# On Supporting Effective Web Extraction

Wook-Shin Han [#], Wooseong Kwak [#], Hwanjo Yu [*]

[#]*Department of Computer Engineering, Kyungpook National University, Korea*
[*]*Department of Computer Science and Engineering, POSTECH, Korea*

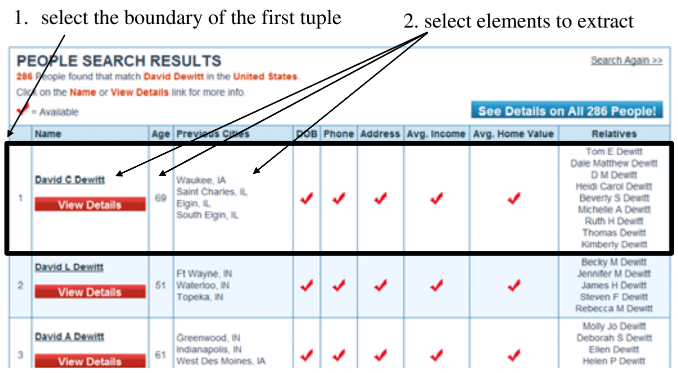
***Abstract*— Commercial tuple extraction systems have enjoyed some success to extract tuples by regarding HTML pages as tree structures and exploiting XPath queries to find attributes of tuples in the HTML pages. However, such systems would be vulnerable to small changes on the web pages. In this paper, we propose a robust tuple extraction system which utilizes spatial relationships among elements rather than the XPath queries of the elements. Our system regards elements in the rendered page as spatial objects in the 2-D space and executes spatial joins to extract target elements. Since humans also identify an element in a web page by its relative spatial location, our system extracting elements by their spatial relationships could possibly be as robust as manual extraction and is far more robust than existing tuple extraction systems.**

## I. Introduction

Extracting tuples from HTML pages has been an important issue [1], [2], [3], [4], [5] in various web applications such as web data integration and mashups that repurpose and selectively combine existing web data and services [5]. Commercial tuple extraction systems (a.k.a. wrapper generators) extract tuples by regarding HTML pages as tree structures and exploiting XPath queries (or regular expressions) to find attributes of tuples in the HTML pages. In such systems, given a sample HTML page $T$, the user first defines a target schema $S$ for the tuples to extract and associates XPath queries of the HTML elements (corresponding to the attributes of the tuples) in $T$ to target elements of $S$. The systems store these associations as mapping rules for extracting tuples. Thus, these rules can be applied to all pages similar to $T$.

Commercial tuple extraction systems typically require two steps to extract tuples from a given web page: 1) selecting the boundary element of the first tuple to extract (assuming the remaining tuples can be extracted by changing the position information of the boundary element) and 2) selecting the corresponding tag for each attribute of the tuple.

Figure 1 shows the conventional tuple extraction process in detail. Suppose a user wants to extract names, ages, and cities of the people whose names are "David Dewitt" from the Yahoo's people search page. The user is first required to select the TR tag (boundary tag) of the *first* tuple. More specifically, an absolute XPath, /HTML/BASE/BODY/DIV[2]/TABLE /TBODY/TR[**3**], is used to locate the TR tag. Once the XPath for the first tuple is identified, the user needs to select tags for the attributes, i.e., names, ages, and cities. Specifically, three relative XPath queries from the TR tag—./TD[2]/DIV[1]/A[1]/B[1], ./TD[3], and ./TD[4]—are used to locate these attributes. The tuple extraction system will increase the index of the TR tag to extract from the second tuple. That is, /HTML/BASE/BODY/DIV[2]/TABLE /TBODY/TR[**4**] will be used for extracting the second tuple.


1. select the boundary of the first tuple
2. select elements to extract

PEOPLE SEARCH RESULTS

286 People found that match David Dewitt in the United States.

Click on the Name or View Details link for more info.

✓ = Available

Search Again >>

See Details on All 286 People!

| | Name | Age | Previous Cities | DOB | Phone | Address | Avg. Income | Avg. Home Value | Relatives |
|---|---|---|---|---|---|---|---|---|---|
| 1 | David C Dewitt<br>View Details | 69 | Waukee, IA<br>Saint Charles, IL<br>Elgin, IL<br>South Elgin, IL | ✓ | ✓ | ✓ | ✓ | ✓ | Tom E Dewitt<br>Dale Matthew Dewitt<br>D M Dewitt<br>Heidi Carol Dewitt<br>Beverly S Dewitt<br>Michelle A Dewitt<br>Ruth H Dewitt<br>Thomas Dewitt<br>Kimberly Dewitt |
| 2 | David L Dewitt<br>View Details | 51 | Ft Wayne, IN<br>Waterloo, IN<br>Topeka, IN | ✓ | ✓ | ✓ | ✓ | ✓ | Becky M Dewitt<br>Jennifer M Dewitt<br>James H Dewitt<br>Steven F Dewitt<br>Rebecca M Dewitt |
| 3 | David A Dewitt<br>View Details | 61 | Greenwood, IN<br>Indianapolis, IN<br>West Des Moines, IA | ✓ | ✓ | ✓ | ✓ | ✓ | Molly Jo Dewitt<br>Deborah S Dewitt<br>Ellen Dewitt<br>Helen P Dewitt |

Fig. 1. An example page from the Yahoo's people search.

