Towards Deeper Understanding of the Search Interfaces of the Deep Web

Hai He[1], Weiyi Meng[1], Yiyao Lu[1], Clement Yu[2], and Zonghuan Wu[3]

[1] Dept. of Computer Science, SUNY at Binghamton, Binghamton, NY 13902
{haihe, meng, ylu0}@cs.binghamton.edu

[2] Dept. of Computer Science, Univ. of Illinois at Chicago, Chicago, IL 60607
yu@cs.uic.edu

[3] Center for Adv. Compu. Studies, Univ. of Louisiana at Lafayette, Lafayette, LA 70504
zwu@cacs.louisiana.edu

Abstract

Many databases have become Web-accessible through form-based search interfaces (i.e., HTML forms) that allow users to specify complex and precise queries to access the underlying databases. In general, such a Web search interface can be considered as containing an interface schema with multiple attributes and rich semantic/meta information; however, the schema is not formally defined in HTML. Many Web applications, such as Web database integration and deep Web crawling, require the construction of the schemas. In this paper, we first propose a schema model for representing complex search interfaces, and then present a layout-expression based approach to automatically extract the logic attributes from search interfaces. We also rephrase the identification of different types of semantic information as a classification problem, and design several Bayesian classifiers to help derive semantic information from extracted attributes. A system, WISE-iExtractor, has been implemented to automatically construct the schema from any Web search interfaces. Our experimental results on real search interfaces indicate that this system is highly effective.

Keywords: Web databases, search interfaces extraction, interface schema

Contact Author:

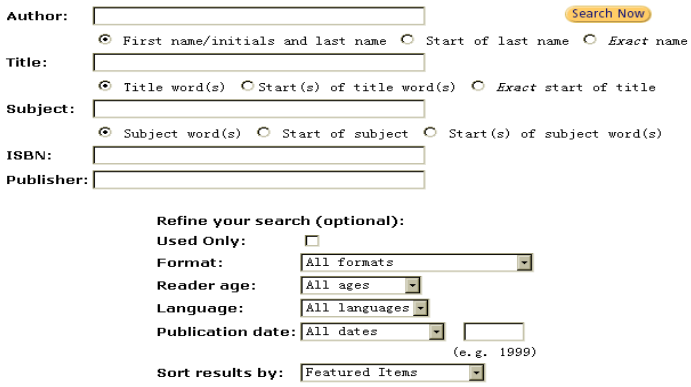
Prof. Weiyi Meng

Email: meng@cs.binghamton.edu
Tel: 1-607-777-2411, Fax: 1-607-777-4729

P.S.: The paper is an extended version of [9] and is invited to the special issue of WWW Journal.

1. Introduction

With the explosive growth of the Web, many Web databases driven by traditional database systems like Oracle and MySQL have become available for online access. Typically there are two ways to access Web databases: Web search interfaces (in HTML form) and Web services (in WSDL). Although some Web databases that support B2B applications can be accessed through Web services, such databases are rare at this time for the public to use. Web search interfaces are still the most popular way for ordinary users to access Web databases. Figures 1.a and 2 show three search interfaces used by three different Web sites to search their book databases. A Web search interface is implemented by an HTML form according to W3C HTML specification [10]. It typically contains some form control *elements* (such as *textbox*, *radio button*, *checkbox* and *selection list*) that allow users to enter search information. A descriptive text *label* is usually associated with an element to describe the semantic meaning of the element. Logically, elements and their associated labels together form different *attributes* (or query conditions) of the underlying database. Using such a schema-based interface, users can specify complex and precise queries and pass them to Web databases.



(a)

```
<FORM action=/exec/obidos/search-handle-form/ref=s_sf_b_as/102-8871152-8876952 method=post>
<TABLE border=0>
  <TBODY>
  <TR>
    <TD><FONT face=verdana,arial,helvetica size=-1><STRONG>Author:</STRONG></FONT></TD>
    <TD><INPUT size=40 name=query-0></TD>
    <TD>  <INPUT type=image alt="Search Now" src="amazon_files/search-now.gif"> </TD>
  </TR>
  <TR>
    <TD></TD>
    <TD colSpan=2><FONT size=-1>
    <INPUT type=radio CHECKED value=author-like name=field-0> First name/initials and last name
    <INPUT type=radio value=author-begins name=field-0> Start of last name
    <INPUT type=radio value=author-exact name=field-0> <I>Exact</I> name </FONT>
    </TD>
  </TR>
  <TR>
    <TD><FONT face=verdana,arial,helvetica size=-1><STRONG>Title:</STRONG></FONT></TD>
    <TD><INPUT size=40 name=query-1></TD>
    <TD><BR></TD>
  </TR>
  <TR>
    <TD></TD>
    <TD colSpan=2><FONT size=-1>
    <INPUT type=radio CHECKED value=title name=field-1> Title word(s)
    <INPUT type=radio value=title-words-begin name=field-1>Start(s) of title word(s)
    <INPUT type=radio value=title-begins name=field-1> <I>Exact</I> start of title </FONT>
    </TD>
  </TR>
  ...
  <TR>
  <TD><FONT face=verdana,arial,helvetica size=-1><STRONG>Publisher:</STRONG></FONT></TD>
  <TD><INPUT size=40 name=field-publisher></TD>
  <TD><BR></TD>
  </TR>
  ...
</FORM>
```

(b)

Figure 1: amazon.com book search interface (a) and a fraction of its HTML source text (b)

Figure 2: Examples of exclusive attributes.

Definition 1: *Given a set of attributes* $S_i = \{A_1, A_2, \ldots, A_n\}$ and *a sub-set* S_x *of* S_i*, i.e.,* $S_x \subseteq S_i$*, if only one attribute in* S_x *can be used to form a query with other attributes in* $S_i - S_x$ *at a time, then the attributes in* S_x *are called* **exclusive attributes**.

In order to utilize Web databases, the first essential step is to understand search interfaces [2]. However, the schemas on search interfaces are not formally defined because Web search interfaces are implemented in HTML for human users to use through a Web browser. Specifically, semantically related labels and elements that form logical attributes are scattered in the HTML source text like in Figure 1.b, and the associations of these labels and elements are not formally defined. Moreover, as search interfaces are designed independently they often have different contents, presentation styles and query capabilities. Therefore, it is challenging to automatically identify the attributes of each interface and to fully understand the semantic information related to those attributes.

Although some works on extracting search interfaces have been reported [11, 13, 18, 22], they are highly inadequate as they focus on only labels and elements. As a matter of fact, beyond individual labels and elements, a substantial amount of semantic/meta information about a search interface is "*hidden*" (i.e., not machine understandable) and needs to be revealed in order to better utilize the underlying Web database. For example, in Figure 1.a `Publication date` implies that the attribute semantically has a *date* value type, and its two elements play different roles in specifying a query condition. As yet another example, in Figure 2, the values in the selection list ("Title", "Author", "Keyword" and "Publisher") are in fact attribute names, and they are *exclusive attributes*.

Many Web applications are related to Web databases and require the contents of their search interfaces to be understood and properly organized for computer processing. These applications include, but not limited to, schema matching across multiple interfaces [1, 6, 8, 21], unified interface generation [8], deep Web crawling [18], programmatically interfacing with Web databases (say for meta-searching and source query mapping applications [3, 14, 23]), clustering/classifying Web databases [7, 15, 17], and annotating the returned results of Web databases [19].

In this paper, we present a general framework for making the search interfaces of Web databases machine understandable. First, an *interface schema model* for capturing the semantic/meta information available on complex search interfaces is presented. Second, we present our method for automatically constructing the

schema model to make the "hidden" information on the search interfaces *explicit* and *machine understandable*. Our current method consists of two major steps: 1) *Attribute Extraction*, i.e., to associate semantically related labels and elements to form logical attributes and to identify an appropriate label for each attribute; 2) *Attribute Analysis*, i.e., to reveal the "*hidden*" meta information to help deeper understanding of the search interfaces.

This paper has the following contributions:

1) We propose an interface schema model to describe the contents and capabilities of Web search interfaces for deeper understanding of them. This model contains not only attributes but also a substantial amount of semantic/meta information about the attributes, such as domain type, value type, and relationships among elements.
2) We present a practical solution to the automatic identification of logical attributes on search interfaces. This solution consists of two novel components: (1) LEX, a layout-expression-based form extraction approach is designed to automatically extract attributes from search interfaces; (2) the knowledge from multiple interfaces in the same domain is leveraged to help extract exclusive attributes.
3) We rephrase the identification of attribute semantic/meta information as a classification problem, and propose the use of multiple Bayesian classifiers exploiting different types of information to derive attribute semantic/meta information.
4) Extensive experimental results are reported, and they indicate that our solution for automatic search interface schema extraction is highly accurate and robust.

