Meaningful Labeling of Integrated Query Interfaces

Eduard C. Dragut
Computer Science Dept.
Univ. of Illinois at Chicago
edragut@cs.uic.com

Clement Yu
Computer Science Dept.
Univ. of Illinois at Chicago
yu@cs.uic.com

Weiyi Meng
Computer Science Dept.
SUNY at Binghamton
meng@cs.binghamton.edu

ABSTRACT

The contents of Web databases are accessed through queries formulated on complex user interfaces. In many domains of interest (e.g. Auto) users are interested in obtaining information from alternative sources. Thus, they have to access many individual Web databases via query interfaces. We aim to construct automatically a *well-designed* query interface that integrates a set of interfaces in the same domain. This will permit users to access information uniformly from multiple sources. Earlier research in this area includes matching attributes across multiple query interfaces in the same domain and grouping related attributes. In this paper, we investigate the naming of the attributes in the integrated query interface. We provide a set of properties which are required in order to have consistent labels for the attributes within an integrated interface so that users have no difficulty in understanding it. Based on these properties, we design algorithms to systematically label the attributes. Experimental results on seven domains validate our theoretical study. In the process of naming attributes, a set of logical inference rules among the textual labels is discovered. These inferences are also likely to be applicable to other integration problems sensitive to naming: e.g., HTML forms, HTML tables or concept hierarchies in the semantic Web.

1. INTRODUCTION

In recent years there is increased awareness regarding the tremendous amount of information available on the Web and especially among online databases. These databases provide dynamic query-based data access through *query interfaces*, instead of static URL links. A user seeking desired information in such repositories needs to be aware of these sources (a recent survey [6] estimated 450,000 online databases), then to learn their query interface capabilities and to proceed with the actual retrieval. As an example, a user buying a car is often interested in probing alternative sources for better price. Given the large number of alternative sources a user has to access numerous sites to achieve lowest price. It is unrealistic to expect for a user to access a large number of pertinent online sources. Therefore, one step towards meeting the user's desire is the construction of an integrated query interface that allows uniform access to disparate relevant sources. Meaningful user interface creation is an important issue underlined also by the Florida "butterfly" ballots incident in US Election 2000. This is still a vivid proof that ill-designed forms make it difficult even for human voters to simply associate candidates with their punch holes.

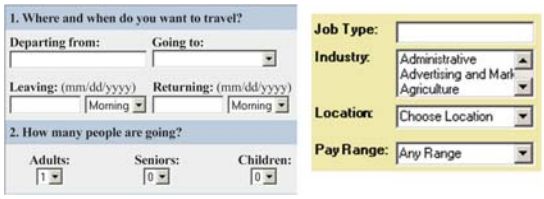

Figure 1: Example of naming problems.

The problem is to construct automatically a *well-designed* unified query interface which contains all or most significant distinct fields of source interfaces. The goal of our work is to lay down a set of properties that allows for a precise characterization of an integrated interface. Much like in the other data models used in the database community (e.g. relational, XML) for a certain domain there is no unique well-designed query interface. In order to distinguish "well" from "bad" constructed unified interfaces a formalism (i.e. a set of desirable properties) is needed. Our study of individual query interfaces in various domains revealed the presence of three major components that contribute to a well-designed query interface. The first component is *structural* [8]; the elements of query interfaces are organized in groups (logical units) of related elements [8, 26] so that semantically related elements are placed in close vicinity. For example, Adults, Seniors, Children of the interface shown in Figure 1 are placed together. In addition, multiple related groups of fields are organized in super-groups (e.g. Where and when do you want to travel?). This bottom-up characterization leads to a hierarchical structure for interfaces (see Figure 2, Vacations tree) [8, 24], where a leaf in the tree corresponds to a field in the interface, an internal node corresponds to a (super)group of fields and the order among the sibling nodes within the tree resembles the order of fields in the interface. The second is *lexical*, i.e., the labels assigned to their elements are carefully chosen so as to convey both the meaning of each individual element and to underline the hierarchical organization of the fields (e.g. the three fields together with the parent field How many people are going?). The third is *the set of instances*; designers rely on instances to further help users grasp the role of elements within a query interface. For example, the attribute Going to in Figure 1 is equipped with a list of predefined values.

The quality of the integrated interface has to be judged along these three components. We already addressed the

VLDB '06, September 12-15, 2006, Seoul, Korea.

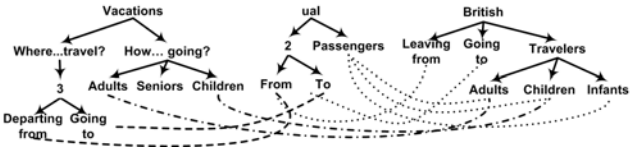

Figure 2: Schema trees and their correspondences.

structural part [8]; we described the construction of a unified query interface that preserves the structural information in disparate query interfaces of the same domain. Since this work draws on the concepts developed there we will briefly introduce them in the following section. Even though the third component is not always present [23], whenever it is we use it to improve the lexical component of the integrated interface. The computation of the domains of the fields on the unified interface is described in some detail in [12] and it will not be covered here. *The second component is the chief focus of this paper.* A key problem addressed in this work is consistent labeling of fields in the same group. As it can be observed in the query interface of Figure 1 between the labels of the fields in the same group there are certain relationships. For instance, Adults, Seniors and Children are all plurals, whereas Departing from and Going to are gerund followed by a preposition. In addition to this uniformity property among the labels of a group the global interface faces an additional problem: fields may come from different sources. Thus, a group of fields within the unified interface might not correspond to any group in a single interface, making even harder the task of uniform assignment of labels to these fields. For example, consider that the global interface is just as the one shown in our example but it has an additional field next to Children, namely, a field denoting the *infants* whose label can be drawn from among Infant, Infants, and Number of Infants. Clearly, the best label for such a field is Infants as it preserves the existing homogeneity within the interface (i.e. all plurals). In addition, there are also semantic ambiguity problems. Consider the query interface in the Job domain (Figure 1, on the right). For the sake of this example suppose the integrated interface of this domain is as shown in the figure except for an additional field denoting job preferences, i.e. part-time, full-time, etc, whose label needs to be selected from among Job Type, Type of Job and Job Preferences. Here the first two labels would not be appropriate, since they are essentially the same as another field Job Type.

The objective of our research is to provide a systematic way to label fields in the integrated user interface so that (i) the labels of the fields within a group are consistent and (ii) the labels of internal nodes in the global interface are consistent with respect to themselves and to the leaf nodes. The former is called *horizontal consistency*, while the latter is called *vertical consistency*.

The contributions of this paper are:

- the most comprehensive treatment of naming attributes automatically within an integrated interface;
- a set of logical inference rules among text attributes with broad applicability;
- extensive experiments on seven domains (see Section 7) which show that in general in each domain, the integrated interface is natural and easy to understand.

2. BACKGROUND

Data in searchable databases are accessible through form-based search interfaces. The basic building blocks of these forms are: text boxes, selection lists, radio buttons, and check boxes. They are generically called *fields*. Fields have a number of properties. Typically, they have *labels*, describing their purpose to the user. Some fields may also have *predefined domains* (e.g. selection lists). Labels and instances are of particular interest in this work. Hereafter, we will interchangeably use *names* and *labels* to refer to labels.

Table 1: The clusters of the mapping in Figure 2

c_Depart	c_Dest	c_Senior	c_Adult	c_Child	c_Infant
Departing from	Going to	Seniors	Adults	Children	
From	To		Adults	Children	Infants
Leaving from	Going to	Passengers	Passengers	Passengers	Passengers

The problem addressed here is part of a larger system we envision for lifting the burden from users seeking information on disparate Deep Web sources. The solution consists of a number of components. First, query interfaces are identified, extracted from the relevant Web pages [11, 26] and clustered into different classes based on the type of products or services they offer (i.e., Airline, Job) [18]. Second, the fields in different interfaces in the same domain are matched [7, 10, 23]. Third, the interfaces of the same class are integrated into a unified interface [8, 12]. This step corresponds to the structural component. As noted in [16, 20], in general, naming (the lexical component) is not part of a merge algorithm semantics. Thus, since query interfaces are highly sensitive to the lexical component the next step is *the meaningful labeling of the unified query interface*. This is the focus of this paper. Fourth, a global query submitted against the integrated user interface is translated into subqueries against individual sources [5, 13, 27]. Finally, returned data by individual sites needs to be correctly extracted and the results ranked in descending order of desirability (e.g. price).