Since absolute (or partial-match) XPath queries are used to extract tuples from web pages, the conventional extraction systems are vulnerable to small changes in the web pages. For example, if the email column is added just before the age column, the systems using either the absolute XPath or partial-match XPath could not extract ages and cities of the people. Thus, a completely different mechanism is needed to resolve these problems.

We propose a robust tuple extraction system that utilizes spatial relationships among elements rather than the XPath queries of the elements. Spatial information (e.g., 2-D coordinates) of elements can be obtained from the DOM tree when a web page is rendered in a browser. Our system regards elements in the rendered page as spatial objects in the 2-D space, and thus, executes spatial joins to extract target elements. Since humans also identify an element in a web page by its relative spatial location, our system extracting elements by their spatial relationships could possibly be as robust as manual extraction. To the best of our knowledge, this is the first research to leverage spatial join to extract tuples from web pages.

Our contributions are summarized as follows: 1) We propose the first framework that leverages spatial join for robust tuple extraction from web pages. 2) We propose a query language RAQuery that supports various matches as well as complex spatial joins among sets of HTML elements. 3) We develop spatial join algorithms that process the RAQuery.

The rest of this paper is organized as follows. Section II describes rectangle algebra and presents RAQuery based on rectangle algebra. Section III presents a spatial join tailored to tuple extraction. We conclude in Section IV.

## II. Query Language for Tuple Extraction

### A. Topological Relationship Using Rectangle Algebra

The *interval algebra* [6] is the most well-known model to identify a set of thirteen atomic relations for any two intervals in the 1-D space. That is, there are seven atomic relations: *before*, *meets*, *overlaps*, *starts*, *during*, *finishes*, and *equals* along with their seven inverse counterparts. The atomic relation *equals inverse* is excluded since it is the same as the atomic relation *equals*.

Balbiani et al. propose the *rectangle algebra* by extending the interval algebra to represent topological relationships between any two bounding boxes in the 2-D space [7]. For a bounding box $B$, we denote $B_x$ as the interval corresponding to the projection of $B$ onto the x-axis and $B_y$ as the interval corresponding to the projection of $B$ onto the y-axis. In rectangle algebra, an atomic relation between two bounding boxes $B$ and $B'$ is a pair ($R_x$, $R_y$) of atomic interval relations, where $R_x$ the atomic relation between $B_x$ and $B'_x$, and $R_y$ the atomic relation between $B_y$ and $B'_y$. Thus, 169 (=13 × 13) possible atomic relations exist to represent topological relationships between any two bounding boxes.

In Figure 1, consider that we want to extract the ages of people whose names are David Dewitt. We first find a tag $T$ with the text "Age" by keyword matching and obtain the 2-D coordinates of the tag using its DOM node. Then, extracting any column $C$ below $T$ can be formulated into the query, "find $C$ such that either $T$ (*equal*, *meets inverse*) $C$ or $T$ (*equal*, *before inverse*) $C$ is satisfied." The former constraint finds columns underneath $T$ that meet it, while the latter finds columns that are below $T$, but not meeting (not adjacent).

### B. Rectangle Algebra Query Language

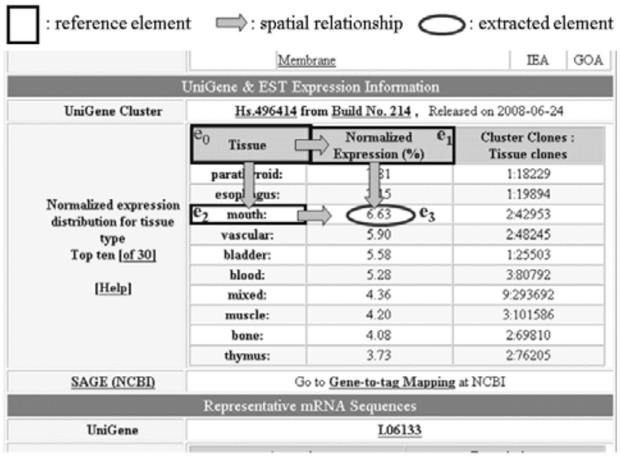
To understand a tuple extraction query language, we need to know how users typically find tuples to extract from a given web page. Before finding an element $e$ to extract from a web page, users typically seek elements relevant to $e$ [8]. Such relevant elements are called *reference elements*. For example, table captions, attribute names, and headers can constitute reference elements.

