Constructing Interface Schemas for Search Interfaces of Web Databases

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

[1] Dept. of Computer Science, SUNY at Binghamton, Binghamton, NY 13902
{haihe, meng}@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., search 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 on the search interface. Many Web applications, such as Web database integration and deep Web crawling, require the construction of the schemas. In this paper, we introduce a schema model for complex search interfaces, and present a tool (WISE-*i*Extractor) for automatically extracting and deriving all the needed information to construct the schemas. Our experimental results on real search interfaces indicate that this tool is highly effective.

1 Introduction

With the explosive growth of the Internet, more and more databases supported by relational databases systems have become Web accessible through form-based search interfaces. For example, amazon.com uses the search interface as shown in Figure 1 to search its book database. The search interfaces of Web databases typically contain some form control *elements*, such as *textbox*, *radio button*, *checkbox* and *selection list*, which allow users to enter search information. A *label* (a descriptive text) 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 to Web databases.

In order to understand Web databases and obtain the information from them, the first essential step is to understand search interfaces [2]. As Web search interfaces are designed in HTML for human users to use, their schemas are not formally defined. First of all, semantically related labels and elements are scattered in HTML text and the formal associations of these labels and elements do not exist. Moreover, search interfaces are heterogeneous in their contents, presentation styles and query capabilities. Therefore, it is difficult to automatically identify the attributes of each interface and to

fully understand the semantics of the information related to the attributes. Although some work on extracting search interfaces has been reported [4, 9, 10, 13, 16], they are highly inadequate as they focus on only labels and elements.

In fact, beyond 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 "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 another example, in Figure 2 the values in the selection list (Title, Author, Keyword and Publisher) are in fact attribute names, and only one of them can be used to submit a query at a time (thus they are called *exclusive attributes* in this paper).

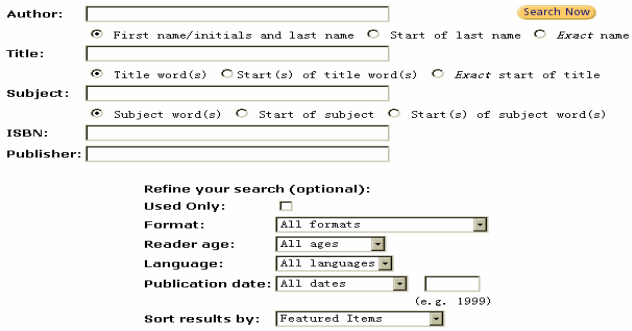

Fig. 1. The book search interface of amazon.com

Fig. 2. Example search interfaces containing exclusive attributes

Many Web applications are related to Web databases and require the contents of search interfaces of the Web databases to be understood and properly organized for computer processing. These applications include schema matching across multiple interfaces [1, 5, 7, 14], unified interface generation [7], deep Web crawling [13], programmatically interfacing with Web databases [3, 11], clustering/classifying Web databases [6, 12], and annotating the returned results of Web databases [15].

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 reported. Second, we present our method for automatically constructing the schema to make the "*hidden*" information on the search interfaces *explicit* and *machine understandable*. For this purpose, we introduce WISE-*i*Extractor whose main goals are: 1) *Attribute Extraction*, i.e., to associate semantically related labels and elements to form logical attributes and identify an appropriate label for each attribute; 2) *Attribute Analysis*, i.e., to reveal the "*hidden*" information to help understand the search interface.

This paper has the following contributions:

1) An interface schema model is proposed 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. This model is an extension of the model described in [8].
2) A practical solution is provided to automatically identify logical attributes on search interfaces. This solution consists of two novel components: (1) LEX, a *l*ayout-*e*xpression-based form e*x*traction 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) 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 how to practically construct the schema, including attribute extraction and attribute analysis. The experimental results are reported in Section 5. In Section 6, we review related work. Finally, Section 7 contains the conclusions.

