of nodes reached from each current node u in Γ and satisfying $e_1 \dots e_m$ predicates. Now, we are ready to prove the Theorem 3.

PROOF. *Space complexity.* In the preprocessing step we create the SDOM structure, then in order to save the spatial relations $O(|V_v|^2)$ additional space is required w.r.t. the XML document. During the query computation, we know that an input Full SXPath query Q has at most $|Q|$ subexpressions, thus at most $|Q|$ CV-Tables are required. We explicitly set up the CV-table for a subexpression e only if the relevant context of e corresponds to a node (i.e. $\{c[1]\}$), or to the empty set. Hence, the CV-Table has at most $|V|$ rows ($O(|D|)$). Moreover, since Full SXPath and Full XPath 1.0 have the same set of operations and return the same result types, then the most costly operation w.r.t. space size is concatenation on string [12]. We have $O(|D| * |Q|)$ maximal size for one entry value in the CV-Table. Hence we obtain the bound $O(|D|^2 * |Q|^2)$.
Time complexity. The SDOM computation, which costs $O(|V|^2)$, precedes the query evaluation step. An SXPath expression Q has at most $|Q|$ subexpressions that have to be evaluated. Each subexpression e of the input query Q has to be evaluated for at most $O(|V|^2)$ different contexts that can be computed in a loop over all possible values $\vec{c}$ corresponding to pairs previous/current of context-nodes (see algorithm 2). Moreover, it was shown in [12] that time required for computing each XPath operation on any context $\vec{c}$ is bounded by $O(|D|^2 * |Q|)$. In our case, we add only spatial relations and the position functions. The former ones, given in input only one context $\vec{c}$ can be computed in time $O(|V_v|)$ (In fact, in this case the method $eval_{\varrho}$ of the algorithm 1, has to run only for a node u (see instruction (1.7)). Given a context $\vec{c}$, a positional function returns the value corresponding to the i^{th} value of the context that defines the semantics of the expression (see Def. 16). As shown in Algorithm 2 topological layering is needed (see instruction (2.2) in the

[5]Given a context $\vec{c}$ then $\vec{c}[i]$ represents the i^{th} value of the context (e.g., $\vec{c}[3]$ represents the context size w.r.t. the document order).

Algorithm 2) for computing the complete context, but this operation does not impact the worst case bound. Hence we have the time combined complexity $O(|D|^4 * |Q|^2)$. $\square$

In order to prove that queries falling in SWF can be evaluated in better time than full SXPath queries (Theorem 2), we: (i) Consider the syntax defined by the grammar presented in Def. 12. (ii) Adopt restrictions described in [12] for EWF. Such restrictions imply that functions which select data from the XML document, and boolean expressions that compare node sets are not allowed. Furthermore, boolean expressions `locpath relop sexpr` or `locpath relop number` (see Def. 12) consider only numbers and strings which size do not depend on the XML document (i.e. are values fixed in the query). (iii) Adopt the bottom-up/top-down query evaluation strategy proposed in [12]. Such an evaluation strategy distinguish between *outer* and *inner location paths*, where an inner path appears within a predicate, whereas an outer path does not.

For showing computational complexity of SWF we describe how the node set resulting for a SWF query can be computed by exploiting the Algorithm 2. In particular, outer location paths are computed by the Algorithm 2 as it is (top-down and forward evaluation). Whereas, inner location paths are bottom-up and backward computed. In this case the evaluation algorithm starts from the last location step in a inner location path, and takes as initial input node set Γ either all nodes in V (when the considered boolean expression coincides with the inner location path itself), or the set of nodes that satisfy allowed boolean expressions. Then evaluation algorithm computes node sets as in the Algorithm 2 where: in instruction (2.2) the axis is χ^{-1} instead of χ (backward evaluation), instruction (2.3) is for each previous $u \in W$, and in instruction (2.4) current node w is in Γ. Having in mind above discussion we can prove the Theorem 2.