2.1 Mapping Structure

Among the inputs of our problem is a mapping globally characterizing the semantic correspondences between equivalent fields in the query interfaces. The mapping is organized in *clusters* [24] that record 1:1 and 1:m matchings of fields. For a given domain, *a cluster consists of all the fields (leaves) of different schemas that are semantically equivalent*. An example of clusters is given in Table 1. The table shows the clusters of the (fragments) schema trees depicted in Figure 2. The fields of the three schemas denoting "adults" are all placed in the cluster c_Adult. For a schema that does not have a field in a cluster a *null* entry is added, marked through an empty entry in the table. A field that matches multiple fields in different clusters is placed in each of the clusters where there is a field it matches (e.g., Passengers). 1:m correspondences lead to granularity mismatches between schemas. To have a uniform representation of the fields within all schemas they need to be reduced to 1:1 correspondences [8]. This is done by expanding the leaf node on the one side of 1:m mapping into an internal node whose children have 1:1 correspondence to the leaf nodes on the many side. In Figure 2 the field Passengers becomes an internal node with four children which have 1:1 correspondences with Adults, Seniors, Children, and Infants. Consequently, the label "Passengers" becomes a candidate label for an internal node and it is removed from all the clusters it occurs.

In this work we assume the semantic relationships between the attributes of the interfaces in the same domain have been already computed. The actual computation of the clusters is defined and analyzed in [10, 24, 23].

2.2 Fields Grouping

Recent works on the problem of integrating query inter-

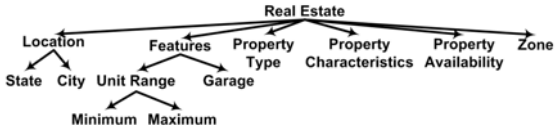

Figure 3: Real Estate unified interface (fragment).

faces on the Deep Web [8, 10, 26] have observed that fields on these interfaces are laid out so as to underline the semantic meaning not only of the fields themselves but also of groups of fields as a whole (e.g. Adults, Children, Infants). In [8], a group is a semantic unit of fields and each group is disjoint from each other. For example, in the airline domain fields denoting the type of passengers (e.g. seniors, children) are never intermixed with the fields about departure date (e.g. month, day, year).

2.3 Merge Algorithm (Brief Description)

Although there are many merge algorithms [4, 12, 15, 20, 22] that might potentially be relevant for the discussion at hand we choose the formalism laid by [8]. The reasons are that (1) both the individual query interfaces and the integrated interface are abstracted as ordered schema trees; (2) the outputted integrated schema tree has the following desirable properties: *all ancestor-descendant relationships in individual schema trees are preserved* (under certain constraints) and *the grouping constraints are satisfied as much as possible*. As revealed later, these properties form the foundation for an accurate labeling of the elements of an integrated interface.

3. PRELIMINARIES

The input of the naming algorithm consists of a set of query interfaces, $\mathcal{QI}$, in a given domain of interest, the set of clusters, $\mathcal{C}$, that globally characterizes the semantic correspondences between equivalent fields in these query interfaces, the set of groups, $\mathcal{G}$, which partition the fields into logical units, and the schema tree of the integrated query interface, $\mathcal{I}$. Based on the placement of the fields in the integrated schema tree, the set of clusters is divided into three disjoint partitions: the set of clusters that belong to some group, denoted $\mathcal{C}_{groups}$, the set of clusters that are children of the root, denoted $\mathcal{C}_{root}$ and the set of clusters that are isolated children of internal nodes, other than the root, denoted $\mathcal{C}_{int}$. We exemplify them on the integrated schema tree fragment of the Real Estate domain drawn in Figure 3. $\mathcal{C}_{groups}$ consists of {State, City}, {Minimum, Maximum}, $\mathcal{C}_{int}$ = {Garage} and $\mathcal{C}_{root}$ = {Property Type, Property Characteristics, Property Availability, Zone}.

We assume that an interface is reasonably well-designed so that users have little or no problem understanding it. This is accomplished by having (1) consistent labels for the fields within a group and (2) consistent labels for internal nodes which have ancestor-descendant relationship. The labels of the internal nodes also have to be consistent to those of their descendant leaves (fields). Consistency of type (1) is exemplified by (Adults, Seniors, Children) (Figure 1), while consistency of type (2) can be exemplified by How many people are going? (Figure 1) as parent of the previous three fields. The problem we address in this paper is to extend the two types of consistency to the integrated interface.

3.1 Normalization

In our naming algorithm a pre-processing step is the normalization of the labels of the attributes. This is a 2-step process. In the first phase attached comments are removed (e.g. Adults (18-64) becomes Adults) and all characters that are not alphanumeric are replaced by a space character (e.g. Price $ becomes Price). The new label is used to perform plain string comparisons (see Definition 1). The second step, which is used to derive more complex relationships between labels (next section), involves more complicated operations: (1) tokenize labels consisting of multiple words, (2) convert each label string to its lower case equivalent, (3) stem extracted tokens using the standard Porter stemming algorithm [19], (4) retrieve the base form of each token using WordNet [9] and remove stop words.

3.2 Semantic Rules on Content Words

In many practical instances we need to establish semantic relationships between labels of the attributes. For labels consisting of one token these relationships are easy to establish using WordNet. However, for labels with more than one content word WordNet cannot be employed directly. For example, to detect that Area of Study and Field of Work are synonyms more complex techniques need to be devised. Thus, it is preferable to treat labels in a more systematic manner, e.g. as n-dimensional vectors or set of tokens.

In the second normalization step each field will be represented by a set of content words of its label. For instance, {area, study} corresponds to the label Area of Study. This view allows us to implement a set of inference rules on labels that will be extensively used in the naming process. In this work we restrict our attention to determining the following semantic relationships between two labels: equality, synonymy, hyponymy/hypernymy. They will be computed based on the relationships between the tokens of the two labels as given by WordNet.

DEFINITION 1. *Given two labels* A *and* B *along with their set of content words representation* $A_{cw} = \{a_1, ..., a_n\}$ *and* $B_{cw} = \{b_1, ..., b_m\}$*, respectively,* $m, n \geq 0$*, we define the following relationships between A and B:*

- A string equal B*, if A is identical to B. E.g.* *From* string equal *From*.
- A equal B*, if* $A_{cw} = B_{cw}$*. E.g* *Type of Job equals Job Type*.
- A synonym B *if* $n = m$ *and all elements of* A_{cw} *and* B_{cw} *participate in at least one relationship, which is either equality or synonymy but at least one such relationship is synonymy by WordNet. E.g.* *Area of Study* synonym *Field of Work since area is a synonym of field and study is a synonym of work, by WordNet.*
- A hypernym B*, if* $n \leq m$ *and for each* $a_i \in A_{cw}$*,* $1 \leq i \leq n$*, there exists* $b_j \in B_{cw}$ *such that* a_i rel b_j*, where* rel *is either equality, synonymy or hypernymy, by WordNet, and either* $n < m$ *or at least one* rel *is hypernymy. E.g.* *Class* hypernym *Class of Tickets*. *We assume* A *and* B *do not contain* and *(&)*, or *(/)*.
- A hyponym B *if* B hypernym A

3.2.1 Most Descriptive vs. Most General

When assigning a label to a field of a unified query interface it is tempting to strive for the one with the most general meaning among the candidate labels [12]. The rationale lies on the assumption that an element of a global interface should be general enough so that its meaning covers all the concepts of the individual interfaces that map to it. Consider the following example from the job domain. A label for the concept Job Category needs to be chosen from the following set of labels {Category, Job Category, Area of Work, Function}. With this approach either Category or Function

Table 2: A group of clusters example

cluster/schema	c_Senior	c_Adult	c_Child	c_Infant
aa		Adults	Children	
airfareplanet		Adult	Child	Infant
airtravel		Adult	Child	
british	Seniors	Adults	Children	
economytravel		Adults	Children	Infants
vacations	Seniors	Adults	Children	

will be chosen. There are at least two problems with this approach: (1) *a most general concept* could turn out to be *too general* and (2) its meaning could be ambiguous and possibly cover other concepts of the interface. In this example both Category and Function are too generic. We prefer most descriptive names over most general ones. Observe that any of the other more descriptive labels does a better job in conveying the actual meaning to the user.