The rest of this paper is organized as follows. In Section 2, we present our schema model for search interfaces. In Sections 3 and 4, we discuss our methods for constructing the schema, including automatic attribute extraction and attribute analysis. We describe the system implementation in Section 5. The experimental results are reported in Section 6. In Section 7, we review related work. Finally, Section 8 contains the conclusions.

2. Schema Model for Representing Search Interfaces

In this section, we present a three-level schema model for Web search interfaces that captures rich semantic information. The top level provides an overview of an interface such as site information and the list of attributes. The second level describes each attribute in detail. As an attribute may consist of multiple elements, the third level contains information related to each element. This model provides some flexibility for supporting various applications that may need information at different level of detail. For completeness of interface representation, the model also includes syntactic information of a search interface, such as element name and element type (see below). In our model, at the top level, an interface is represented as *F*

= (S, {A_1, A_2, …, A_n}, C_f), where S is the site information associated with the form, such as the site URL, the server name and the HTTP communication method, {A_1, A_2, …, A_n} is an ordered list of attributes on the interface, and C_f is the form constraint (the logic relationship of the attributes for query submission). At the second level, each A_i is represented as (L, P, DT, DF, VT, U, R_e,{ E_j, E_{j+1},…, E_k }, C_a), where L is the attribute label of A_i (if applicable), P is the layout order position of A_i, DT is the domain type of A_i, DF is the default value of A_i, VT is the value type of A_i, U is the unit of A_i, {E_j, E_{j+1},…, E_k} is an ordered list of *domain elements* of A_i, R_e is the relationship type of the domain elements, and C_a is the constraints of the attribute. At the third level of our model, each domain element E_i is represented as (L_e, N, F_e, V, DV), where L_e is the *element label* (possibly empty), N is the (internal) name of the element, F_e is the format (e.g. textbox, selection list, checkbox and radio button), V is the set of values of the element (it is null for textbox), and DV is the default value of the element (possibly null). The choice of the specific information included in our model for attributes and elements is based on the usefulness of such information in several applications such as Web interface clustering and integration. Other information could be added to the model when the need arises.

We would like to emphasize here that both of {A_1, A_2, …, A_n } and {E_j, E_{j+1},…, E_k } are ordered in the model. Changing the order of the elements in {E_j, E_{j+1}, …, E_k } may change the semantics. For example, changing the order of the two elements of `Publication Year` in Figure 3 would make the semantics very confusing. Although changing the order of the attributes in {A_1, A_2, …, A_n } would not effect semantics of attributes, the order somehow reflects the importance as well as how the grouping of the attributes on the search interface as perceived by the designers or users. This model can be implemented in XML-based ontology languages such as OWL but this aspect will not be discussed in this paper.

Wu at al. [21] model Web search interfaces as a hierarchical tree structure; however, their model just captures the structure semantics of search interfaces and does not consider the content semantics of search interfaces. Our model is more complete in the sense that it considers both structure and content semantics.

In the following, we will explain some of the concepts in the proposed interface schema model while some other concepts will be explained in subsequent sections.

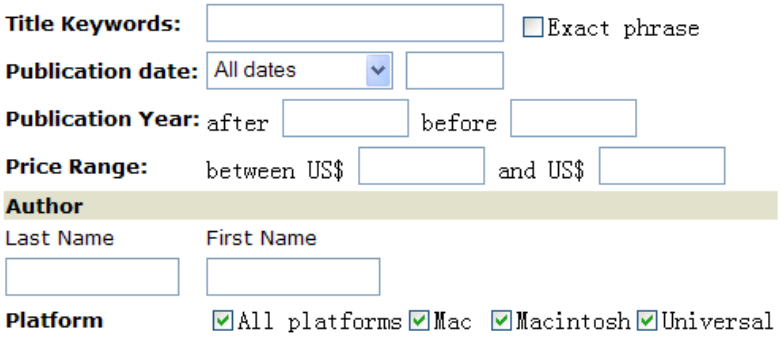


Figure 3: Examples of element relationship type.

Relationships of elements:

On a search interface, an attribute may have multiple associated elements and they may be related in different ways. There exist four types of element relationships: *range type*, *part type*, *group type* (multiple checkboxes/radio buttons are sometimes used together to form a single semantic concept/attribute) and *constraint type*. For example, in Figure 3, the relationships between the elements of `Publication Year` are of *range type* (for specifying range query conditions); the relationship of the elements of `Author` is of *part type* (i.e., last name and first name are parts of a name); that of `Platform` is of *group type*; and finally "Exact phrase" is of *constraint type* as it specifies a constraint on `Title keywords`.

When an attribute has multiple associated elements, we shall classify them into two types: *domain elements* and *constraint elements* because they usually play different roles in specifying a query. Domain elements are used to specify domain values for the attribute while constraint elements enforce some constraints to domain elements. For example, in Figure 3, element "Exact phrase" is a constraint element while the textbox following `Title keywords` is a domain element.

Identifying the relationships between the elements of each attribute would help interface schema integration [8, 21] and query mapping [3, 14]. Consider two interfaces. One interface contains an attribute `Publication date`, and another interface contains an attribute `Publication year`, and they should be matched in terms of their semantics. But we cannot match them by only using names because they do not have exactly the same attribute name. However, if we can identify that the elements of both attributes are of *range type*, it would increase the confidence of matching them. As to query mapping, consider a local search interface containing an attribute `Title keywords` as shown in Figure 3. When a user specifies a query on the global attribute `Title` of a mediated interface, during the query translation the query value should be mapped to the domain element of `Title keywords` instead of the constraint element "Exact phrase".

Element label:

In Figure 3, attribute `Publication Year` has two elements whose labels are "after" and "before" respectively. In this case, "Publication Year" is treated as the label of the attribute. Element labels are considered as the *child* labels of their attribute, which usually represent the semantics of the elements. For example, "after" and "before" are used to indicate the lower and upper bounds of the attribute.

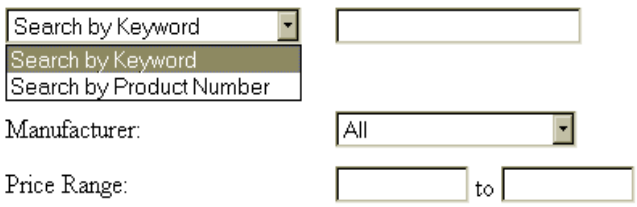


Figure 4: Example of hybrid relationship.

Logic relationship of attributes:

Attributes on a search interface can be logically combined in different ways to form a query to access the underlying database. Correctly identifying the logic relationship of an interface is important for successful query mapping and submission by a program. Generally, there are four possibilities:

1) **Conjunctive**. All the attributes are combined through an "*and*" Boolean logic operator, meaning that all specified conditions must be satisfied at the same time.
2) **Disjunctive**. All the attributes are combined through an "*or*" Boolean logic operator, meaning that at least one specified condition must be satisfied.
3) **Exclusive**. In this case, only one attribute can be chosen to form a query at a time. In Figure 2, the attribute names appear in a group of radio buttons or a selection list and only one attribute can be used at a time to submit a query.
4) **Hybrid**. This is a combination of the above three cases. In this case, some conditions may be conjunctive, some may be disjunctive, and some may be exclusive. For example in Figure 4, `Keyword` and `Product Number` are *exclusive*, while each of them is *conjunctive* with `Manufacturer` and `Price Range`.

Now we give an example to show how an attribute is represented using the proposed model.

Example 1: The attribute `Price Range` on the search interface in Figure 3 can be represented as (Price Range, 4, *range*, *null*, *currency*, *USD*, *range*, {(*"Between US$"*, *"low"*, *textbox*, ∅, *null*), (*"And US$"*, *"high"*, *textbox*, ∅, *null*)},∅), where "low" and "high" are the internal names of the two textboxes in the HTML text, and ∅ denotes an empty set of constraints.

3. Attribute Extraction

Labels and elements are the basic components of a search interface, but it is insufficient to just extract individual labels and elements because many applications (such as [7, 8, 21, 19]) rely on the *logical attributes* formed by related labels and elements. In order to extract logical attributes, it is essential to determine the semantic associations of labels and elements. However, there are no explicit definitions of such associations in the HTML text of the search interface. We observe that labels and elements that represent the same attribute have a certain layout pattern and are usually close to each other, and that in most cases they have some similar information in common. Furthermore, the search interfaces that contain exclusive attributes have certain composition regularities. On the basis of this, we develop a three-step approach to tackle the problem of automatic attribute extraction.