PROOF. *Time complexity.* An SWF query Q has at most $|Q|$ subexpressions that have to be evaluated. Such subexpressions are computed by the Algorithm 2 used as described in the above discussion for enabling both top-down and bottom-up evaluation. Location path evaluation is performed in time $O(|V|^2)$ in instruction (2.2) of the Algorithm 2 (see Algorithm 1). For any expression e, the computation of the result value of e for a single context $\vec{c}$ takes at most $O(|Q|)$ time (see restrictions presented above). Furthermore, we have $O(|V|^2)$ contexts (instructions (2.3) and (2.7)) and at most $|Q|$ sub expressions, hence the total time required by each expression e of Q is limited by $O(|V|^2 * |Q|)$. However, let u be the node under evaluation, we have to compute the layering for reached nodes in order to obtain the complete context (instruction (2.6) in the Algorithm 2). The layering method costs $O(|V|^2)$ and has to be computed for reach node under evaluation. So, we need $O(|V|^3)$ time for computing all contexts (see operations (2.3)-(2.6) in Algorithm 2). Hence, the total computational complexity is bounded by $O(max(|V|^3 * |Q|, |V|^2 * |Q|^2))$. Since normally, $V >> Q$ then, the complexity bound follows. $\square$

B.3 Complexity Comparison

The complexity results are summarized in Table B.3. These results are compared with the fragment of XPath 1.0 that they extend.

	XPath 1.0		**SXPath**	
Space	Core[11]	$O(\|D\| * \|Q\|)$	Spatial	$O(\|D\|^2 * \|Q\|)$
Time		$O(\|D\| * \|Q\|)$	Core	$O(\|D\|^2 * \|Q\|)$
Space	EWF[12]	$O(\|D\| * \|Q\|^2)$	SWF	$O(\|D\|^2 * \|Q\|^2)$
Time		$O(\|D\|^2 * \|Q\|^2)$		$O(max(\|D\|^3 * \|Q\|, \|D\|^2 * \|Q\|^2))$
Space	Full[11]	$O(\|D\|^2 * \|Q\|^2)$	Full	$O(\|D\|^2 * \|Q\|^2)$
Time	Xpath 1.0	$O(\|D\|^4 * \|Q\|^2)$	SXPath	$O(\|D\|^4 * \|Q\|^2)$

Table 5: Comparison between complexity bound of SXPath and XPath 1.0 for a XML document D and a query Q.

C. EXPERIMENT RESULTS

C.1 System Efficiency

For testing system efficiency we have considered: (i) a dummy Web page that presents a table with 3 columns and an increasing number of rows; (ii) 3 types of queries, falling in the SWF, based on: (a) traditional axes and position functions, (b) spatial axes (both directional and topological) and spatial position functions; (c) a mix of traditional and spatial features. Queries were constructed as follows. For query types (a) and (c), the first query was given by: "`/descendant::TD/following-sibling::TD[1]`". For query type (b), the first query was given by: "`/CD::TD/E::TD[W,1]`". The $(i+1)$-th query was constructed by appending the following location paths to the i-th query: (a) "`/following-sibling::TD[1] /preceding-sibling::TD[1]`"; (b) "`/E::TD[W,1]/W::TD[E, 1]`", and (c) "`/following-sibling::TD[1]/W::TD[E,1]`". For instance, the third query ($i = 2$) for the spatial query type (b) was "`/CD::TD/E::TD[W,1]/E::TD[W,1]/W::TD[E,1]/E::TD[W, 1]/W::TD[E,1]`". All queries aim at extracting the central column of the table in the page. They are based on opposite axes (i.e. "at-east" and "at-west"), so they redundantly jump back and forward within the input documents. Our rationale was that by this way the query processor must handle many different spatial paths in parallel coping thus with an intuitively "difficult" query. Figure 8 and 10 show that time needed for query evaluation is linear w.r.t. the document size and the query size respectively (curves are all straight lines in log log scale and curves slopes are all 1). Figure 9 shows the curve obtained for the SDOM construction. The obtained curve is a straight line in log log scale with slope equal to 2 indicating the polynomial complexity of the whole system.