A tuple extraction query language must provide language constructs to express such user behaviors. This motivates us to develop a new tuple extraction language RAQuery to extract tuples from web pages using the rectangle algebra. RAQuery uses an XQuery-like syntax, adding two new constructs Match and RA. Match finds reference elements by matching various values. RA supports spatial joins by specifying the rectangle algebra among reference elements and extracted elements. For ease of understanding, we first explain RAQuery using examples rather than presenting the whole grammar.

Match(text="value") returns a set of elements in a web page whose text fields are "value." Various predicates on attributes of elements can be specified in Match as well.

RA($e_1, e_2, p$) represents a predicate that returns true if $e_1$ has the spatial relationship $p$ with $e_2$. Here, atomic relations in the rectangle algebra are used to represent the spatial relationship. For example, a constraint "$T$ (*equal*, *meets inverse*) $C$ or $T$ (*equal*, *before inverse*) $C$" is represented as RA($T$, $C$, [$x$:$equals$, $y$:$meets\_inverse$ or $before\_inverse$]).

Figure 2(a) shows a web page from a meta web site called SOURCE, which dynamically collects and compiles data from many bioinformatics web sites [9]. The web page shows a summary report about the gene named ATP7A.



| | Membrane | IEA | GOA |
|---|---|---|---|

UniGene & EST Expression Information

| | |
|---|---|
| UniGene Cluster | Hs.496414 from Build No. 214 , Released on 2008-06-24 |

Normalized expression distribution for tissue type
Top ten [of 30]
[Help]

| Tissue | Normalized Expression (%) | Cluster Clones : Tissue clones |
|---|---|---|
| parat[illegible]roid: | [illegible]31 | 1:18229 |
| esop[illegible]gus: | [illegible]15 | 1:19894 |
| mouth: | 6.63 | 2:42953 |
| vascular: | 5.90 | 2:48245 |
| bladder: | 5.58 | 1:25503 |
| blood: | 5.28 | 3:80792 |
| mixed: | 4.36 | 9:293692 |
| muscle: | 4.20 | 3:101586 |
| bone: | 4.08 | 2:69810 |
| thymus: | 3.73 | 2:76205 |

| | |
|---|---|
| SAGE (NCBI) | Go to Gene-to-tag Mapping at NCBI |

Representative mRNA Sequences

| | |
|---|---|
| UniGene | L06133 |

(a) An example page from the Stanford's SOURCE.

```
for e0 in Match(text="Tissue"),
    e1 in Match(text="Normalized Expression (%)"),
    e2 in Match(text="mouth:"),
    e3 in Match(*)
where RA(e0, e1, [x:meets_inverse or before_inverse, y:equal]) and
      RA(e0, e2, [x:equal, y:meets_inverse or before_inverse]) and
      RA(e1, e3, [x:equal, y:meets_inverse or before_inverse]) and
      RA(e2, e3, [x:meets_inverse or before_inverse, y:equal])
return e3;
```

(b) An RAQuery for the tuple extraction.

Fig. 2. Tuple extraction using RAQuery.

Suppose we want to extract from the web page the normalized expression of the mouth of ATP7A. We see from the figure that the target element we want to extract is located below the attribute "Normalized Expression" and to the right hand side of "mouth," which is one of the attributes of "Tissue." We denote the Tissue, Normalized Expression, mouth, and the target elements as $e_0, e_1, e_2$ and $e_3$ respectively. We first identify the three reference elements $e_0, e_1, e_2$ and their spatial relations such that:

```
for e0 in Match(text="Tissue"),
    e1 in Match(text="Normalized Expression (%)"),
    e2 in Match(text="mouth:"),
    ...
where RA(e0, e1,[x:meets_inverse or before_inverse, y:equal]) and
      RA(e0, e2, [x:equal, y:meets_inverse or before_inverse]) and
      ...
```

Then, the target element $e_3$ is expressed as anything (Match($*$)) that has spatial relations with $e_1$ and $e_2$ as follows.

```
for ...
    e3 in Match(*),
```

**where** ...
  **RA**(e1, e3, [x:equal, y:meets_inverse or before_inverse]) and
  **RA**(e2, e3, [x:meets_inverse or before_inverse, y:equal])

Figure 2(b) shows the final RAQuery to extract the target element.