3.1 Interface Expression

Labels and elements on a search interface are *visually* arranged in one or more rows by a browser. To approximately capture the *visual layout* of these labels and elements, we introduce the concept of ***i**nterface **exp**ression* (IEXP). For a given search interface, its IEXP is a *string* consisting of three types of basic items 't', 'e' and '|', where 't' denotes a label/text, 'e' denotes an element, and '|' denotes a row delimiter which represents a physical row border on the search interface.

For example, the search interface in Figure 1.a can be represented as "`te|eee|te|eee|te|eee|te|te|t|te|te|te|te|tee|t|te`", where the first 't' denotes the label "Author", the first 'e' denotes the textbox following the label "Author", the first '|' denotes starting the second row as shown in Figure 1.a, and the following three 'e's denote the three radio buttons below the textbox (the text on a radio button/checkbox is treated as the value of the element in this dissertation, thus the text and its radio button/checkbox together are considered as a whole entity). The remaining 't's, 'e's and '|'s can be understood in a similar manner.

The IEXP of an interface is obtained when extracting individual labels and elements from the search interface (see Section 3.2 below). The IEXP provides a *high-level* description of the *visual layout* of different labels and elements on the interface while ignoring the details like the values of the elements and the actual implementations of laying out labels and elements. Our method for automatic extraction of attributes will rely on this expression. We may sometimes encounter HTML forms that use complex nested tables for alignment. In such cases, it is possible that the IEXP may fail to capture the exact layout of a search interface. However, our attribute extraction method does not require the IEXP to be completely accurate because our method is able to work robustly on the approximate layout captured in the IEXP as confirmed by our experimental results.

3.2 Extracting Individual Labels and Elements

This is the first step of our automatic attribute extraction method. Given a search interface, the extraction starts with its "`<FORM>`" tag. When a label or an element is encountered, a 't' or 'e' is appended to its IEXP (initially it is empty) accordingly. Each element itself contains its values (if available). Four types of input elements are considered: *textbox, selection list, checkbox* and *radio button*. When a row delimiter like "`<BR>`", "`<P>`" or "`</TR>`" is encountered, a '|' is appended to the IEXP. This process continues until the "`</FORM>`" tag is encountered. In this process, some irrelevant texts (e.g., "Refine your search (optional)**"** in Figure 1.a) may be included in the IEXP even though some efforts are made to identify and discard them (e.g., texts that are too long or parenthesized are discarded). In our experiments, we found that these irrelevant texts can be discarded automatically during the step of grouping labels and elements.

After this process, all individual labels and elements on the search interface are extracted, and the corresponding IEXP of the interface is also constructed.

3.3 Identifying the Names of Exclusive Attributes

Exclusive attributes are actually the ones whose names may appear as *values* in some elements, such as a group of *radio buttons* or a *selection list* (e.g., attribute names in Figure 2). Correctly recognizing such attributes automatically is difficult because they do not appear on search interfaces as descriptive texts like those in Figure 1.a.

By our observation, exclusive attributes appear frequently on real Web search interfaces. Among the 184 real Web search interfaces we collected and investigated, 50 interfaces contain exclusive attributes; in particular, more than 34% of the interfaces in books, movies and music domains have exclusive attributes. A significant flaw of existing approaches [4, 11, 13, 18, 22] for interface extraction is that they do not extract exclusive attributes.

B. He et al [6] reported their theory that there is a *hidden common schema* among Web search interfaces for each domain and the vocabulary of this common schema stabilizes at a reasonably small size even though the number of search interfaces in the domain can be very large. So it is possible that we can obtain a vocabulary of attributes for a domain when a large number of search interfaces of the same domain are considered. We also observed that the names of exclusive attributes are often the *most commonly used attribute names* of a domain. Based on the aforementioned theory and observation, we propose a novel and simple statistical approach to tackle the problem of identifying exclusive attributes. The basic idea of our approach is that we consider multiple interfaces in the same domain at the same time rather than separately (Web interfaces can be clustered first such that each cluster contains the interfaces from the same domain [7, 15, 17]). Then we use the extracted labels from all search interfaces of the same domain to construct a vocabulary for the domain. Finally we use the vocabulary to automatically identify and extract the names of exclusive attributes. The details of our approach are provided below.

After the labels are extracted from all the interfaces of the same domain using the method described in Section 3.2, we compute the *Label Interface Frequency* (*LIF*) of each label, which is the number of search interfaces of the domain that contain the label. Some simple normalization operations on each label (e.g., convert each word to its base form using WordNet [20]) are done before the computation. Then only labels whose $LIF \geq ft$ are selected to construct the *common schema vocabulary* of the domain, where ft is computed by $ft = p \times n$, with n being the total number of considered search interfaces of the domain, and p being the threshold known as the minimal percentage of search interfaces that contain the label. With this approach, for example, the common schema vocabulary for the *books* domain of our testing data is

{*subject, publisher, author, keyword, title, ISBN*}; obviously, these attribute names are the ones most commonly used for books.

Using the obtained schema vocabulary, we now find the search interfaces that contain some elements whose values are words in the vocabulary, and then extract the attribute names from these elements. On search interfaces, two patterns as shown in Figure 2 are widely used to organize exclusive attributes: one pattern consists of a group of *radio buttons* and a *textbox*; the other consists of a *selection list* and a *textbox*. Based on these two patterns, if a certain percentage (*pv*) of the values of the element(s) (a selection list or multiple radio buttons) are found to be contained in the schema vocabulary and the *textbox* is nearby, the values are extracted as attribute names and the textbox is treated as the domain element of each attribute. When exclusive attributes are identified from the interface, the old IEXP of the interface is replaced by a new IEXP to reflect the fact that new labels have been identified (note that only labels are captured in IEXP but values are not). The new IEXP is formed by placing each of the newly identified labels at a different row followed by a textbox element.

3.4 Grouping Labels and Elements

This step is to group the labels and elements that semantically correspond to the same attribute, and to find the appropriate attribute label/name for each group. For example, in Figure 1.a, label "Author", the textbox, the three radio buttons and their values below the textbox all belong to the same attribute, and this step aims to group them together and identify label "Author" as the name of the attribute.

To achieve this objective, a *l*ayout-*e*xpression-based e*x*traction technique (LEX) is developed. The basic idea of LEX is as follows. Consider a search interface whose IEXP organizes texts/labels and elements into multiple rows. For each element *e* in a row, LEX finds the text either in the same row or in the two rows above the current row that is most likely to be the attribute label for *e* based on an *association weight* of the text with *e* computed using five measures (see below), and groups *e* with this text. In the end, all related elements that are grouped with the same text and the text itself will be considered to belong to the same attribute by our method. The reason for considering only two rows above the current row of an element is that related labels and elements are generally never laid out too far apart on a search interface. The details of LEX are given below.

Step 1: Take an element *e* in the current IEXP. For each text in the row of *e*, compute its association weight with *e* using the five measures to be described shortly. Let *curRowWeight* denote the highest weight among all these texts.

Step 2: Take the first closest row above the current row and compute the association weight between each text in this row and *e*. If the highest weight is positive, denote it as *aboveRowWeight*, and then go to step 3;

otherwise, take the second closest row above the current row and repeat the same process to get *aboveRowWeight*. No other rows are considered.

Step 3: If *aboveRowWeight* is greater than a threshold *wt*, compare *curRowWeight* and *aboveRowWeight*, and choose the text with the larger weight as the *attribute label* associated with *e*. Otherwise, if *curRowWeight* is greater than zero, the text with *curRowWeight* is chosen as the attribute label associated with *e*. If it is not, we assume no associated label can be found, and *e* will stay alone with an anonymous label.

The above process continues until all the elements in the IEXP are processed. Once again, if multiple elements are associated with the same attribute label by the above process, those elements and the label together will form a single semantic attribute. Besides, if the text closest to *e* is not recognized as the attribute label of *e*, it is assumed to be the element label of *e*. For example, in Figure 3 if text "after" is not recognized as the attribute label for the textbox, it will be considered as the element label of the textbox.