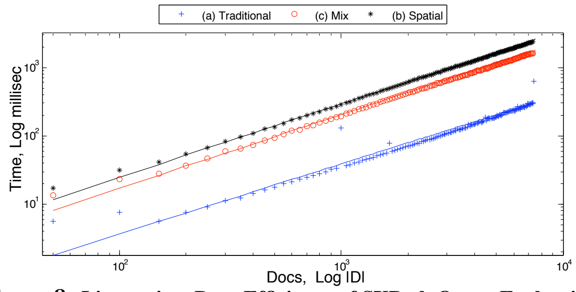


Figure 8: Linear-time Data Efficiency of SXPath Query Evaluation

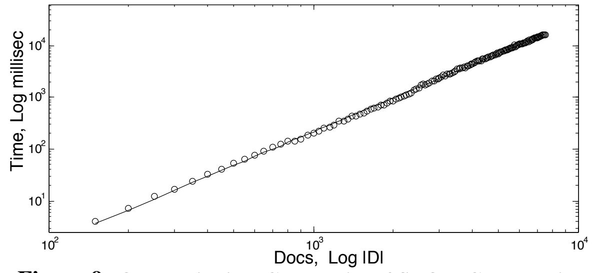


Figure 9: Quadratic-time Complexity of SDOM Construction

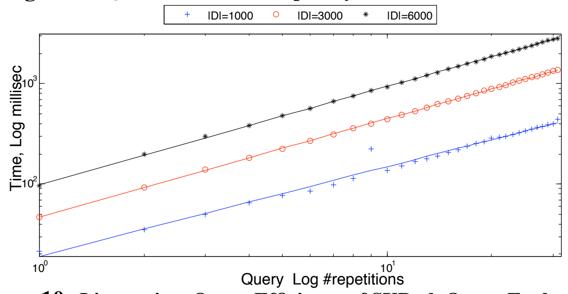


Figure 10: Linear-time Query Efficiency of SXPath Query Evaluation

C.2 Evaluation of Usability

Experiment 2. Tab. 6 reports results of the experiment aimed at assessing the effort needed for learning SXPath and the user feeling in applying the language. The table gives: (i) in column 1 the users identifiers; (ii) in column 2 the time needed for learning SXPath and manually writing the assigned query; (iii) in columns 3 and 4 answers provided by users to the questions"How easy is the SXPath language?" and "What is your level of satisfaction in using the SX-Path language?" respectively. Possible answers have been designed as the following seven-item Likert scale: very easy/satisfactory (3), easy/satisfactory (2), quite easy/satisfactory (1), neutral (0), quite difficult/unsatisfactory (-1), difficult/unsati-sfactory (-2), very difficult/unsatisfactory (-3); (iv) in columns 5 and 6 the number of attempts that each user needed for writing spatial location paths for names and prices respectively; (v) in the last two rows the average values and the standard deviations for all observed values.

#user	Time (min)	Easiness/ Difficulty	Satisfaction/ Unsatisfaction	#attempts	
				name	price
1	75	2	0	7	6
2	45	3	2	4	2
3	65	1	1	5	4
4	40	2	1	2	3
5	50	3	2	4	4
6	30	3	3	2	1
7	125	-1	-1	9	8
8	50	2	1	3	4
9	35	3	2	2	2
10	55	2	1	5	2
Average	**57**	**2**	**1.2**	**4.3**	**3.6**
σ	**26**	**1.18**	**1.1**	**2.2**	**2**

Table 6: Evaluation of the Effort Needed for Learn and Apply SXPath

Experiment 3. Table 7 reports results of the experiment aimed at finding out whether it is easier to specify an SXPath query than an XPath query given only the rendered Web page, but no information about the internal structure. Tab. 7 gives: (i) the average number of pairs (product names and prices) correctly extracted ("Cr.") using SXPath and XPath in columns 2 and 5 respectively; (ii) in columns 3 and 6 the corresponding average number of pairs wrongly (Wr.) extracted (false positive – FP/ false negative – FN); (iii) in columns 4 and 7, the average number of attempts (the maximal was fixed to 5) performed by users for obtaining the most accurate results; (iv) in the last two rows the average recall and precision respectively.