2 Schema Model for Representing Search Interfaces

In this section, we present a schema model for Web search interfaces that captures rich semantic information. For completeness of interface representation, the model also includes syntactic information of a search interface, such as element name and element type. In our model, an interface is represented as $F = (S, \{A_1, A_2, \ldots, A_n\}, C_f)$, where S is the site information associated with the form, such as the site URL, the server name and the HTTP communication method, $\{A_1, A_2, \ldots, A_n\}$ 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). Each A_i is represented as $(L, P, DT, DF, VT, U, R_e, \{E_j, E_{j+1}, \ldots, E_k\}, C_a)$, 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, $\{E_j, E_{j+1}, \ldots, E_k\}$ 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. Each domain element E_i is itself represented as (L_e, N, F_e, V, DV), where L_e is the element label (possibly empty), N is the name of E_i, F_e is the format (e.g., textbox, selection list, checkbox and radio button), V is the set of values of E_i, and DV is the default value of E_i (possibly null).

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.

Relationships of elements:
In a search form, an attribute may have multiple elements and they may be related differently. Generally, there exist four types of element relationships: *range type*, *part type*, *group type* and *constraint type*. For example, in Figure 3, the relationship between the elements of "Publication Year" is of *range type* (for specifying range query

conditions); that 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 a *constraint type* of "Title keywords".

When an attribute has multiple associated elements, they shall be classified 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 specify some constraints on 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 relationship between the elements of each attribute helps interface schema integration [7, 14] and query mapping [3, 11]. As an example for 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" on a *mediated interface*, the query translation should map the query value to the domain element of "Title keywords" instead of the constraint element "Exact phrase".

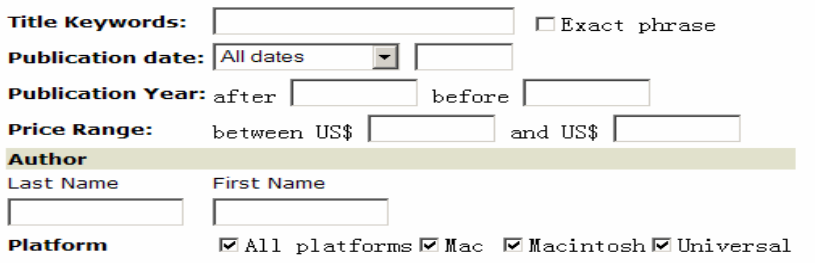

Fig. 3. Examples of element relationship type

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.

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 is important for successful query mapping and submission. Generally, there are four possibilities:

1) *Conjunctive*. All the attributes are combined through the "*and*" Boolean logic operator, meaning that all specified conditions must be satisfied at the same time.
2) *Disjunctive*. All the attributes are combined through the "*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 any given time. In Figure 2 the attribute names appear in a selection list (or a group of radio buttons) and only one attribute can be used at a time to form 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.

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 (e.g., [1, 7, 11, 12, 13, 15]) 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.

3.1 Interface Expression

On a search interface, labels and elements are *visually* arranged in one or more rows. To approximately capture the *visual layout* of these labels and elements, we introduce the concept of *interface expression* (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' for an element, and '|' for a row delimiter that starts a new physical row.

For example, the search interface in Figure 1 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' for the textbox following the label "Author", the first '|' for the start of a new row on the interface, the following three 'e's for the three radio buttons below the textbox. The remaining items can be understood in a similar manner.

The IEXP is obtained when extracting individual labels and elements from a search interface (see Section 3.2 below). It 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 elements and the actual implementations of laying out labels and elements. Our method for automatic extraction of attributes will rely on this expression.

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 an 'e' will be appended to the IEXP (initially it is empty) accordingly. Each element contains its values (if available). Four types of input elements are considered: *textbox*, *selection list*, *checkbox* and *radio button*. When a row delimiter like "
", "<P>" or "</TR>" is encountered, a '|' is appended to the IEXP. This process continues until the "</FORM>" tag is encountered. In this process, some texts that are too long will be discarded. After this process, all labels and elements in the search interface are extracted, and the corresponding IEXP is also constructed.