Now we present the measures for computing the association weight of a text and an element. Since search interfaces are independently designed, designers often use different ways to semantically associate labels and their elements so that human users can understand the search interface easily. For example, labels and their elements must be laid out close to each other either vertically or horizontally in search interfaces; a label ending with a colon might be the attribute label of its nearby elements. Based on our analysis of several sets of search interfaces (see details below), we identified five heuristic measures that are useful for predicting the association of labels and elements. The five heuristic measures are described below:

1) *Ending colon.* Frequently, an attribute label ends with a colon, while other texts do not. For example, in Figure 3, "Publication Year" is the attribute label with an ending colon, but "after" and "before" are not.
2) *Textual similarity of element name and text.* An attribute label and its elements may have some words/characters in common. For example, in Figure 1.b, we can see that the label `Publisher` has its element whose name is "`field-publisher`".
3) *Distance of the element and text.* An attribute label and its elements usually are close to each other in a search interface. Such relationships are also expressed in the IEXP. If there exist multiple texts nearby, then the closest text to an element is most likely to be the attribute label for the element. We define the distance of a text and an element as $dist = 1 / |Ie - It|$, where *Ie* and *It* are the position indexes of the element and the text in the IEXP, respectively.
4) *Vertical alignment.* An element and its associated attribute label may be placed into different rows with one row having label and the row below it having the element. In this case, they are aligned vertically, i.e., they have the same visual position in their respective rows. For example, in Figure

5, the four attribute labels `Price`, `Format`, `Age` and `Subjects` are all vertically aligned with their elements.

5) *Priority of the current row.* Although we consider two extra rows above the current row of an element, the current row is often given higher priority because in most cases an attribute label appears in the same row as its elements.

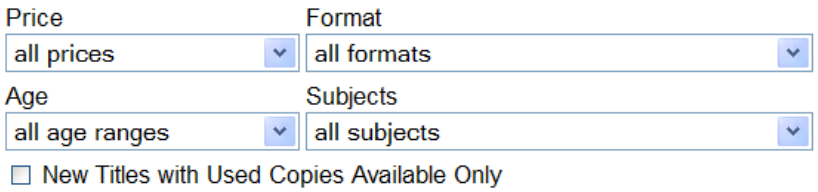


Figure 5: Example of vertical alignment.

The selection of the above heuristics is based on our analysis of three different datasets used in our experimental study (the first dataset is collected by ourselves, the other two datasets are independently collected; see Section 5 for more details). The result of our analysis is shown in Table 1. For each heuristic measure, we compute the percentage of all attributes for which this measure is applicable. In Table 1, the column "#Attrs", which denotes the number of attributes, does not include exclusive attributes since they are dealt with separately. From Table 1, we can see that these measures are widely applicable; in particular, heuristic measures 1, 3 and 5 are applicable for most attributes. We would like to mention that we also analyzed several other measures (e.g., bold style of labels) but they were found not as useful as these and, therefore, are not used.

	#Attrs	**#h1**	**#h2**	**#h3**	**#h4**	**#h5**
Dataset 1	922	60%	56%	88%	16%	73%
Dataset 2	736	50%	54%	68%	31%	54%
Dataset 3	139	72%	53%	88%	36%	60%

Table 1: Applicability statistics of the selected heuristic measures.

4. Attribute Analysis

Once attribute extraction is finished, all labels and elements have been organized into logical attributes, and each attribute has an *element list* that contains its elements. In this section, we first introduce the naïve Bayesian classification as the basic method of our attribute analysis, and then discuss how to derive semantic information about attributes beyond labels and elements using the classification approach.

4.1 Naïve Bayesian Classification

The naïve Bayesian classifier is one of the most popular and effective classifiers [5]. Suppose there are m classes, C_1, C_2, … , C_m. Given an unknown data sample $d = \{f_1, f_2, \ldots, f_n\}$ with n features, the naïve Bayesian classifier will assign d to the class C_i that has the maximum probability $P(C_i|d)$. By Bayes theorem, this is equivalent to maximizing $P(d|C_i)P(C_i)$. $P(C_i)$ is estimated as the portion of the training examples that belong to class C_i. With the assumption of class conditional independence (values of the features are conditionally independent of one another), $P(d|C_i)$ can be computed as $P(f_1|C_i)\,P(f_2|C_i) \ldots P(f_n|C_i)$, where $P(f_j|C_i)$ is estimated from the training examples. As we have to identify different types of semantic information for each attribute, we design multiple Bayesian classifiers for different purposes. While different classification tasks may need different feature information, almost all the feature information comes from attribute labels and elements (labels, names, formats and values). A feature is treated as a string of tokens that may come from the attribute label or one or more elements. $P(f_j|C_i)$ is estimated as the portion of the training examples of class C_i whose feature f_j token string has at least one common token with the corresponding feature of d. It is possible that a feature value of d is unknown to all training examples of class C_i or is null. That will produce a zero probability $P(f_j|C_i)$. We use the approaches described in [12] to address this special issue.

For each classifier, we construct a different training dataset and manually assign a class label for each data sample. To improve the performance, we also remove some meaningless stopwords such as “the”, “with” and “any” from each feature token string.

4.2 Differentiating Elements

As mentioned earlier, differentiating domain elements and constraint elements is important for precisely understanding the composition of the attributes. We observed that domain elements and constraint elements have the following characteristics:

1) Textboxes cannot be used for constraint elements.
2) Radio buttons or checkboxes or selection lists may appear as constraint elements.
3) An attribute consists of a single element cannot have constraint elements.
4) An attribute consisting of only radio buttons or checkboxes does not have constraint elements.

Based on these observed characteristics, a simple two-step method is used to differentiate domain elements and constraint elements. First, identify the attributes that contain only one element or whose elements are all radio buttons, or checkboxes or textboxes. Such attributes are considered to have only domain elements. Next, an Element Classifier is designed to process other attributes that may contain both domain elements and constraint elements. In order to do this, each element is represented as a feature vector of four features including element name, element format type, element relative position in the element list, and element

values. Each element is considered separately and classified into either domain element or constraint element.

4.3 Identifying the Relationships and Semantics of Domain Elements

If an attribute has multiple *domain elements*, we need to identify how these elements are related and the semantic meaning of each individual element. For example, in Figure 3, the two domain elements of `Price range` are used to represent a range with the first element for the *lower bound* of the range and the second for the *upper bound*. Three types of relationships for multiple domain elements are defined and they are *group*, *range* and *part*. As the *group* type is easy to identify (elements in *group* relationship are all checkboxes or radio buttons), we will focus on the discussion of the other two types.

Domain elements of most attributes have *part* relationships with each one representing a semantic component value of the attribute. *Range* type can be considered as a special case of *part* type indicating that elements are used to specify a range query condition. Range type is always implied by some widely used range keywords and patterns in names and values, for example, "`between-and`" and "`from-to`". To identify the relationships and semantics of domain elements, we designed two naïve Bayesian classifiers: one is for identifying relationships and the other for identifying element semantics. For the relationship classifier, we model all the domain elements of each attribute as a feature vector of four features including attribute name, element names, element labels, and element values. *Group* and *part* are the two class labels of the classifier. For the element semantics classifier, we model each element as a feature vector of five features including its attribute name, relationship with other domain elements, element label, element name, and element values.

4.4 Deriving Meta-information of Attributes

In our interface schema model four types of meta-information for each attribute are defined: *domain type*, *value type*, *default value* and *unit*. These meta-data are only for *domain elements* of each attribute.

Domain type: Domain type indicates how many distinct values can be used for an attribute for queries. Four domain types are defined in our model: *range*, *finite* (with a finite number of possible values but no range semantics), *infinite* (with possibly unlimited number of values, e.g., textbox, but no range semantics) and *Boolean*. For the *Boolean* type, the attribute just consists of a *single checkbox*, and this checkbox is usually used to mark a yes-or-no selection. In our model, *Boolean* type is separated from the regular *finite* type as this separation makes the *Boolean property* of the attribute explicit. We only focus on identifying the other three types. In general, an attribute that has a textbox can be assumed to have *infinite* domain type, and that consists of selection lists can be assumed to have *finite* domain type. However, this assumption is not always the case. For example, in Figure 3, two textboxes are used to represent a range for

the attribute `Publication year`, thus the attribute should have *range* domain type; in Figure 5, the attribute `price` has a selection list that contains several price ranges for users to select. Therefore, identifying domain type of an attribute cannot only depend on the format type of its elements. Instead, the correct domain type should be predicted from the attribute labels, element labels and names, values together with format types. We design a Domain Type Classifier to combine these features and predict correct domain type based on the training examples.

Value type: Each attribute on a search interface has its own semantic value type even though all input values are treated as text values to be sent to Web databases through HTTP. For example, the attribute `Reader age` semantically has *integer* values, and `departure date` has *date* values. Value types currently defined in our model include *date*, *time*, *datetime*, *currency*, *id*, *number* and *char*, but more types could be added. We can obtain the useful information for identifying value type of attributes from their labels, element names and values. An important thing about value type is that the identical or similar attributes from different search interfaces of the same domain should have the same value type. We design a Value Type Classifier to classify each attribute into an appropriate value type, and construct a feature vector {*attLabel, elemLabels, elemNames, elemValues*} for each attribute using the available information.