Next, we introduce our methodology for naming the fields within groups. Then we describe the requirements imposed on the labels of the internal and the leaf nodes that need to be satisfied so as to ensure a consistent labeling of the entire schema tree. A formalism is provided for capturing naming consistency among attributes (fields and internal nodes) of an integrated interface. We conclude with a discussion on the role of instances within the naming problem.

4. CONSISTENT LABELING OF FIELDS

The task addressed in this section is the assignment of meaningful labels to each group of clusters in $\mathcal{G}$. We will treat the set of clusters in $\mathcal{C}_{root}$ as a group. Hence, we will apply the solution we devise for regular groups to this special group, as well. It should be pointed out, that generally among the fields that appear as children of the root in the schema tree there are loose naming consistency constraints. Thus, for this particular group of clusters partially consistent solutions will be accepted. Clusters in $\mathcal{C}_{int}$ will also be treated in this section. However, being isolated, i.e. not part of a group, their labels do not require significant correlation with the labels of the surrounding elements, therefore, finding satisfactory labels is easier. In fact, this is the only place where we found helpful approaches of the past works [12].

4.1 Naming Consistency within Groups

The key idea of our solution is introduced using the example in Table 2. Suppose the airline domain consists of the query interfaces in the first column of the table and the group of clusters [c_Senior, c_Adult, c_Child, c_Infant] is present in the integrated interface. Thus, a consistent naming assignment should be determined for this group. A row in the table corresponds to the labels provided for these concepts by an individual interface. For example, query interface aa has labels for c_Adult and c_Child but not for c_Senior and c_Infant. In line with our assumption that source interfaces are well-designed, *the labels in such a row are consistent to each other*. Note that there is no interface providing labels for all global fields. The existence of such an interface would have given us a good candidate naming solution. However, there is a systematic way to construct a consistent solution even when such a candidate solution is not present. Observe that by combining the labels given by british and economytravel a consistent naming assignment, namely, Seniors, Adults, Children, Infants, can be attained. Each of the first three and the last three labels are consistent as they are members of the same groups in the user interfaces they originate from, i.e. british and economytravel, and they all are consistent because the two sets share labels

Table 3: A set of non-consistent clusters

cluster/schema	c_State	c_City	c_ZipCode	c_Distance
100auto	State			
Ads4autos			Zip Code	Distance
CarMarket	State	City		
cars-1			Your Zip	Within

that are consistent with the labels in both sets. We call this strategy *intersect-and-union*. The underlying idea is that a nonempty intersection of consistent labels from different interfaces is a strong indication that their union should lead to a larger set of consistent labels. Hence, the general strategy would be to figure out a way to combine multiple such rows of labels so that a new row is created that has a label in each column and the labels are consistent. The problem that needs to be addressed is to define precisely when the labels of two query interfaces can be combined.

There are instances of well-defined semantic groups of fields for which the above strategy cannot be directly applied. Table 3 shows an example that falls in this category. The group [c_State, c_City, c_ZipCode, c_Distance] is an actual group of fields of the integrated interface of auto domain. Again, the tabular organization helps us identify the issue. Namely, there is a clear partition of the clusters with respect to the labels supplied by the individual query interfaces: {c_State, c_City} and {c_ZipCode, c_Distance}. In other words, there is no row of labels that links these two sets of clusters. Therefore, no consistency that spans the two sets can be enforced. The best we can do is to compute a consistent solution for each partition and then to concatenate them, producing a best possible consistent naming solution, which will be referred as *partially consistent*. In this case, a solution would be [State, City, Zip Code, Distance].

We next define the precise terms that allow to construct labeling solutions along the logic presented above. First, we organize the clusters of a group in a *(n+1)-ary relation*, where n is the number of clusters in the group and a component denoting the name of the interface. Such a relation is called a *group relation*. A tuple in this relation denotes the labels a particular interface supplies for the clusters of the group. Tables 2 and 3 are examples of such relations.

DEFINITION 2. *Consider a group relation,* gR*, consisting of $n (n > 0)$ clusters, $C_1, ..., C_n$ and $C \subseteq \{C_1, ..., C_n\}$. We define three levels of naming consistency between the tuples of $\pi_C(gR)$:*

1. string level*: two distinct tuples $s, t \in \pi_C(gR)$ belong to string level of consistency if there exists a cluster $A \in C$ so that $t.A = s.A$ (plain string comparison).*
2. equality level*: tuples $s, t \in \pi_C(gR)$ are in the equality level of consistency if there exists $A \in C$, such that $t.A$* equal *$s.A$.*
3. synonymy level*: tuples $s, t \in \pi_C(gR)$ belong to synonymy level of consistency if there exists $A \in C$, so that $t.A$* synonym *$s.A$.*

DEFINITION 3. *Let* gR *be a group relation. For any two consistent tuples r, $s \in gR$, Combine(r, s) = t, where the non-null components of t are all the non-null components of r plus the non-null components of s for which r has null entries. All the other components of t, if any, are null.*

The first level of consistency (i.e. string level) is the strongest and it is adequate for computing consistent solutions for groups similar to the group shown in Table 2. Supposedly, a naming solution computed in its terms should return the "most" consistent one. However, while the strongest in terms of its potential to find a best possible consistent

Table 4: Semantic consistency example.

schema	c_NumConnections	c_TicketClass	c_Airline
aa	NonStop		Choose an Airline
airfare	Number of Connections		Airline Preference
alldest		Class of Ticket	Preferred Airline
cheap	Max. Number of Stops		Airline Preference
msn		Class	Airline

solution, it is also the most restrictive in terms of its applicability. The other levels of consistency are devised to deal with the heterogeneity specific to such a rich environment. They can all be summarized in the *semantic level of consistency*. If set C in the definition is composed of all the clusters of the group shown in the table then the tuples of respectively british and economytravel query interfaces are string level consistent. They both supply the label Adults for the cluster c_Adult and the label Children for the cluster c_Children. Moreover, Combine(british, economytravel) = (Seniors, Adults, Children, Infants), where the first three components of the new tuple are given by the british whereas the last component is given by the tuple economytravel.

We use the example in Table 4 to exemplify the semantic level of consistency. The table shows a group relation in the airline domain (a fragment of it). In this group the labels are much more diverse and more descriptive than in the previous examples and there are no two string level consistent tuples. Nonetheless, the tuples (null, Class of Ticket, Preferred Airline) and (Max. Number of Stops, null, Airline Preference) can be shown to be in the equality level of consistency. There exists cluster c_Airline for which the two supply the same set of content words. That is, the set representation of both Preferred Airline and Airline Preference is {prefer, airline} (Porter's stemming algorithm returns the same stem for Preference and Preferred, i.e. prefer).

The generalization of the Combine operator, Combine*, over a nonempty set of tuples iteratively takes two consistent tuples and combines them, ignoring eventual duplicates. Each resulting tuple may then be combined with another consistent tuple to have more non-null components. The process is repeated until all possible consistent tuples are generated. Hopefully, consistent tuples having no null components, which are exactly what we desire, are generated. The following definition states what the desired tuples are.

DEFINITION 4. *Let* G *be a group, gR its group relation, consisting clusters, $C_1, ..., C_n$, $n > 0$, and $C \subseteq \{C_1, ..., C_n\}$. A tuple t is a* consistent naming solution *for the set of clusters in C, called* tuple-solution*, if (1) $t \in Combine^*(\pi_C(gR))$ and (2) it has no null components. If t with these properties belongs to $\pi_C(gR)$, it is called a* candidate solution.