## III. Spatial Join For Tuple Extraction

To understand how spatial join works for elements in a web page, we need to understand the structure of the DOM tree (more precisely, the W3C DOM tree) maintained in a web browser. The DOM tree maintains two important hierarchies: the *document hierarchy* and the *containment hierarchy*. Parent-child relationships in an HTML page constitute the document hierarchy, while containment relationships in an HTML page rendered by a web browser constitute the containment hierarchy. Each DOM node maintains three important types of pointers: *parentNode*, *childNodes*, and *offsetParent*. The first two pointers are used to form the document hierarchy: *parentNode* and *childNodes* point to the parent and child DOM nodes respectively in the document hierarchy, and *offsetParent* points to the nearest parent DOM node in the containment hierarchy. To represent spatial information (spatial coordinates) for each DOM node, the relative coordinates are used. That is, the coordinates of a bounding box for each DOM node $n$ are represented as relative coordinates from $n$'s parent bounding box in the containment hierarchy.

The parser in our system translates each RAQuery into a query execution plan (QEP). We provide two basic operators, Match and RAJoin. The algorithms of these operators are constructed using the iterator model [10] commonly used in query processors of commercial DBMSs. In the iterator model, the operators composing the QEP receive an input from the child operators and then provide the processed result to the parent operator. To obtain a result from each operator, the operator provides GetNext() function as the interface.

Figure 3 shows a QEP for the query of Figure 2(b). First, the results of Match("Tissue") and Match("Normalized Expression(%)") will be joined by an RAJoin operator. The results of that RAJoin will be joined with the results of Match("mouth") by the next RAJoin operator, which will then be joined with the results of Match($*$) by the final RAJoin operator.

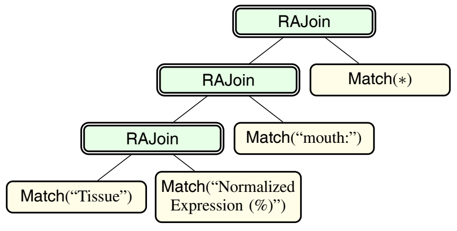


Fig. 3. A query execution plan for the query of Figure 2(b).

The Match operator runs as follows: it traverses DOM nodes in the document hierarchy using *childNodes* in breadth-first order, starting from the root of the DOM tree. For each DOM node $n$, the operator executes a matching operation using its parameter information. For example, Match("Tissue") executes a keyword match using the keyword "Tissue" on each DOM node. If $n$ is qualified, $n$ is forwarded to the parent operator. The absolute coordinates of the bounding box for $n$ are calculated at the same time and forwarded to the parent operator as well, which will be used for spatial joins. To calculate the absolute coordinates for $n$, we follow the *offsetParent* link from $n$ to the root in a bottom-up fashion.

The algorithm for the RAJoin operator employs a nested loop-based spatial join. The reason for using the nested-loop join is that the nested-loop is very effective when the number of tuples in the outer loop is small, and the number of results qualified by Match operators is typically very small in a web page. Specifically, the RAJoin operator is associated with an RA predicate list that must be applied to this join operator. For example, in Figure 3, the top RAJoin operator has two RA predicates, one between Match("Normalized Expression (%)") and Match($*$) and the other between Match("mouth:") and Match($*$).

## IV. Conclusion and Future Work

This paper proposed a novel framework and algorithms to robustly extract tuples from web pages. Existing research has focused on extracting tuples using XPath or the regular expression. However such techniques are vulnerable to small changes on the web pages. Our system, on the other hand, exploiting spatial joins can correctly extract tuples from web pages even when the pages are updated over time. Specifically, our system regards elements in the rendered page as spatial objects in the 2-D space and executes spatial joins to extract target elements. We believe that this work lays the foundation for future studies on robust tuple extraction from web pages.

## Acknowledgement

This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology(MEST) / Korea Science and Engineering Foundation (KOSEF), grant number R11-2008-007-03003-0. Hwanjo Yu is a corresponding author.

## References

[1] R. Baumgartner, S. Flesca, and G. Gottlob, "Visual web information extraction with lixto," in *VLDB*, 2001.
[2] W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak, "Towards domain-independent information extraction from web tables," in *WWW*, 2007.
[3] L. Liu, C. Pu, and W. Han, "Xwrap: An xml-enabled wrapper construction system for web information sources," in *ICDE*, 2000.
[4] K. Simon and G. Lausen, "Viper: augmenting automatic information extraction with visual perceptions," in *CIKM*, 2005.
[5] J. Wong and J. I. Hong, "Making mashups with marmite: towards end-user programming for the web," in *CHI*, 2007.
[6] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.
[7] P. Balbiani, J.-F. Condotta, and L. F. nas del Cerro, "A model for reasoning about bidimensional temporal relations," in *KR*, 1998.
[8] M. Kowalkiewicz, T. Kaczmarek, and W. Abramowicz, "Myportal: robust extraction and aggregation of web content," in *VLDB*, 2006.
[9] "Stanford SOURCE," http://smd.stanford.edu.
[10] G. Graefe, "Query evaluation techniques for large databases," *ACM Comput. Surv.*, vol. 25, no. 2, pp. 73–170, 1993.