Default value: Default values in many cases indicate some semantics of the attributes. For example, in Figure 1.a the attribute `Reader age` has a default value "`all ages`". A default value may occur in a selection list, a group of radio buttons and a group of checkboxes. It is always marked as "`checked`" or "`selected`" in the HTML text of search forms. Therefore, it is easy to identify default values.

Unit: A unit defines the meaning of an attribute value (e.g., *kilogram* is a unit for *weight*). Different sites may use different units for values of the same attributes. For example, a search interface from USA may use "`USD`" as the unit of its `Price` attribute, while another from Canada may use "`CAD`" for its `Price` attribute. Identifying the correct units associated with attribute values can help understand attributes, but not all attributes have units (e.g., attribute `author` and `title` do not have applicable units). Unit information can be contained in site URL (e.g., amazon.ca and amazon.co.uk respectively have suffixes *ca* and *uk*) and attributes themselves. We represent an attribute as a feature vector {*URLsuffix, attLabel, elemLabels, elemNames, elemValues*}, and design a Unit Classifier to identify units.

4.5 Identifying the Logic Relationships

As defined in Section 2, a query submitted through a search interface can be formed in four possible ways (i.e., *conjunctive*, *disjunctive*, *exclusive* and *hybrid*). As the *exclusive* relationship of attributes is already addressed in Section 3.2, we focus on the other three types of relationships in this section.

Clearly, if two attributes are in a *conjunctive* relationship, the number of results returned from a Web database for a query using both attributes to specify conditions cannot be greater than that using only one of these two attributes in another query; and if two attributes are in a *disjunctive* relationship, then the number of results using both attributes cannot be less than that using only one of them. Thus, in principle, logic relationships between attributes could be identified by submitting appropriate queries to a Web database and by comparing the numbers of hits for different queries. In reality, however, it is rather difficult to automatically find appropriate queries to submit and to extract the numbers of results correctly. Therefore, in our current implementation, we take a simple and practical approach to tackle the problem. We observe that some Web databases contain logic operators (i.e., *and*, *or*) on their interfaces. In this case, it is easy to identify the logic relationships among the involved attributes. Attributes that are not involved in explicit logic operators or exclusive relationships are assumed to have conjunctive relationships among themselves and with other attributes on the interface. Most interfaces have no explicit logic operators or exclusive attributes (e.g., Figure 1.a), so conjunctive relationships are assumed for attributes. If different types of logic relationships exist among the attributes on an interface, then a *hybrid* relationship is recognized for the interface. This simple approach, though heuristic, is really effective for identifying the logic relationships of attributes as shown in our experiments (by our experiments, 180 out of 184 forms used in our dataset are correctly identified).

5. System Implementation

We have implemented a new version of WISE-*i*Extractor in Java. We apply an HTML parser package (it is an open source package and can be downloaded from http://sourceforge.net/projects/htmltok/) to parse an HTML page to get all the labels and elements of each search interface on the page. Then we use the proposed methods to associate labels and elements, and derive semantic information. In the implementation, each attribute is modeled as an object that contains attribute label, semantic information and elements, therefore it is easy to manipulate attributes on the GUI, such as updating attributes, removing or recomposing an attribute. All the learning data for the classification purpose is stored in a database accessed through JDBC, which facilitates the classification steps. Figure 6 shows a screen-shot of WISE-*i*Extractor. With WISE-*i*Extractor, users can type a URL in the address bar to download a Web search interface directly from the Web, or use "File" menu to load a cached Web page containing a Web search interface from a local directory. Users can load multiple Web pages in this manner. Note that WISE-*i*Extractor requires all the Web search interfaces loaded to be for the same domain. When an interface is extracted, the extractor shows its extracted attributes. The original Web search interface is also shown in the tab window of "Local Interface", which makes it convenient for users to verify the extracted attributes. Figure 6 shows the Web search interface of Figure 1.a and its extracted attributes in two tab windows respectively.

To view the meta-information of each attribute, users just need to right-click the attribute and then the tab window of Meta-data will show the semantic/meta information of the attribute. Figure 7 shows the screen-shot of the meta-information of the selected attribute (i.e., `publication date`).

Users can also perform some updates on the extracted meta-information so that the updates can be made effective immediately. WISE-*i*Extractor also provides users with the flexibility of removing incorrect attributes and adding recomposed correct ones. Without going back to the original HTML text of a search interface, users can finish this within just a few clicks. That will significantly save the time to manually check the original HTML text to identify and extract the components of the correct attributes.

WISE-*i*Extractor can also generate a common schema vocabulary from multiple Web search interfaces of the same domain, and then use it to extract exclusive attributes. With multiple loaded Web search interfaces, users just select the command of "Construct common schema vocabulary" in "Parser" menu and the common schema vocabulary of these Web search interfaces will be automatically constructed and displayed in the right drop-down list following the address bar. Figure 8 shows the constructed vocabulary and the exclusive attributes extracted from one interface of Figure 2.

When interface extraction is finished, WISE-*i*Extractor uses the extracted information of the search interface to construct the interface schema model in XML format that can be used by other Web applications. Figure 9 shows the DTD of the schema model.

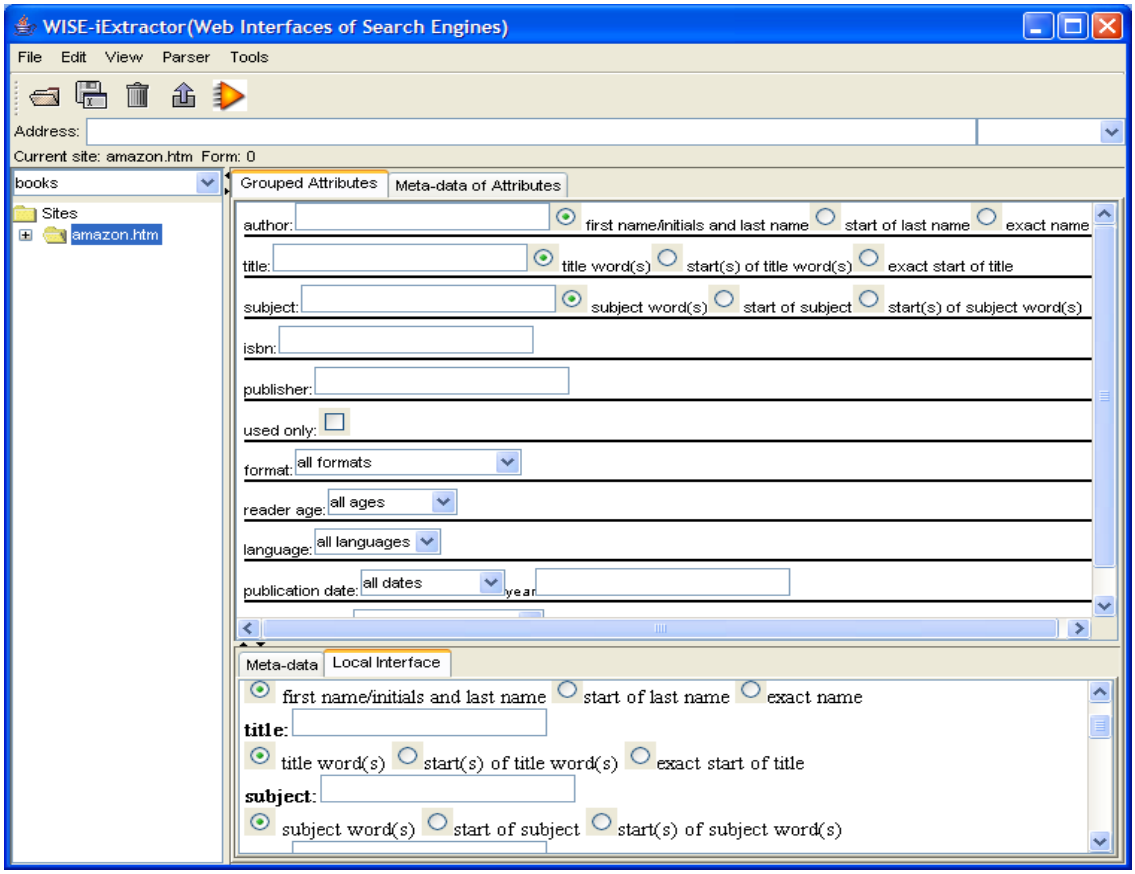


Figure 6: The screen shot showing the Web search interface and its extracted attributes.