3.3 Identifying the Names of Exclusive Attributes

Exclusive attributes are those 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 Figures 2).

By our observation, exclusive attributes appear frequently on real Web search interfaces. Among the 184 real search interfaces we collected, more than 34% of the in-

terfaces in the books, movies and music domains have exclusive attributes. A significant flaw of existing approaches [4, 9, 10, 13, 16] is that they do not extract them.

B. He et al [5] reported their discovery that the vocabulary of schemas in a domain stabilizes at a reasonably small size even though the number of search interfaces in the domain can be very large. We also observed that the names of exclusive attributes are often the *most commonly used attribute names* of a domain. Based on these, we propose a novel and simple statistical approach to tackle the problem of identifying exclusive attributes. The basic idea of our approach is the construction of a vocabulary for a domain by considering multiple interfaces in the same domain at the same time (Web interfaces can be clustered first such that each cluster contains the interfaces from the same domain [6, 12]).

To construct the vocabulary, we compute the **L**abel **I**nterface **F**requency (*LIF*) of each extracted label of all considered search interfaces of a domain, which is the number of search interfaces of the domain that contain the label. Then only labels whose $LIF \geq ft$ are selected to construct the *common schema vocabulary* of the domain, where ft is computed by $ft = p*n$, 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.

Using this schema vocabulary, we find the search interfaces that contain some elements whose values are words in the vocabulary, and then extract the attribute names from these elements. Two patterns widely used in search interfaces as shown in Figure 2 are applied to identify such search interfaces: one pattern consists of a group of *radio buttons* and a *textbox*; the other consists of a *selection list* and a *textbox*. If enough values in the element(s) (selection list or radio buttons) are found to be contained in the 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. After exclusive attributes are identified from the interface, the IEXP of the interface needs to be adjusted accordingly such that other attributes (if any) would not be affected for grouping in the next step (Section 3.4).

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, 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 extraction technique (LEX) is developed. The basic idea of LEX is as follows. The IEXP of an interface 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 within 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 then groups *e* with this text. The process continues until all the elements in the IEXP are processed. The reason for considering two rows above the current row of an element is that related labels and elements are generally never laid out sparsely on a search interface.

During the above process, 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.

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.* An attribute label often ends with a colon, while other texts do not.
2) *Textual similarity of element name and text.* An attribute label and its elements may have some words/characters in common.
3) *Distance of the element and text.* An attribute label and its elements usually are close to each other on a search interface. Such relationships are also expressed in the IEXP. If there exist multiple texts, 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 frequently aligned vertically. Thus, when we consider two rows above the current row of the element, we check if the element and the text are aligned vertically.
5) *Priority of the current row.* Although we consider two extra rows above the current row of an element, the current row is given higher priority because in most cases an attribute label appears in the same row as its elements.

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 maintains its elements. In this section, we discuss how to derive semantic information about attributes beyond labels and elements themselves as described in Section 2.

4.1 Differentiating Elements

As mentioned earlier, differentiating domain elements and constraint elements is important for precisely understanding the composition of attributes. Our observation indicates that domain elements often precede constraint elements either horizontally or vertically, and that constraint elements often appear in the form of a group of radio buttons (as shown in Figure 1) or a group of checkboxes. Based on the observation, a simple two-step method can be used to differentiate domain elements and constraint elements. First, identify the attributes whose elements are all radio buttons or checkboxes. Such attributes are considered to have only domain elements. Next, other attributes that may contain both domain elements and constraint elements can be easily recognized by following the regulations of the observation.

4.2 Identifying the Relationships and Semantics of Domain Elements

If an attribute has multiple *domain elements*, we identify their relationship and semantics. In our model, three types of relationships for multiple domain elements are defined and they are *group*, *range* and *part*.