Experiment 4. In this experiment we have evaluated the effort needed by users for obtaining sound and complete SXPath and pure XPath queries, giving to users the possibility to look at both DOM/SDOM and visualized Web page. Tab. 7 gives: (i) in columns 8 and 10 the average number of attempts needed by all users for writing sound and complete XPath and SXPath queries respectively. For SXPath queries already sound and complete in Experiment 3, we have reported in column 8 the value of attempts already observed in column 4. (ii) in columns 9 and 11 the average number of location steps in SXPath and pure XPath queries respectively. These numbers has been computed as the average between the number of location steps in location paths that identify product names and prices; (iii) in columns 12 and 13 the average number of further attempts and the average number of location steps needed for expressing pure relative XPath queries respectively. All users have noticed that for some Web sites (highlighted by "⋄" in Tab. 7) it has been very difficult to find a sound and complete pure XPath query because of the very intricate DOM structures that make necessary long disjunctions of location paths. Such difficulties are due to the fact that data records in these Web sites are contained in discontinuous pieces of the DOM (e.g. www.amazon.com), and that the tag structure representing a given data item can be different from a record to another (e.g. www.ebay.com), even though records have the same spatial arrangement in all selected pages.

Deep Web Sites	**Querying Without DOM/SDOM**						**Querying With DOM/SDOM**					
	SXPath			**XPath**			**SXPath**		**Abs. XPath**		**Rel. XPath**	
	Cr.	**Wr.**	**Att.**	**Cr.**	**Wr.**	**Att.**	**Att.**	**Steps**	**Att.**	**Steps**	**Att.**	**Steps**
amazon.com ⊙⋄	58	0/0	2.5	43	32.5/15	5	2.5	5.5	6.5	24	13	10
bestbuy.com †	68	0/2	2.2	70	825/0	5	4.2	4.5	3	12.5	2.5	6
bigtray.com ⊙⋄	125	5/0	2	125	130/0	5	4	4.5	3	10	6	7
bol.de †	60	0/0	2	60	15/0	5	2	4.5	3	16.5	3.5	6.5
buy.com †	100	2/0	3.2	100	30/0	5	4.2	5.5	3	19	4.5	8
ebay.com ⋄	258	0/0	3	258	60/0	5	3	5.5	4.5	21	9	7
mediaworld.it †⊙	125	0/0	2	125	30/0	5	2	5	3.5	21.5	2	5
shopzilla.co.uk †⊙⋄	100	0/0	1.5	100	937/0	5	1.5	4	6.5	23	4	9
apple.com †⊙	50	0/0	1.4	50	0/0	5	1.4	4	4	14	2	4
venere.com †⊙	75	0/0	3.2	75	75/0	5	3.2	4.5	3	9	2.5	4
powells.com ⊙	125	0/0	1.5	125	247.5/0	5	1.5	5	3	11.5	2	4
barnesandnoble.com †⊙	50	0/0	2.3	50	255/0	5	2.3	4.5	3	15.5	2.5	7
shopping.yahoo.com †⊙	75	0/0	1.7	75	187.5/0	5	1.7	4	3	21	3	5
cooking.com † ⋄	100	7.5/0	3.2	100	180/0	5	7.2	7.5	8	36	6	9
cameraworld.com †⊙	125	0/0	2	125	10/0	5	2	5.5	3	11.5	2	4.5
drugstore.com † ⋄	45	0/4	1.5	41	5/0	5	5.5	8	6.5	16.5	2.5	4.5
magazineoutlet.com †⊙	45	0/0	1	45	125/0	5	1	5	3	21.5	5	9
dealtime.com †⊙	150	0/0	1	150	20/0	5	1	5.5	3	16	3.5	7
borders.com †⊙	125	0/0	1.6	125	40/0	5	1.6	6	3.5	18	4	6
google.com/products ⊙	50	0/0	1.5	50	130/0	5	1.5	5.5	3.5	9.5	2.5	5.5
nothingbutsoftware.com †⊙	80	0.8/0	2.2	80	55/0	5	4.2	6.5	3.5	28	3	8
abt.com †⊙⋄	200	5/0	1.3	200	0/0	5	2.3	6.5	4.5	27	4	6
cutleryandmore.com †⊙⋄	150	1/0	1.5	150	68/0	5	2.5	5	10	36	4	10
cnet.com †⊙⋄	150	0/0	2	130	0/20	5	2	4	4	17.5	5.5	8
target.com †	50	0/0	3	50	2/0	5	3	5.5	3.5	16.5	2.5	5
Average			**2**			**5**	**2.7**	**5.3**	**4.2**	**18.9**	**4**	**6.6**
Total	**2535**	**27.3/6**		**2506**	**3459.5/35**							
Recall	**100%**			**99%**								
Precision	**99%**			**42%**								