Frequently, the set C in the definition is the group's set of clusters itself. However, as shown in the example of Table 3, there are instances where a consistent solution for the entire set of clusters cannot be located. In such circumstances consistent solutions for subsets of clusters are computed and the final solution is obtained by concatenating them. Hence, defining the concepts with respect to a subset of clusters in the group instead of the entire set of clusters enables a more flexible way to introduce both the concepts and the algorithms (as many steps will be similar).

The general directions of the algorithm: The algorithm proceeds along the three consistency levels. That is, for all clusters of each group, first a solution is sought based on the string consistency level. If a solution can be obtained using only this level of consistency the algorithm stops and reports the consistent naming solution. Else, the

Figure 4: An example of partitioning the tuples.

algorithm relaxes the constraint and looks for consistent solutions based on the second level of consistency (i.e. equality level). Again, it checks for the existence of a consistent solution. If one cannot be computed then it moves to the third level of consistency. If even this cannot guarantee a consistent naming solution, a solution is constructed based on the consistent solutions for subsets of clusters.

4.1.1 Find Consistent Related Clusters in a Group

In this section we describe our methodology for finding the subsets of clusters of a group from which a consistent naming solution can be constructed. The ideal situation is when it can be shown that the entire set of clusters of a group is name consistent. The input of the algorithm consists of the consistency level (initially *string level*) and the group relation for which a solution has to be computed. To discover the sets of consistent clusters we compute a partition over the set of tuples of the group relation with respect to the consistency definition (see Definition 2) so that each partition is maximal with respect to the given consistency level. In order to put together consistently related tuples we employ a graph oriented closure computation algorithm. That is, we construct an undirected graph, $G(V,E)$, where each vertex $v \in V$ represents a tuple and its set of clusters with non-null values. Hence, a vertex plays two roles: it represents (1) a unique tuple, t, in the group relation and (2) the set of clusters with non-null values of tuple t. Tuples whose entries are all null are discarded. Given two distinct nodes, $v_1, v_2 \in V$, there is an edge between them if and only if the tuples are consistent. In this graph, we find all connected components. In each connected component, the union of the sets of clusters represented by its vertices denotes the set of clusters for which a consistent naming solution can be constructed and the union of the vertices themselves gives the set of consistent tuples. In this way we accomplish two orthogonal goals: we identify the sets of clusters of a group from which a consistent naming solution can be computed and for each such set of clusters we confine the set of tuples among which a consistent solution can be constructed. The goal is to determine if there are partitions whose tuples can be used to construct a consistent solution for the group. A partition with this property is said *to supply a consistent solution* for the group. Whenever they can be identified the algorithm obtains all of them. The algorithm returns the set of all partitions, and the set of all those partitions, each of which covers all clusters of the group. If there are no partitions with this property, the later will be empty and the naming algorithm will pursue with the creation of a partially consistent solution from the set of all partitions.

EXAMPLE 1. *We illustrate this process on the group of Table 2. Figure 4 shows the resulting graph for the tuples in this group when the string level consistency is applied. For instance, there is an edge between the tuples* aa *and* vacation *as they have the same labels in the clusters* c_Adult *and* c_Child*, whereas there is no edge between* aa *and* airtravel *as they do not share any label within this level of consistency. The resulted partitions consists of* {aa, british, economytravel, vacation} *and* {airfareplanet, airtravel}*. A quick inspection of the entries in the table reveals that the former partition supplies a candidate solution for the group, whereas the second does not as it does not have a label for* c_Senior.

PROPOSITION 1. *A* consistent naming solution *for the clusters of a group exists iff there exists a partition of consistent tuples which contains all clusters of the group.*

4.2 Obtaining Labels for Groups

We now describe the extraction of the actual labels for the fields of the group from either a partition supplying a solution or from multiple partitions when a consistent solution does not exist.

4.2.1 Consistent Naming Solution

Upon invocation the algorithm for finding a consistent solution receives a partition that covers with labels all the clusters of a group. The solution can be constructed in many ways once we know the tuples. Here we describe two strategies that strive to capitalize on more properties among the labels. If the time to retrieve a consistent solution is an issue then one can always be found in linear time by applying the *Combine* operator along a spanning tree of the connected component. Otherwise, all consistent solutions are obtained by utilizing the *Combine** operator on all spanning trees of all connected components, each containing all clusters. We have the following elements to decide which one to choose: (1) the frequency of occurrence of each tuple-solution in the relation resulted from the application of *Combine**, and (2) its expressiveness. The expressiveness of a tuple-solution is defined as the number of content words of its constituent labels. The frequency of occurrence is only considered for candidate solutions; we disregard the eventual duplicates generated by the operator. Consider the group relation of Table 4. Suppose the tuples of the relation are partitioned in three disjoint subsets {aa}, {airfare, alldest, cheap} and {msn}. This set of partitions is obtained if only the first two levels of consistency (i.e. string and equality) are considered. Since the second partition comprises all the clusters of the group, a consistent solution can be extracted. Assume two tuple-solutions were generated: (Max. Number of Stops, Class of Ticket, Preferred Airline) and (Number of Connections, Class of Ticket, Airline Preference). Employing the expressiveness criterion the former will be preferred since it has more distinct content words. Whenever there are multiple tuple-solutions we opt for the most expressive one. If multiple candidate solutions have the same expressiveness then we apply the frequency of occurrence criterion.

4.2.2 Partially Consistent Naming

When there is no partition covering all the clusters of a group relation we conclude that a consistent naming solution for the group cannot be constructed. This decision only occurs after all three consistency levels were exhausted. As our goal is to find a naming solution that accommodates most of the clusters in a group a greedy strategy is employed to find a partially consistent solution. The solution is constructed as follows. First, a consistent solution is constructed for each partition of tuples. Such a solution provides consistency only among the clusters covered by the partition. The partially consistent solution is constructed from these solutions. We start with the tuple-solution, t, that has the largest number of non-null values. Then, for the clusters of t having null entries we pick among the remaining tuple-solutions the one that contains the largest number of non-null values, denoted t_1. The labels of t_1 corresponding to the non-null labels of t are added to t. If the new tuple, t, has labels for all the clusters in the group then t is the partially consistent solution. If not, we repeat the steps above until all clusters will have a label.

4.2.3 Naming Conflicts

Before a naming solution for a group is reported we need to make a final check upon its quality. It is possible that the final solution will have the so called *homonym problem* [2, 10]. Namely, two fields of a group may have the same name but different meanings. The classic solution to this problem (e.g. [2, 21]) is in terms of fields' types and values. Since fields with pre-defined domains are rather scarce [23] we present an alternative solution that ignores instances. We start by determining a pair of clusters with similar labels (i.e. homonym) in the tuple-solution. If one is found then we derive another consistent solution which avoids this possible ambiguity. The new solution is computed as follows. We first identify tuples in the group relation with the property that they have non-null entries for both clusters and one of the entries is one of the conflicting labels but the other is not. If a tuple is found, we replace the labels for these conflicting clusters in the naming solution with the labels in the tuple. The assumption is that designers of source interfaces avoid these evident ambiguities. Here is an example. The tuple-solution retrieved is (Position Options Job Type, Type of Job, Company Name), where the second and the third entries are similar. We look for tuples that have either Job Type (or an equivalent label) for the second entry or Type of Job (or an equivalent label) for the third. Each of these tuples should provide good replacements for the conflicting labels (designers avoid ambiguities). Suppose we find a tuple (X, Job Type, Employment Type, X) (X denotes labels which are not useful in this discussion). We replace label Type of Job with the new label Employment Type. The new solution will be (Position Options, Job Type, Employment Type, Company Name). After all these consistency issues are investigated the labeling solution is ready to be reported.

4.3 Naming Groups, Concluding Discussion

We have developed the main ingredients for labeling the fields of a group. The naming algorithm returns a set of pairs $\langle p, CLabels \rangle$, where the first component represents a partition supplying a consistent solution and the second is the naming solution derived from the partition. When there is no consistent solution the set of pairs consists of one pair. Its first component is empty as there is no partition supplying a consistent solution and the second is the partially consistent solution. The selection of a consistent solution for the group at hand from a set of possible solutions is not the responsibility of this algorithm as the solution needs to be correlated with the labels of other attributes within the schema tree. This is accomplished in a different stage of the algorithm which will be shortly introduced.