We identify the group type by checking whether all the elements of the attribute are checkboxes or radio buttons and whether there are at least *two* (group implies at least two) checkboxes/radio buttons. If both are true, group type is recognized.

Range type is recognized by checking labels against certain domain-independent keywords and patterns that are widely used to represent range semantics. If some keywords and a pattern are found in the labels, the elements are considered to be of range type. For example, "between-and" and "from-to" are widely used range value patterns. At the same time, we also identify the semantic meaning of each element. For the "between-and" example, the element with "between" represents the lower bound of the range and the other with "and" represents the upper bound. Such semantics are useful for query mapping and schema matching.

The elements that are not of range type or group type will be treated as part type by default. The semantic meaning of each element involved in a part type relationship should also be identified whenever possible. Our approach is that we build a thesaurus of some commonly used concepts, such as date, time, name, address, etc. For example, if an attribute has three selection lists, each of them has its own set of values for "month", "day" and "year", respectively; by matching the patterns of values with the existing concepts, the system can derive what each element represents semantically.

4.3 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*. Note that these meta-data are for *domain elements* of each attribute only. Deriving meta-information is relatively straightforward as described below.

Domain type: Domain type indicates how many values can be specified on an attribute for each query. Four domain types are defined in our model: *range*, *finite* (with finite values but no range semantics), *infinite* (with possibly unlimited number of values, e.g., textbox, but no range semantics) and *Boolean*. The range domain type is addressed first. Given an attribute, if the relationship of its domain elements is of range type, then the domain type of the attribute is *range*. Otherwise, if a certain number of range keywords or a *range* pattern exists in the attribute label and the values of each element, the domain type of the attribute is recognized to be *range*. For example, keywords "less than" indicates a range; "from $10 to $30" satisfies a range pattern. If the domain type of the attribute is not a range type, it is then *infinite* if it involves a textbox. For the *Boolean* type, the attribute has just a single checkbox, and this checkbox is usually used to mark a yes-or-no selection. An example of such an attribute is "Used only" in Figure 1. *Boolean* type may be considered as a special case of the finite type because an attribute of Boolean type takes two (finite) possible values conceptually. In our model, *Boolean* type is separated from the regular *finite* type as this

separation makes the Boolean property of the attribute more obvious and precise. If the attribute is not *Boolean* type, then *finite* type is set by default.

Value type: Value types currently defined in our model include *date*, *time*, *datetime*, *currency*, *id*, *number* and *char*, but more types could be added. To identify *date*, *time*, *datetime*, *currency* and *id*, commonly used keywords and patterns can be employed. For example, keyword "date" and value "12/01/2005" imply a date value type; keyword "morning" and value "10:30 AM" imply a time value type; "$" implies a currency value type. Value patterns can be expressed as regular expressions. If an attribute is not one of these five value types, the value type is declared to be *number* if the values of each element are numeric; otherwise the value type is *char*.

Default value: Default values in many cases indicate some semantics of the attributes. For example, in Figure 1 the attribute "Reader age" has a default value "all ages". Identifying default value is easy because it is always marked as "checked" or "selected" in an element.

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 may use "USD" as the unit of its "Price" attribute, while another may use "CAD" for its "Price" attribute. Identifying the correct units associated with attribute values can help understand attributes. Not all attributes have units. The unit information may be contained in labels or values for some attributes, which is easy to identify. Some resources available on the Web (e.g., the dictionary of units of measurement at www.unc.edu/~rowlett/units/) can also be utilized for helping identify the unit of an attribute.

4.4 Identifying the Logic Relationships

As the *exclusive* relationship of attributes is already addressed in Section 3.3, we focus on the other three relationships in the section.