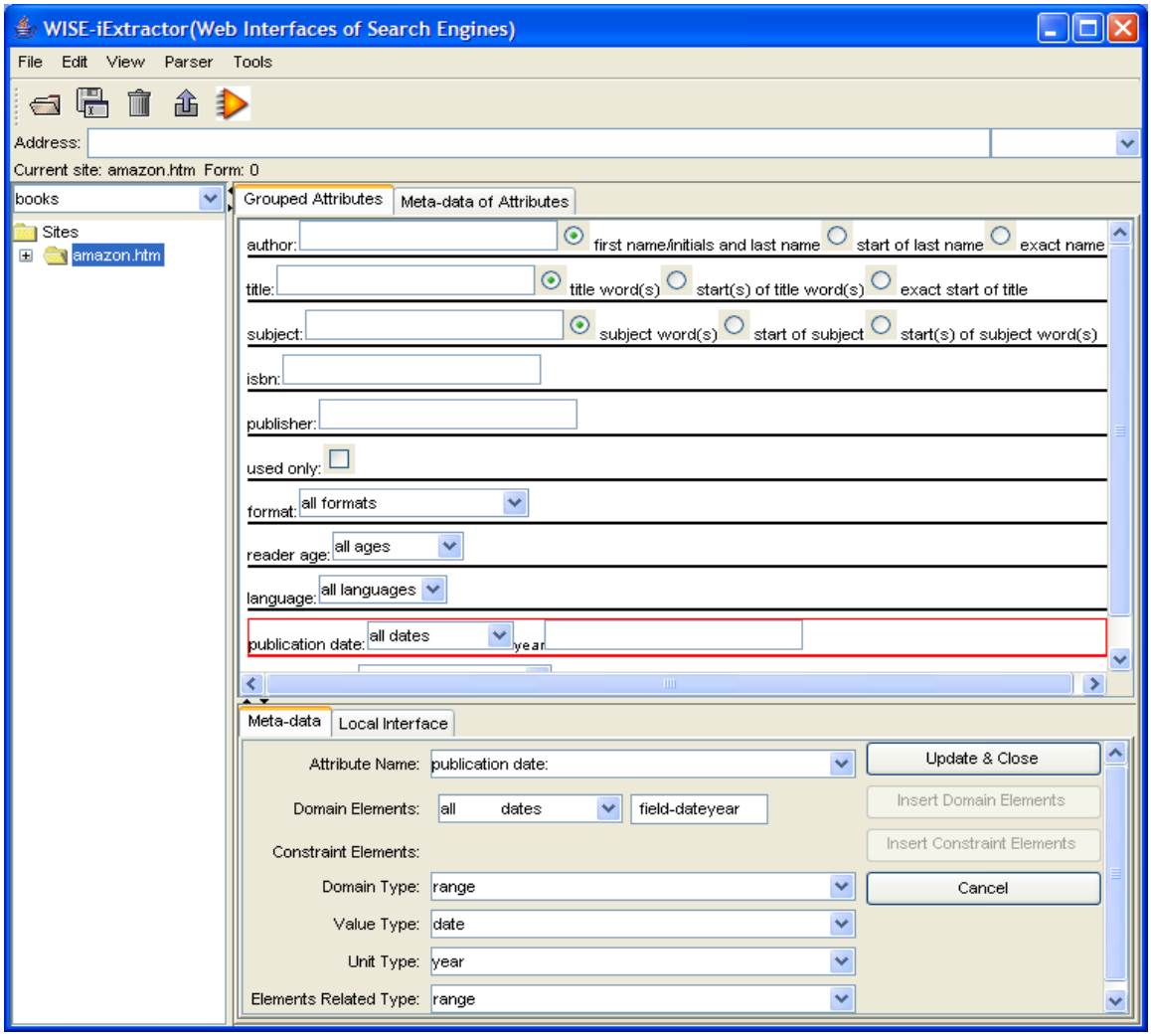


Figure 7: The screen shot showing extracted attributes and the semantic/meta information of the selected attribute.

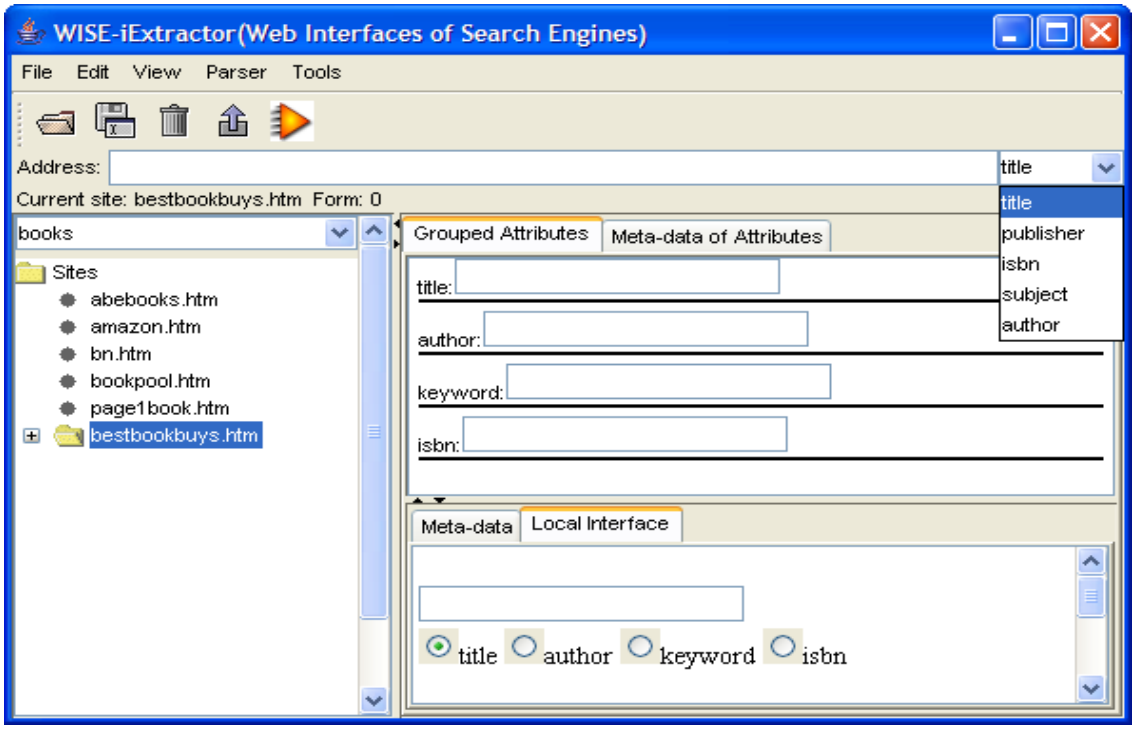


Figure 8: The screen shot showing common schema vocabulary, local Web search interface and its extracted logical attributes.

```
<!DOCTYPE  interface [
    <!ELEMENT interface(site_info,attribute+,form_constraint*)>

    <!ELEMENT site_info(domain,site_url,action,method)>
    <!ELEMENT domain(#PCDATA)>            //the domain of the web site
    <!ELEMENT site_url(#PCDATA)>          //the site  URL
    <!ELEMENT action(#PCDATA)>            //the action
    <!ELEMENT method(#PCDATA)>            //get or post

    <!ELEMENT attribute(label, elements?, att_constraint*)>
    <!ELEMENT label(label_name, position?, label_inter_name?, label_inter_value?)>
    <!ELEMENT label_name(#CDATA)>         //attribute label name
    <!ELEMENT position(#CDATA)>           //attribute position
    <!ELEMENT label_inter_name(#CDATA)>   //label internal name(exclusive atts only)
    <!ELEMENT label_inter_value(#CDATA)>  //label internal value(exlcusive atts only)

    <!ELEMENT elements(element+)>
    <!ELEMENT element(elem_name, elem_label?, defaut_value?, filled_values_num?, value*)>
    <!ELEMENT elem_name(#CDATA)>          //element internal name
    <!ELEMENT elem_label(#CDATA)>         //element label (child label)
    <!ELEMENT default_value(#CDATA)>      //default value of the element
    <!ELEMENT filled_values_num(#CDATA)>  //the number of values filled(selectionlist only)
    <!ELEMENT value(exter_value, inter_value)>  //value pair
    <!ELEMENT exter_value(#CDATA)>              //external value
    <!ELEMENT inter_value(#CDATA)>              //internal value

    <!ELEMENT att_constraint(element+)>         //attribute constraint
    <!ELEMENT form_constraint(label_name+)>     //form constraint(logic relationship of attributes)

    <!ATTLIST attribute  domain_type CDATA (infinite|finite|range|boolean)  "infinite" >
    <!ATTLIST attribute  value_type CDATA (date|time|datetime|currency|number|id|char)  "char">
    <!ATTLIST attribute  unit (usd|cad|…|unknown)  "unknown">
    <!ATTLIST label  type CDATA (text|hidden|selectionlist|checkbox|radio)  #REQUIRED>
    <!ATTLIST elements elem_related_type CDATA (alone|range|part|group)  #REQUIRED>
    <!ATTLIST element  format_type  CDATA (textbox|selectionlist|checkbox|radio)  #REQUIRED>
    <!ATTLIST att_constraint  type CDATA (constraint|other)  "constraint">
    <!ATTLIST form_constraint type CDATA (and|or|exclusive)>
]>
```

Figure 9: The DTD of interface schema model

6. Experiments

6.1 Datasets

To evaluate the interface extraction technique proposed in this paper, we collected various numbers of *raw* search interfaces from 7 application domains: books (60 interfaces), electronics (21), games (12), movies (30), music (32), toys (13) and watches (16). The total number of search interfaces used is 184 from 146 different sites (some site has multiple search interfaces). We also use two *independently collected datasets* from the works [19, 22] for performance comparison (see Robustness of LEX in Section 6.2). Currently we consider only search interfaces that do not contain scripts such as Javascript or Vbscript.