In our experiments we noticed that for almost all regular groups (i.e. $\mathcal{C}_{groups}$) a partition that covers all the clusters exists. The reason lies in the semantic relationships among the elements in the group. For instance, Number of Connections, Class of Tickets, Preferred Airline all describe service characteristics. However, not the same "homogeneity" can be found among the fields that are children of the root (i.e. $\mathcal{C}_{root}$). For these fields consistency can be rarely enforced along the entire set of fields. Nevertheless, it can be enforced for subsets of fields most of the time. Our experiments have revealed that for domains whose query interfaces are rather flat consistent naming solutions can be only constructed for subsets of fields, that is, partially consistent solutions.

4.4 Assign Names to Isolated Clusters

For this type of clusters (i.e. cluster in $\mathcal{C}_{int}$) we employ a variation of the representative attribute name (RAN) algo-

rithm [12]. To determine the name of an isolated cluster, we use a method based on the generality rule and the most descriptive rule. First, we build hypernymy hierarchies based on the field labels in the cluster. The hypernymy relationships are established through the semantic rules given in Definition 1. The roots of these hierarchies represent the labels with the most general meaning among the labels in the cluster. Next, the most descriptive label among the roots that appears in most interfaces in the cluster is elected as the label of the cluster. Consider a cluster containing the following labels: Class, Class of Ticket, Preferred Cabin and Flight Class. Two hypernymy hierarchies are generated, one has Class as the parent of Class of Ticket and Flight Class, and the other has Preferred Cabin by itself. Then the label for this cluster will be elected between Class and Preferred Cabin. The latter will be chosen as it is the most descriptive.

5. LABELING INTERNAL NODES

To this point we have developed means to compute consistent labels for the fields (i.e. horizontal consistency). What misses is a methodology to select the right labels for both internal and leaf nodes so that a consistent labeling is obtained for the entire integrated schema tree. To achieve this goal we need to define the consistency requirements between the labels of the internal nodes and those of the leaf nodes, and among the internal nodes' labels themselves. We first define how the various candidate labels of the internal nodes are related to the set of possible solutions for their descendant leaves and then we build on this definition to enforce consistency constraints among the labels of the internal nodes.

In assigning labels to the internal nodes of a global schema tree (called herein *global internal nodes*) we mainly exploit two types of knowledge: (i) the relationship between internal nodes of source schema trees with overlapping set of descendant leaves and (ii) the semantic relationships among the labels of these internal nodes. Consider two internal nodes v and w in distinct schema trees. Denote by X and Y the descendant leaves of v and of w, respectively. For ease of presentation, we say X is a subset of Y if for each $x \in X$ there exists $y \in Y$ so that x and y are in the same cluster.

DEFINITION 5. *Let $I_1, I_2 \in \mathcal{QI}$ (set of query interfaces) and two internal nodes, v_1 and v_2, of respectively I_1 and I_2, we say that the label l_{v_2} of the internal node v_2 is* semantically at least as general *as the label l_{v_1} of v_1 if (i) either l_{v_2} is a hypernym of l_{v_1} or (ii) the set of descendant leaves of v_1 is a subset of v_2's set of descendant leaves. If l_{v_1} is also semantically at least as general as l_{v_2} then they are* semantically equivalent.

As a consequence of this definition we derive a *logical inference (LI)* to determine when two labels of two internal nodes in distinct interfaces are semantically equivalent.

LI 1. *Given two distinct query interfaces, $I_1, I_2 \in \mathcal{QI}$, and two internal nodes, v_1 and v_2, of respectively I_1 and I_2 with v_1's set of descendant leaf nodes a subset of v_2's set of descendant leaf nodes. If v_1's label is a hypernym of v_2's label (by Definition 1) then v_1's label is* semantically equivalent *to v_2's label in the given domain of discourse.*

Consider two internal nodes of two schema trees in the Real Estate domain. One of them has an internal node whose label is Location and its set of descendant leaf nodes consists of State and County, whereas the other schema has an internal node labeled Property Location whose set of descendant leaves is composed of State, County and City. As the set of descendant leaves of the former is a subset of the latter it is reasonable to imply that the label of the later, i.e. Property Location, is at least as general as the former's label, Location. On the other hand, if we use the semantic relationships introduced in Definition 1 we can infer that Property Location is a hyponym of Location. Combining the two results we could say that the two labels are equivalent.

Table 5: Example of vertical consistency.

Group_Year		Group_Car_Model			Internal Node	
c_From	**c_To**	**c_Make**	**c_Model**	**c_Keyword**	**Label**	**Label**
Min	Max	Brand	Model			Year Range
Year	To Year	Make	Model		Car Information	
From	To	Make	Model	Keyword	Make/Model	Year Range

While this sort of logic is a reasonable path to consider within the same domain of discourse (here, Real Estate) this might lead to undesirable inferences when applied across domains. For instance, Location with the above characteristics, but in a different domain (e.g. car rental), might denote the place to pick the rented car.

Here are the hypotheses upon we build the consistency requirements. For each group of fields there is a set of partitions, which can be used to construct a (partially) consistent solution. Assume that each group has a consistent solution. Every such partition consists of tuples of labels and each tuple belongs to *at most* one partition. Moreover, an individual schema tree supplies *at most* one tuple per group relation. The fields within $\mathcal{C}_{root}$ do not impact the consistency interaction between the groups and the internal nodes as they have no descendants. In what follows we disregard them. We regard isolated clusters, i.e. $\mathcal{C}_{int}$, as being groups with one cluster only as to provide a more uniform treatment. Therefore, the descendant leaves of any internal node are organized in groups and, thus, we can speak of internal nodes as having descendant groups. That is, a group is seen as a node, called *group node*. Figure 6 (on the right) shows the integrated schema tree of AUTO domain (on the left) with the groups collapsed into group nodes.

Since fields are organized in groups and each group is regarded as a well-defined semantic unit, we capture the consistency relationship between the labels assigned to the leaves and to the internal nodes through the relationship between internal nodes and groups. In this way we accomplish the two goals: horizontal consistency and vertical consistency. The assumption is that if a label of a global internal node is consistent with the solution constructed for a group then it is also consistent with each of the labels assigned to the fields within the group. The definition below states the consistency requirements between labels of the internal nodes and the naming solutions for groups, *with the underlying assumption that the labels of an internal node and those of its descendant nodes within a user interface are consistent* (well-designed assumption).

DEFINITION 6. *Consider a global internal node, v, and one of its descendant groups, G. Let l_v be a candidate label of v that originates in some schema tree I. Denote by t_v the tuple of labels I has in the group relation of G. We say that label l_v is consistent with a consistent naming solution, S, for G if t_v belongs to the partition of tuples of G that was used to compute S. Moreover, if $\{G_1, ...G_n\}$, $n \geq 1$, is the set of all descendant groups of v and $\{S_1, ...S_n\}$ is the set of consistent solutions of these groups then the candidate label l_v of v is consistent with $\{S_1, ...S_n\}$ if l_v is consistent with each S_i, $0 \leq i \leq n$.*

Table 5 shows the organization of two groups, Group_Year and Group_Car_Model. Additionally, a candidate label, Car

Figure 5: Interfaces in auto domain

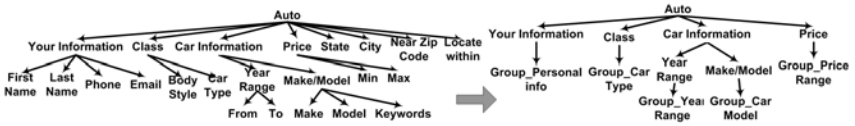

Figure 6: The integrated schema tree of the auto domain.