Clearly, if two attributes are in a conjunctive relationship, the number of results returned from the search engine 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 search engine and 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 search engines 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. Most interfaces have no explicit logic operators or exclusive attributes (e.g., Figure 1), 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 (by our experiments, 180 out of 184 forms used in our dataset are correctly identified).

5 Experiments

We have implemented an operational prototype of WISE-*i*Extractor using Java. WISE-*i*Extractor takes as input raw HTML pages containing search interfaces (search forms containing scripts such as Javascript or Vbscript are not considered) in the same domain, and outputs the schemas of these interfaces in XML format for use by other applications (say schema matching). The entire extraction process is fully automated.

5.1 Datasets

To evaluate the interface extraction technique proposed in this paper, we selected various numbers of 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 forms). We also use two independently collected datasets from the works [15, 16] for performance comparison.

5.2 Form Extraction Results

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*.

Table 1. Element-level accuracy of LEX

Domain	#elements	# elements whose label is correctly identified by LEX	Accuracy
Books	484	473	97.7%
Electronics	191	185	96.9%
Games	72	72	100%
Movies	275	258	93.8%
Music	240	237	98.8%
Toys	62	62	100%
Watches	258	258	100%
Overall	1582	1545	97.7%

Element level: A label is correctly extracted for an element if it matches the manually identified label. Table 1 shows the experimental results of our method on the element level accuracy. The overall accuracy of 1582 elements from 184 forms is over 97%. The element level accuracy only considers a single element but not the whole attribute.

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. Table 2 shows the experimental results for attribute-level accuracy, where "#Man" denotes the number of attributes identified manually; "#LEX" denotes the number of attributes correctly extracted by LEX. The overall accuracy of 7 domains is over 95%. 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*). 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".

Table 2. Attribute-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	**95.6%**	27	12	10

Table 3. Usefulness measurement of LEX heuristics

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

Effectiveness of LEX heuristics:

LEX uses five different heuristics to discover the association between elements and labels (see Section 3.4). It would be interesting to find out the impact of each individual heuristic on the accuracy of LEX. To do so, we remove a particular heuristic and use the remaining heuristics to group related labels and elements. The relative importance of the remaining heuristics is not changed. Table 3 shows the experiment results. Column headed by "Rh#" indicates the number of attributes that are negatively affected when the #th heuristic listed in Section 3.4 is not used, compared to the results of using all heuristics. 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 reasonable to only use distance to associate labels and elements. The results in Table 3 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 short, 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 [16, 17] (the work is closely related to ours). In this dataset, there are 106 search forms from 9 different domains (airfares, books, autos, jobs, music, movies, hotels, carRental and realestate). We run WISE-*i*Extractor on this dataset without making *any change* to WISE-*i*Extractor. The overall attribute-level extraction accuracy of our method is 92.5%, while the accuracy obtained based on the method in [16] is 85.9%**.** Clearly, our method is significantly more accurate than the method reported in [16]. The second dataset is from DeLa [15] that contains 27 search forms from 3 domains (books, cars and jobs). The overall attribute-level accuracy of our method on this dataset is 92.7%.

Effectiveness of extracting exclusive attributes:
We also evaluate the effectiveness of the statistical approach for identifying and extracting exclusive attributes. Table 4 shows the experimental results, where "#IE" denotes the number of search interfaces that have such attributes, "#Prog" for the number of such search interfaces that are correctly identified by our approach, "#Latts" for how many such attributes exist in those interfaces, "#Eatts" for how many such attributes are correctly extracted. Note that the "#LEX" in Table 2 includes the "#Eatts", which means without using this approach, the "#LEX" of Table 2 would lose "#Eatts" in each corresponding domain, and thus the accuracy of Table 2 would decrease accordingly. We can see that our approach is very effective for recognizing such attributes that appear in the elements.

Table 4. Accuracy of identifying & extracting exclusive attr.