6.2 Form Extraction Results

In our current implementation, we only consider texts and form elements occurring within "<FORM>" and "</FORM>" tags. The text on a radio button or a checkbox is considered as the value of the element rather than a separate label. Currently we cannot extract labels that appear as icons/images. In our experimental dataset, only about 10 labels appear in icons/images.

Grouping extracted labels and elements into separate attributes is the most complex problem in search interface extraction. To evaluate our LEX method for this task, we manually identify the attributes of each search interface, and then compare them with the results of LEX. We evaluate the accuracy of LEX in two levels of granularity: *element level* and *attribute level*.

- **Element level:** A label is correctly extracted for an element if it matches the manually identified label. The experimental results of our method on the element level accuracy are shown in Table 2. The overall accuracy based on 1,582 elements from 184 forms is 97.66%.
- **Attribute level:** An attribute consists of up to three aspects of information: the name/label of the attribute, the set of domain elements and the set of constraint elements. An attribute extracted by LEX matches a manually extracted attribute if they match on all three aspects. Our experimental results for attribute-level accuracy are reported in Table 3, where "#Man" denotes the number of attributes identified manually, and "#LEX" denotes the number of attributes correctly extracted by LEX. The overall accuracy of 7 domains is 95.61%. We can see that the majority of the failures are caused by mistakes in extracting attribute labels. Our examination revealed that most of these failures in attribute label extraction are caused by the presence of a single checkbox (*single-checkbox problem*). In many cases a checkbox serving as a constraint element of an attribute is laid out in a separate row below other elements and the label of the attribute, while in other cases a checkbox with the same relative layout serves as a separate attribute. The current version of LEX could not differentiate these two cases and would mistake the latter case as the former case. For example, in Figure 3, if the checkbox "`Exact phase`" were below the textbox, it would be hard for LEX to know whether or not the checkbox is part of the attribute `Title keywords`.

Domain	**# elements**	**# elements whose label is correctly identified by LEX**	**Accuracy**
Books	484	473	97.7%
Electronics	191	185	96.85%
Games	72	72	100%
Movies	275	258	93.8%
Music	240	237	98.75%
Toys	62	62	100%
Watches	258	258	100%
Overall	1582	1545	97.66%

Table 2: Element-level accuracy of LEX.

Domain	#Man	# LEX	Accuracy	Errors		
				Label	Dom. elems	Const. elems
Books	370	351	94.8%	8	8	3
Electronics	146	137	93.8%	6	1	2
Games	63	63	100%	0	0	0
Movies	195	178	91.3%	11	1	5
Music	200	196	98%	2	2	0
Toys	59	59	100%	0	0	0
Watches	84	84	100%	0	0	0
Overall	1117	1068	95.61%	27	12	10

Table 3: Attribute-level accuracy of LEX.

The element level accuracy only considers a single element but not the whole attribute. It is more difficult to achieve high accuracy at the attribute level. For example, suppose an attribute has three elements and LEX identified only two of them; at the attribute level, this will be considered as a complete failure; but at the element level, the correctness rate will be 2/3. One problem with *element-oriented* approaches is that when an attribute and its elements all have labels, only the element labels are likely to be extracted but the attribute label is ignored. For example, in Figure 3, the attribute `Publication Year` has two elements with labels "after" and "before", respectively. An element-oriented approach like the one in [18] will recognize the element labels (they are closest to the elements) but ignore the attribute label "Publication Year". Our attributed-oriented approach will recognize "Publication Year" as the attribute label of the two elements, and "after" and "before" as the labels of the two elements, respectively.

Domain	Rh1	Rh2	Rh3	Rh4	Rh5	Only Keep h3
Books	-2	0	0	-6	-8	-14
Electronics	-1	-10	0	-4	-9	-19
Games	0	-1	0	0	0	-2
Movies	0	0	0	0	0	-7
Music	0	-1	-3	0	-9	-1
Toys	0	0	0	0	-4	0
Watches	0	-6	0	0	-2	-6

Table 4: Usefulness measurement of LEX heuristics.

Effectiveness of LEX heuristics: LEX uses five different heuristics to discover the association between elements and labels. It would be interesting to find out the impact of each individual heuristic on the accuracy of LEX. Additional experiments are conducted for this purpose. To find the impact of a particular heuristic, we remove it and use the remaining heuristics to group labels and elements into attributes and identify labels for them. The relative importance of the remaining heuristics is not changed. Table 4 shows

the experiment results. Column headed by "Rh#" indicates that the number of attributes that are negatively affected compared to the results of using all heuristics when the #th heuristic listed in Section 3.4 is not used. For example, for the books domain, when the first heuristic is removed, 2 more attributes are not correctly extracted compared to the results of using all heuristics. We also list the result when only the third heuristic (Distance) is used. Intuitively a label closest to an element is likely to be the attribute label and thus it seems to be reasonable to only use distance to associate labels and elements. The results in Table 4 indicate that, somewhat surprisingly, removing the third heuristic has little effect on the accuracy of LEX but keeping only the third heuristic would cause a large number of failures. In conclusion, each individual heuristic contributes positively to the attribute extraction process and using all the heuristics increases accuracy.

Robustness of LEX: To further evaluate the robustness of our approach, we carried out additional experiments using *independently collected datasets*. Two datasets are used for this purpose. The first is the dataset from the MetaQuerier project of UIUC [22] (the work is closely related to ours). We collected all the search forms from its demo site [16]. In this dataset, there are 106 search forms from 9 different domains (airfares, books, autos, jobs, music, movies, hotels, carRental and realestate). The search forms and the extracted results using the method proposed in [22] are posted at this demo site, and are readily usable for performance comparison with our approach. We run our system on these search forms without making *any change* to it. The overall attribute-level extraction accuracy of our method is 92.46%, while the accuracy obtained based on the method in [22] is 85.96%**.** Clearly, our method is significantly more accurate than the MetaQuerier method. The second dataset is from DeLa [19] that was used for testing the label assignment algorithm in DeLa. This dataset has 27 search forms from three domains (books, cars and jobs). The overall attribute-level accuracy of our method on this dataset is 92.73%.

We can see that the overall accuracies obtained using these two datasets are about 3% lower than the accuracy of 95.61% obtained on our dataset. The main reason for this performance drop is that there are more cases of the *single-checkbox problem* in these datasets than in our dataset. Despite of this, however, our approach still outperforms the approach in [22] significantly.

Effectiveness of extracting exclusive attributes: We also evaluate the effectiveness of the statistical approach for identifying and extracting exclusive attributes. The p used in Section 3.3 is 14% in our experiments to select the most popular labels to form the vocabulary of the common schema. With the vocabulary, our approach finds search interfaces in which a certain percentage (pv=30%) of the values of some element(s) are contained in the common schema vocabulary. The experimental results of this approach are shown in Table 5, where "#IE" denotes the number of search interfaces that have such attributes, "#Prog" denotes the number of such search interfaces that are correctly identified with this approach, "#Latts" denotes how many such attributes exist in those interfaces, and "#Eatts" denotes how many such attributes are correctly extracted. Note that the "#LEX" in Table 3 includes the "#Eatts", which

means, without using this approach, the "#LEX" of Table 3 would lose "#Eatts" in each corresponding domain, and thus the accuracy of Table 3 would decrease accordingly. We can see that our approach is very effective for recognizing such attributes that appear in the elements.

Domain	# IE	# Prog	# Latts	# Eatts
Books	21	20	98	95
Electronics	2	1	7	5
Games	3	3	10	10
Movies	12	10	40	32
Music	11	11	38	38
Toys	1	1	2	2
Watches	0	0	0	0

Table 5: Accuracy of identifying & extracting exclusive attributes.