Information, for the least common ancestor of the fields of these two groups (Figure 6, on the left) is shown in the last column. It will be shown later (next section) that this label is determined to be a candidate label. In the table each tuple of the Group_Year group belongs to a different partition as there are no two labels in the two clusters of the group to ensure any sort of consistency among them. The tuples of the Group_Car_Model group are all in the same partition since they are all consistent. The query interface supplying the label Car Information (Figure 5) provides also tuples in each of the two groups, i.e. (Year, To Year) and (Make, Model, null). By Definition 6 the label Car information is consistent with (Year, To Year) and with any solution resulted from the sole partition of the Group_Car_Model. The Year Range label in the table is a candidate label for the parent node of Group_Year. This label is given by a number of query interfaces, some providing the tuple (From, To) (Figure 5, on the right) and others (Min, Max).

The next objective is to precisely define consistency requirements between the labels assigned to internal nodes. We are only concerned with internal nodes having ancestor-descendant relationship. There are two requirements on their label assignment. First, if w is an ancestor of v in the integrated schema tree, and l_w, l_v their respective labels obtained from the set of query interfaces, $\mathcal{QI}$, then l_w must be semantically at least as general as l_v (Definition 5). Second, we use the consistency of the internal nodes' labels with the labels of the groups to infer consistency rules between labels assigned to the internal nodes. Specifically, we assume that if the labels of both descendant and ancestor internal nodes satisfy Definition 5 and they are consistent with the labels of the common descendant groups then the labels are consistent. The following definition rephrases this observation in a more systematic way.

DEFINITION 7. *Let v and w be two global internal nodes, with v a descendant of w, and l_v and l_w be their respective potential labels. Suppose $\{G_1, ...G_n\}$, $n \geq 1$, is the set of all common descendant groups of v and w. The labels l_v and l_w are called consistent if:*
1. *l_w is semantically at least as general as l_v, and*
2. *there is a set of solutions $\{S_1, ...S_n\}$ for $\{G_1, ...G_n\}$ so that both l_v and l_w are consistent to these solutions.*

To illustrate it we use our running example based on Table 5. We analyze the consistency between Car information and Model/Make, Year Range of the internal nodes in the auto integrated schema tree (Figure 6). Assume the first condition in the definition is satisfied. It will be shown in the next section how this can be established. Year Range is a potential label for the parent node of the Group_Year group, Model/Make for Group_Car_Model and Car information for the lowest common ancestor of Group_Year and Group_Car_Model. First, Car information and Model/Make are consistent with the tuple-solution (Make, Model, Keywords) (see Table 5). However, in Table 5 there does not exist any tuple-solution for Group_Year which is consistent to both Car information and Year Range. As this example shows, the second condition, while ideal and important to meet, in practice weaker constraints can also lead to well-labeled query interfaces. Therefore, whenever these strong constraints are not met we probe for a labeling solution that comply with the first condition only. The internal nodes (see Car Information and Year Range) whose labels satisfy only the first condition of the definition are said to be *weakly consistent*.

5.1 Find Candidate Labels for Internal Nodes

Here we introduce the main set of inference rules to identify suitable labels for internal nodes. For a global internal node we first identify all its descendant leaves, denoted by X. Then we scrutinize all individual schema trees for internal nodes whose sets of descendant leaves are subsets of X. The label of such an internal node is a *potential label* for the global internal node. The potential labels are collected, analyzed and (if possible) a *candidate label* is derived for the global internal node. Once a candidate label is found we add it to the set of candidates labels of the global internal node and annotate it with the set of partitions of tuples (if any) of the groups it is consistent with. We say that a label, l, of the global internal node *semantically covers* X, and thus, it is a candidate label, if it can be shown that each leaf node in X is a descendant of an internal node whose label is l or a label, l_1, satisfying that l is semantically at least as general as l_1 among the individual schema trees.

5.1.1 Overlapping Descendant Leaf Nodes Scenario

We look for labels with the following property:

LI 2. *Let gn be the global internal node for which a label is sought and X its set of descendant leaves. Consider a potential label l that appears in multiple schema trees, $\mathcal{QI}_{sub} \subseteq \mathcal{QI}$. Denote by V the set of internal nodes in $\mathcal{QI}_{sub}$ having label l and whose leaves are among the elements of X. If $X_l = \bigcup_{v \in V} \{v's\ leaf\ descendants\} \subseteq X$ then l semantically covers X_l. If $X_l = X$ then l is a candidate label for gn.*

The example in Figure 8 (on the left) shows an instance of this situation, where the same label, i.e. Location, is used in multiple query interfaces to denote various related pieces of data (i.e. parts of an address). The union of the sets covered by Location is the same as the set of leaves whose lowest common ancestor in the integrated interface needs to be labeled. Thus, Location is a candidate name for this internal node.

5.1.2 Hypernymy Hierarchy Scenario

The next inferences exploits the relationship between the labels of the nodes.

LI 3. *Let v and w be two internal nodes, of respectively query interfaces I and J, whose descendant leaves, X_v and X_w, are contained in X, a set of descendant leaves of a global internal node. If the label l_v of v is a hypernym of w's label then l_v semantically covers the set $X_v \cup X_w$*

An immediate consequence of this line of logic is the following inference.

LI 4. *Suppose there is a set of labels of multiple internal nodes in the set of query interfaces whose leaf nodes union to* X. *A hypernymy hierarchy is constructed from their labels.*

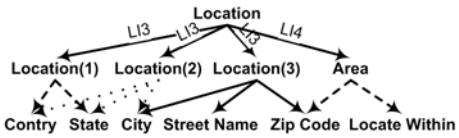

Figure 7: Combine the inference rules.

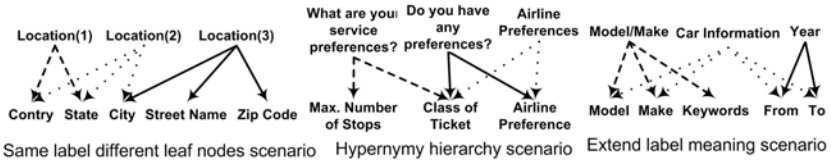

Figure 8: Examples of assigning labels to internal nodes.

From this hierarchy, by applying the above logical inference, the semantic coverage of each label in the hierarchy is established. A label that semantically covers X *is sought. This is a candidate label for the global internal node.*

For illustration we use the example in Figure 8 (in the middle). Using our semantic rules we determine that both Airline Preferences and What are your service preferences? are hyponyms of Do you have any preferences?. After removing all stop words from Do you have any preferences? its set representation will be {prefer}, while of the others will be {airline, prefer} and {service, prefer}, respectively. Thus, the hypernymy hierarchy will have one root, i.e. Do you have any preferences?, and it has the desired qualities to be selected a label of the global internal node since its semantic coverage can be extended to cover all descendant leaves.

5.1.3 Extend Label Meaning Scenario

Consider the configuration in Figure 8 (on the right). Our goal is to extend the meaning of label Car Information to encompass Keywords. First, observe that none of the previous rules help in this scenario. Second, given the (fragment) schema tree in Figure 5 it should be clear that Keywords is rather a dependent concept. That is, it merely characterizes the concepts denoted by the fields Make and/or Model. Hence, if we knew that the semantic meaning of Keywords is bound to Make and/or Model, it would be safe to say that Car Information is general enough to cover the meaning of Keywords, too. The following inference rule states more systematically the discussion above.

LI 5. *Let* X *be the set of leaf nodes whose lowest common ancestor needs to be labeled within the integrated query interface.* X *can be partitioned in two disjoint nonempty sets,* Y *and* Z*, with the property that the fields in* Z *are characterized by a nonempty subset of fields,* W*, of* Y*, if one of the following properties holds:*

1. *the set of instances of the fields in* Z *is a subset of the set of instances of the fields in* W*, or*
2. *there exists a schema tree in* $\mathcal{QI}$ *so that* W *and* Z *are the descendant leaves of one of its internal nodes,* v*, with label* l_v*, having the property that the set of content words of* l_v *is a subset of the set of content words of the labels of the fields in* W*.*

Let v *be a labeled internal node in an individual query interface whose descendant leaves include those in* Y*, but not* Z*'s. If* Z *is characterized by* Y*, then the semantic meaning of* v*'s label can be extended so as to cover* $Z \cup Y = X$*.*

In our example, the label of the internal node whose set of descendant leaves consists of {Make, Model, Keywords} is composed of the content words make and model. This should be a strong indication that the foremost concepts of this group of fields are Make and Model and the rest, i.e. Keywords, just refers to these two. Hence, Car Information semantically covers Keywords as {Make, Model} $\subset$ {Make, Model, From, To} $= Y$ and $Z =$ {Keywords} is characterized by $W =$ {Make, Model}.