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

5.3 Obtaining Attribute Meta-information

To evaluate the accuracy of obtaining attribute meta-information, we consider only attributes that are correctly identified by LEX. Table 5 shows our experimental results for each domain. "DT" denotes domain type, "VT" for value type, "U" for unit, "ER" for element relationship, "DDC" for differentiation of domain elements and constraint elements. For example, for the book domain, out of the 351 attributes considered, the domain types of 347 attributes are correctly identified. The results show that nearly all needed meta-information can be correctly identified using the proposed methods.

Table 5. Accuracy of deriving meta-information of attr.

Domain	# LEX	DT	VT	U	ER	DDC
Books	351	347	349	341	347	343
Electronics	137	136	136	137	137	137
Games	63	63	63	61	63	63
Movies	178	171	171	175	176	177
Music	196	191	194	194	192	196
Toys	59	58	58	59	59	59
Watches	84	79	84	84	84	84

6 Related Work

The works reported in [9, 13, 16] are the most related to ours. The major differences between the three works and our LEX approach are as follows: (1) The approaches in [9, 13] are not completely *attribute-oriented*, i.e., they find labels for elements but not for attributes. In contrast, LEX is *attribute-oriented*. (2) Even though a more formal framework (it views search interfaces as a visual language) is used in [16], the grammar rules are actually defined in advance based on extensive manual observations of a large number of search interfaces. More importantly, as reported in Section 5, our experiments on the same dataset indicate that our method outperforms the method in [16] significantly. (3) The three works do not 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 relationships among elements and value type, for each attribute. Such information is quite important for precisely representing and understanding a search interface.

We also note that Ontobuilder [4] supports the automatic extraction of ontologies from Web search interfaces. However, the ontologies are limited to the properties of labels and elements themselves, other semantic/meta information such as data type is not revealed. Kushmerick [10] uses a Bayesian learning approach to classify Web search forms to a specific domain and predict a label for each element. However, the work does not address semantic/meta information on search interfaces.

7 Conclusions

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. The information in this model can be used to support many important Web applications.

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 such attributes. Such a comprehensive study of Web search interfaces has rarely been conducted in the literature. Our extensive experimental results showed that our techniques can achieve very high accuracy in automatically extracting form information. Although our solutions involve significant heuristics (just like other solutions [9, 13, 16]), our insight into Web search interfaces is unique and deeper.

In the future, we plan to apply data mining approaches to attribute extraction and analysis. We will also study the problem of constructing a complete schema for Web databases by combining the interface schema and the data schema reflected in the search result.

Acknowledgement: This work is supported by the following grants from NSF: IIS-0208574, IIS-0208434, IIS-0414981, IIS-0414939 and CNS-0454298.

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. B. He and K. Chang. Statistical Schema Matching across Web Query Interfaces. In *SIGMOD Conference*, 2003.
6. B. He, T. Tao, and K. Chang. Clustering Structured Web Sources: a Schema-based, Model-Differentiation Approach. In *EDBT-ClustWeb workshop* 2004.
7. 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.
8. H. He, W. Meng, C. Yu, and Z. Wu. Automatic Extraction of Web Search Interfaces for Interface Schema Integration. In *WWW Conference*, 2004.
9. O. Kaljuvee, O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Efficient Web Form Entry on PDAs. In *WWW Conference*, 2000.
10. N. Kushmerick. Learning to Invoke Web Forms. In *ODBASE Conference*, 2003.
11. A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB Conference*, 1996.
12. Q. Peng, W. Meng, H. He, and C. Yu. WISE-Cluster: Clustering E-Commerce Search Engines Automatically. In *WIDM workshop*, 2004.
13. S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In *VLDB Conference*, 2001.
14. 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.
15. J. Wang and F.H. Lochovsky. Data Extraction and Label Assignment for Web Databases. In *WWW Conference*, 2003.
16. Z. Zhang, B. He, and K. Chang. Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. In *SIGMOD Conference*, 2004.
17. MetaQuerier:http://metaquerier.cs.uiuc.edu/formex