Table 7: Usability Evaluation of SXPath on Deep Web Pages.

Experiment 5. In this experiment we aimed to qualitatively evaluate whether SXPath queries are more general and abstract than XPath queries given different Web pages and the same extraction task. Firstly, we have considered the subset of Web sites in the dataset that show the same visual pattern for product names. Each record is represented by the product image that has at east more than one product attribute, the first attribute from north is the product name. Such Web sites are highlighted by "⊙" in column 1 of Tab. 7. We have observed that the spatial location path `/CD::img[GS|GN::img][GE::*[W,1][N,2][self::text]]/GE::text[W,1][N,1]` is able to identify product names in all these Web sites in a sound and complete way. In contrast, pure XPath location paths for the same subset of sites were all completely different and no reuse of code was possible. Secondly, we have asked users to extract lists of friends in social networks listed in column 1 of Tab. 8. The table, also, gives: (i) in column 2 sound and complete queries defined by user #6 without looking at the (S)DOM; (ii) in column 3 the pure XPath location paths that allow for extracting friend names produced by the user looking at the DOM. The variety of location paths produced by users, looking at the DOM, and listed in Tab. 8 indicates the large heterogeneity of internal tag structures observed for different social networks.

Social Networks	**Queries**	
	SXPath	Pure XPath
facebook	`/CD::text()[.="Amici"]/ CR::*[CR,1]/CD::img/GS::*[N,1]`	`//div[@id='profile_friends_box_ inner_content']/div//div/div/div/a`
youtube	`/CD::text()[.="Amici"]/ CR::*[CR,2]/CD::img/GS::*[N,1]`	`//div[@id='user_friends']// div[1]/div/center/a`
netlog	`/CD::text()[.="Amici"]/ CR::*[CR,2]/CD::img/GS::*[N,1]`	`//div[@id='nicknameFriends']/ div/div/a[2]`
care	`/CD::text()[.="Friends"]/ CR::*[CR,1]/CD::img/GS::*[N,1]`	`//td[@id='col_right']// table[1]/tbody/tr[1]/td/a[2]`
bebo	`/CD::text()[.="Friends"]/ CR::*[CR,2]/CD::img/GS::*[N,1]`	`//div[@id='content_Friend']/ ul[2]/li/span[2]/a`

Table 8: Generality of SXPath Queries on Social Network Sites

Acknowledgements

Work done by Steffen Staab was partially funded by German National Science Foundation (DFG) in the project 'Multipla'. Work done by Ermelinda Oro and Massimo Ruffolo was partially funded by Regione Calabria in the projects 'Voucher Ricercatori' and 'Easy-PA - ID: 1220000277'.