Comments: It is not difficult to see how the inference rules can be utilized in combination. As a case in point, consider the example shown in Figure 8 (on the left). Suppose an additional leaf node is added, called Locate within, along with an internal node, called Area, whose descendant leaves are Locate within and Zip Code. In addition, assume none of the Location's include Locate within among their descendant leaves. This new configuration is depicted in Figure 7. As we showed previously, using the inference rule LI2 we can enlarge the semantic coverage of Location to the entire set of fields except for the newly added field, Locate within. However, employing the inference rule LI3, (Location is an hypernym of Area by Definition 1), we can discover that Location covers the entire new set of leaves. Thus, by the two rules we determine Location as a candidate label for the global internal node with these new set of leaves.

If these strategies fail to produce a candidate label then there is a good chance that a label does not exists.

6. PUT ALL THE PIECES TOGETHER

We have developed most of the machineries to define when a global query interface is or is not consistently labeled and to identify where exactly the problems occur. The following definition expresses the precise terms to decide when an entire naming solution for an integrated interface is consistent.

Definition 8. *For a given integrated schema tree there exists a* consistent naming solution *if (1) there is an assignment of consistent solutions for its groups such that each internal node has a label consistent with this assignment and (2) the labels of the internal nodes are consistent (see Definition 7). Moreover, the schema tree is* weakly consistent named *if there are internal nodes whose labels satisfy the first condition of Definition 7, but not the second, and it is* inconsistent named *if either there are groups for which consistent naming solutions cannot be constructed or there are internal nodes without labels whose set of potential labels is nonempty.*

Let e be a global internal node and $path(e)$ the set of nodes on the path from e to the root, excluding e. Denote by L_e the set of candidate labels of e and by $L_{path(e)}$ the union of the candidate labels of the internal nodes in $path(e)$.

Proposition 2. *There exists a weakly consistent labeling for* $\mathcal{I}$ *if (1) there is an assignment of consistent solutions for all groups and (2) for each internal node,* e*, distinct from the root* $L_e - L_{path(e)} \neq \emptyset$*.*

Clearly, a weakly consistent integrated schema tree, containing an internal node with the property that none of its candidate labels are consistent with at least a solution of each of its descendant group nodes, does not have a consistent labeling solution.

The naming algorithm is a three-phase traversal algorithm. In the first phase, in a bottom-up traversal, it determines the set of candidate labels for leaves and internal nodes. Second traversal determines the level of consistency which may be possible for the schema tree. In the third phase, each node is assigned a label from its set of candidate labels so that the label complies with consistency level established in the previous phase. Each of these processes was described in detail in the previous sections.

Table 6: Characteristics of interfaces per domain.

Domain	Domain Characteristics (Avg)				Integrated Query Interface						Statistics			
	Leaves	Int Nodes	Depth	LQ	Leaves	Groups	Iso. Leaves	Root Leaves	Int Nodes	Depth	FldAcc	IntAcc	HA	HA′
Airline (20)	10.7	5.1	3.6	53%	24	8	0	1	13	5	100%	84.6%	96.6%	98.3%
Auto (20)	5.1	1.7	2.4	79.7%	18	5	0	4	7	3	100%	100%	100%	100%
Book (20)	5.4	1.3	2.3	83.3%	19	5	1	8	6	3	100%	100%	98.9%	100%
Job (20)	4.6	1.1	2.1	80%	19	1	0	15	2	2	100%	100%	100%	100%
Real Estate (20)	6.7	2.4	2.7	79.1%	28	8	1	7	8	4	96.4%	100%	97.8%	97.8%
Car Rental (20)	10.4	2.4	2.5	52.5%	34	9	3	3	15	5	100%	93.4%	97.9%	98.2%
Hotels (30)	7.6	2.4	2.3	70.1%	26	8	3	2	15	5	100%	93.4%	95.3%	96.1%

6.1 Where Instances Could Help

As opposed to the classical entities treated in the integration problem (e.g. relational) the fields of query interfaces may not have pre-defined domains. Nonetheless, we underline some circumstances where instances may come in handy. Specifically, we describe two very sensitive problems for the naming process: picking a label from a set of labels while avoiding too generic or too specific labels and discarding those that are merely values of other fields.

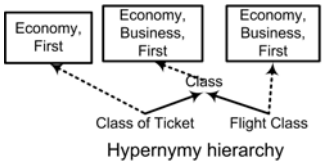

Figure 9: Most general vs most descriptive.

6.1.1 *Reconcile* Most General *vs* More Descriptive

There are several places in our algorithm where a label needs to be drawn out from a set of candidate labels. Blindly relying on the most descriptive strategy is not always the best strategy. However, it is possible to reconcile the apparent divergent goals of the most general and the more descriptive criteria. The idea is to bound the meaning of a most general label to a more descriptive one employing their instances collected from all the query interfaces they are used to denote equivalent fields.

LI 6. *Consider a cluster, C, for which a label has to be chosen from a nonempty set of labels, L. For a label $l \in L$ denote by domain(l) the union of instances of all the fields in cluster C having label l. Let l_1 and l_2 be two labels in L so that l_1 is an hypernym of l_2 (Definition 1). If domain(l_1) $\subseteq$ domain(l_2) then l_1 and l_2 are semantically equivalent in the given domain.*

Suppose we need to elect a label among Class, Class of Tickets, Flight Class. Using the semantic relationships of their tokens we obtain the hypernymy hierarchy shown in Figure 9. Assume each of these labels come from fields equipped with instances. For each of the fields with the same label we compute the union of their domains. Then we probe among the hyponyms of the root to determine the presence of a label whose associated domain includes the domain of the root label. For example, Flight Class and Class have the same domain. Thus, it would be safe to infer that although Class is more generic its use in this domain (i.e. airline) is bounded to the semantic meaning of the more descriptive label Flight Class. Hence, Flight Class will be chosen as the label over Class.

6.1.2 *Discard Labels as Values*

This problem is known in the schema matching literature as either *schema element name as value* [25] or *schema mismatch* [7] and it characterizes those cases when matches relate the data of a schema with the labels in another schema. For example, in the Book domain labels like hardcover or paperback are data instances of a field with label Format or Binding. This mismatching type occurs very frequently in practice, e.g., product description, course listing. Obviously, such too specific labels need to be identified and discarded in the process of label selection. The following logic inference is meant to identify such occurrences:

LI 7. *Consider two fields* f *and* e *in the same cluster. If* e*'s label occurs among the instances of* f*, then* f*'s label is semantically at least as general as* e*'s label.*

7. EMPIRICAL EVALUATION

Experiment setup: We evaluated our algorithm on 150 sources over 7 real-world domains on the Web, each consisting of 20 query interfaces, excepting Hotels that has 30. Table 6 (columns 2-5) shows a summary of the characteristics of the source data sets per domain, i.e. the average number of fields and of internal nodes on the individual interfaces (columns 2,3) and their average depth of individual interfaces (column 4). In the 6th column we provide a metric describing each domain's labeling property. *LQ* stands for *labeling quality* and it is the average of per query interface percentage of labeled nodes (both internal and leaves). That is, on average an individual query interface in Car Rental domain has about 52% of its nodes labeled, whereas in Book domain more than 80%. The characteristics of the resulted global interfaces for the 7 domains are illustrated in columns 6-11. Herein we will focus on the naming only. Due to space limitation we cannot exemplify all seven integrated interfaces, but the reader is encouraged to study them on our project web page [1].

To show the effectiveness of our solution we need, first, to evaluate the resulted integrated interface against the desired properties (i.e. (weakly) consistent) and, second, to evaluate the ease of being used by an ordinary user.