6.3 Results of Attribute Analysis

As discussed earlier, we build multiple Bayesian classifiers to help derive some semantic information about attributes. To accomplish this, we select various numbers of search interfaces from the 7 domains of our dataset (184 search interfaces) and treat them as training examples. Totally 50 search interfaces are selected for the training purpose. We manually analyze the attributes in the training set and set the label for each type of semantic information. Then different Bayesian classifiers are built for each type of semantic information based on the training examples. These Bayesian classifiers are utilized to derive the semantic information of attributes on the remaining search interfaces (134 search interfaces). To evaluate the accuracy, we consider only attributes that are correctly identified by LEX. Table 6 shows our experimental results for each domain, where "DT" denotes domain type, "VT" denotes value type, "U" denotes unit, "ER" denotes element relationship, "DDC" denotes differentiation of domain elements and constraint elements, and "ElemSem" denotes identification of element semantics. Consider the book domain, out of the 200 attributes, the domain types of 200 attributes and the value types of 198 attributes are correctly identified respectively. Among these 200 attributes, 21 elements need to be given meaningful labels and the Element Semantics Classifier correctly identifies the labels for 17 elements. We can see that the Element Semantics Classifier does not work well in music and toys domain. The major reason for such failures is that no sufficient information exists in the training set to support the correct identification. We believe that better performance can be achieved if more training examples are collected and used. While the performance of the Element Semantics Classifier still has room for improvement, other 5 classifiers work very effectively and nearly all needed semantic/meta information can be correctly identified using the proposed methods.

Domain	# LEX	DT	VT	U	ER	DDC	ElemSem
Books	200	200	198	200	200	195	21 (17)
Electronics	74	74	71	74	74	74	12 (12)
Games	44	40	43	42	42	41	4 (2)
Movies	112	110	109	109	112	112	4 (4)
Music	127	125	123	125	123	126	3 (0)
Toys	40	40	39	39	39	39	5 (3)
Watches	63	61	62	61	63	63	2 (2)

Table 6: Accuracy of deriving semantic/meta information.

7. Related Work

The works reported in [11, 18, 22] are the most relevant to our work. The method in [11] uses a number of heuristic algorithms (such as N-gram and table layout) to find a good matching label for each form element. The LITE method in [18] uses a layout engine to obtain candidate labels that are physically closest to an element. Zhang et al [22] view search interfaces as a visual language and therefore use a number of manually pre-defined grammar rules to extract semantically related labels and elements.

The major differences between the techniques reported in [11, 18, 22] and our LEX approach are as follows. (1) The approaches in [11, 18] are not completely *attribute-oriented*, i.e., they find labels for elements but not for attributes. In contrast, LEX aims to group all logically related labels and elements together and identify an appropriate label for each attribute. (2) A grammar rule based framework is used in the approach reported in [22]. The grammar rules are defined in advance based on extensive manual observations of a large number of search interfaces. These rules are directly based on the visual layout of labels and elements. However, the similarities (such as textual similarity) between labels and elements are not considered in this approach. As indicated by the authors in [22], the method itself can easily cause two major problems (i.e., conflicting and missing elements); for example, the same element may be grouped into two different attributes and an element may be missed from an attribute even though it has some similar information with its label. Our approach not only considers the layout regularities of attributes but also considers similarities between labels and elements. As a result, our approach will not cause any element to be grouped with more than one attribute and it will also reduce the number of missing element cases significantly even though such cases cannot be completely avoided. Our experiments based the same data set in [22] indicate that our method outperforms the method in [22] significantly. (3) None of the techniques reported in [11, 18, 22] address how to extract exclusive attributes, which appear in many real search interfaces. Leveraging the knowledge from multiple interfaces in the same domain to help extract exclusive attributes is a unique feature of our approach. (4) Our solution is much more comprehensive for representing search interfaces than the three existing solutions. We not only model the search interfaces and extract attributes, but also identify more useful information implicitly existing in search interface schemas,

such as domain type, value type, and relationships between domain elements, for each attribute. Such information is quite important for precisely representing a search interface as well as for other purposes such as automatic interface integration and query mapping/submission.

We also note that Ontobuilder [4] supports the automatic extraction of ontologies from Web search interfaces. However, we found out that the ontologies are only limited to the properties of labels and elements themselves, but other semantic/meta information such as value type is not identified. Moreover, the details of their extraction method are not publicly available. Kushmerick [13] uses a Bayesian learning approach to classify Web search forms to a specific domain and predict a label for each element. However, the learning process involves extensive manual efforts and many heuristic assumptions. Also, the work does not address how to model and extract semantic/meta information on search interfaces.

8. Summary

In order to better understand and utilize Web databases, it is critical to first understand their Web search interfaces. As Web search interfaces are designed autonomously in semi-structured HTML, automatically understanding their contents is a challenging task. In this paper, we proposed a schema model for representing form-based search interfaces of database-driven search engines. This model precisely captures the relationships between elements and labels in terms of logical attributes, as well as a significant amount of semantic/meta information on this type of search interfaces. We also presented our techniques for automatically constructing the schema for any search interface, including grouping labels and elements into logical attributes and deriving the semantic/meta information of attributes. Such a comprehensive study of Web search interfaces has rarely been conducted in the literature. Our experimental results based on 317 Web database interfaces from 3 different datasets in 13 different application domains showed that the proposed techniques can achieve very high accuracy in automatically extracting form information. Our techniques should be easily applicable to new domains, as evidenced by the result of the additional experiments using search interfaces of two new datasets.

Acknowledgement: This work is supported by the following grants from NSF: IIS-0414981, IIS-0414939 and CNS-0454298. We also would like to express our gratitude to the anonymous reviewers of our manuscript for their valuable suggestions to improve this paper.

References

[1] S. Bergamaschi, S. Castano, M. Vincini, D. Beneventano. Semantic Integration of Heterogeneous Information Sources. *Data & Knowledge Engineering*, 36: 215-249, 2001.

[2] K. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured Databases on the Web: Observations and Implications. *SIGMOD Record*, 33(3), September 2004.

[3] K. Chang and H. Garcia-Molina. Mind Your Vocabulary: Query Mapping Across Heterogeneous Information Sources. In *SIGMOD Conference*, 1999.

[4] A. Gal, G. Modica and H. Jamil. OntoBuilder: Fully Automatic Extraction and Consolidation of Ontologies from Web Sources. In *ICDE Conference*, 2004.

[5] J. Han, and M. Kamber. Data Mining Concepts and Techniques. Morgan Kaufmann Publishers. August 2000.

[6] B. He and K. Chang. Statistical Schema Matching across Web Query Interfaces. In *SIGMOD Conference*, 2003.

[7] B. He, T. Tao and K. Chang, Organizing Structured Web Sources by Query Schemas: A Clustering Approach, In *CIKM Conference*, 2004.

[8] H. He, W. Meng, C. Yu, and Z. Wu. WISE-Integrator: An Automatic Integrator of Web Search Interfaces for E-commerce. In *VLDB Conference*, 2003.

[9] H. He, W. Meng, C. Yu, and Z. Wu. Constructing Interface Schemas for Search Interfaces of Web Databases. In *WISE Conference*, 2005.

[10] HTML4: http://www.w3.org/TR/html401/

[11] O. Kaljuvee, O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Efficient Web Form Entry on PDAs. In *WWW Conference*, 2000.

[12] R. Kohavi, B. Becker, and D. Sommerfield. Improving simple Bayes. In *ECML Conference*, 1997.

[13] N. Kushmerick. Learning to Invoke Web Forms. In *ODBASE Conference*, 2003.

[14] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB Conference*, 1996.

[15] Y. Lu, H. He, Q. Peng, W. Meng, and C. Yu. Clustering E-Commerce Search Engines based on Their Search Interface Pages Using WISE-Cluster. *Data & Knowledge Engineering (DKE)*, 2006.

[16] MetaQuerier:http://metaquerier.cs.uiuc.edu/forme

[17] Q. Peng, W. Meng, H. He, and C. Yu. WISE-Cluster: Clustering E-Commerce Search Engines Automatically. In *WIDM workshop*, 2004.

[18] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In *VLDB Conference*, 2001.

[19] J. Wang and F.H. Lochovsky. Data Extraction and Label Assignment for Web Databases. In *WWW Conference*, 2003.

[20] WordNet: http://www.cogsci.princeton.edu

[21] W. Wu, C. Yu, A. Doan, and W. Meng. An Interactive Clustering-based Approach to Integrating Source Query interfaces on the Deep Web**.** In *SIGMOD Conference*, 2004.

[22] Z. Zhang, B. He, and K. Chang. Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. In *SIGMOD Conference*, 2004.

[23] Z. Zhang, B. He, and K. Chang. Light-weight Domain-based Form Assistant: Querying Web Databases On the Fly. In VLDB Conference, 2005.