Consistency Quality: To appraise the accuracy of the automatic consistency enforcement we define two metrics. The first, called *fields consistency accuracy*, measures the level of accuracy the algorithm succeeds to enforce consistency among the fields. Which means that for a group either all leaves are labeled consistently or if there are leaves without a label then they will have instances associated with them. It is defined as the ratio of fields (within the groups, isolated and children of the root) consistently labeled over the total number of fields. The outcome for the seven domains is given in column "FldAcc". We get almost perfect accuracy. In the Real Estate (Figure 11) Lease Rate has two children, one having no label. However, the semantics of the No Label node can be easily inferred by a user given the label of its sibling, To. Moreover, the field does not have a label in any of the source interfaces where it is present. Hence, there is no way the algorithm can assign a label to it. The second metric, called *internal nodes accuracy* is the ratio of the number of internal nodes that have labels (i.e. they are at least weakly consistent) over the total number of internal

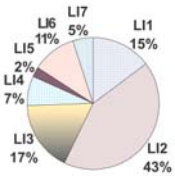

Figure 10: LIs Involvement

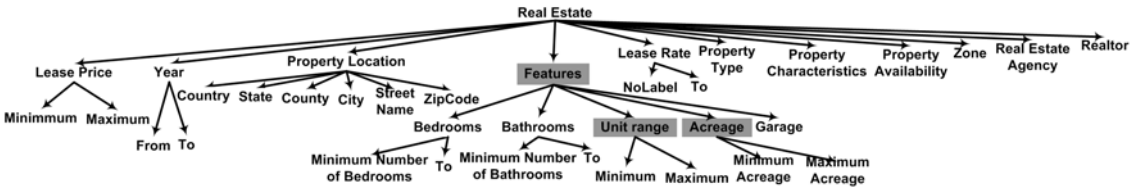

Figure 11: The integrated schema tree of the Real Estate domain.

nodes (column "IntAcc"). Observe that we get very good accuracy. For airline the accuracy is influenced by a group of attributes that occurs once among the individual interfaces and it does not have a label. Such a group can adversely influence the results as it can propagate the inconsistency among the internal nodes along the path to the root. This group was troublesome for many of the people involved in the survey (see below). The airline global interface is inconsistent. The Car Rental integrated interface is inconsistent as well. It has a node whose set of candidate labels is promoted to its ancestors. Thus, the algorithm cannot assign a label to this node. All the other schema trees are either weakly consistent or consistent, e.g., Auto, Job and Book. In the Real Estate, Features is only weakly consistent with Unit Range and Acreage.

Human Acceptance: It is worth contrasting the result of the automatic labeling algorithm to human acceptance. We asked a number of people (a total of 11 people with different backgrounds reported: e.g., students, actuaries, engineers) to answer three simple questions. The first question asked was: "Do you have any difficulty in filling in an entry for each field?". The column *HA* (i.e. human acceptance) shows the feedback received. *HA* is defined as the average of per person percentage of non-ambiguous attributes within an integrated interface. For instance, nobody identified any problem in the Auto and Job unified interfaces, whereas all the others have some problems. For example, for the airline integrated interface 4 persons found the group of fields [Return From, Return To] confusing. Hotels integrated interface has several fields that people found too specific to be included in a generic interface (e.g. Wyndham ByRequest No, see [1]). The next question was "If you do, identify the fields you have difficulty filling in?". When such fields are identified, we submitted a source query interface which contains these fields and asked "Are the fields understandable on the source interface?". The latter helped us distinguish between those instances where the errors are due to our algorithm from those that are merely due to those fields which are inherently difficult for user to understand. We discount those fields which are difficult to understand in both integrated interface and on some source interfaces. Then we recomputed the metric above, called HA' (see last column in the table). Note that in the Book, Airline, Car Rental and Hotels people have accounted the sources for some of the errors. As a case in point, all the errors in the Book integrated interface are due to the input interfaces and in the Airline half of the errors originate from source interfaces. For all the others, they believe that the fields are easier to grasp on the interfaces they originate from. The answer to the second question has revealed a point already stated in the literature [12, 8]. That is, fields with low frequency can be removed from the global interface to improve its quality. In our survey, without exception all the fields that people found hard to understand have very low frequency. More precisely, they all have a frequency of 1. Which means they are too specific to be included in the global interface. For example, all ambiguous fields in the Hotels interface refer to discount programs specific to certain hotel chains (e.g. WYNDHAM). Car Rental has the same problem.

An important aspect of the survey was the comments collected. People pointed out that certain elements are not organized in the order expected. For example, within some of the unified interfaces the date does not have the American layout (mm/dd/yyyy). In our view this is mainly a regional issue, which might be solved by considering only search engines from the same geographic area. Our data set has interfaces from both Europe and US. Moreover, some people noticed a sort of redundant concepts on the global interface. One comment was: "Do you need to request the number of nights if you already asked for the check in and check out dates?". Which is a very valid point and we plan to address it in the future.

Evaluation of Inference Rules: The pie chart (Figure 10) gives a graphical image of the overall involvement of the inference rules in the seven domains. Each slice represents a logical inference involvement in all seven domains, i.e. the ratio of the total number of times the inference was used to produce candidate labels over the total number all inferences were used to produce candidate labels. All inference rules were used in the seven domains, with the inference rules LI2 and LI3 being employed more frequently.

The experimental results show that query interfaces satisfying the naming consistency definitions, come very close to a human accepted query interface. Additional information on our work, including further experimental evaluation, can be found on our project's Web site [1].

8. RELATED WORK

There are many individual aspects our naming algorithm depends on that are too voluminous to cite here, e.g., effectiveness of works on query interface matching [7, 10, 24], fields grouping [8] and merging [4, 8, 15, 17, 20]. Thus, we will focus on works where naming is significantly treated.

Although the problem of meaningfully labeling of elements within a model (e.g. query interfaces, forms, ontologies) is not a particularity of query interfaces only, to the best of our knowledge, there is no work to tackle the problem to the extent we do in this paper. In the integration literature the existing efforts either recognize the problem and ignore it (e.g. [16, 20]), delegating it to the designer, or acknowledge the problem and commit to some resolution (e.g. [3, 12, 17, 21]), which is covered next. With the exception of [3, 12, 17], most of the works treat a rather specialized aspect of the problem, namely, *naming conflict*. Synonyms and homonyms are the two sources of naming conflicts, and renaming is the most frequently chosen solution in traditional methodologies [2].

Ontology merging is an area sensitive to naming and the state of the art algorithms isolate naming conflicts during the merge process and either pursue some automated resolution [17], by renaming one of the offending concepts, or it provides a list of candidate names for the user to choose

from [14]. The common denominator of all these works is the presence of a domain expert who is deeply involved during the merge process being required to decide on most of the naming conflicts (and other conflicts as well), whereas our approach strives to eliminate any human involvement. [12] builds hypernymy hierarchy of labels in the same cluster and it chooses the representative label for the cluster (i.e. global field) among the roots using the majority rule. We adopted this technique for finding a label for an isolated cluster of an integrated schema tree with a modification by replacing the majority rule by the most descriptive rule.

The salient aspects distinguishing all existing works on labeling user interfaces from ours are: interfaces are modeled as flat schemas, thus, no labeling is required for nodes denoting groups of related fields, there is no notion of fields grouping, no consistency is sought among the labels, except for avoiding too specific labels [12], instances are not considered during the labeling process, and except for knowledge acquired from external thesauruses/dictonaries, inference rules are not involved in the naming process

9. CONCLUSIONS & FUTURE WORK

We consider our naming framework to be also pervasive to other integration areas (e.g. concept hierarchies, HTML tables, ontologies) where more descriptive labels are commonly used and often preferred over abbreviations.

The central goal of this work is to show that well-designed integrated query interfaces for given domains can be achieved automatically. We described a naming algorithm for assigning meaningful labels to the elements of an integrated query interface. A novel abstraction to capture the consistency among the labels assigned to various attributes within a global interface was introduced. Moreover, a set of inference rules were discovered and experimentally proven to accurately suggest candidate labels for the elements of a global interface. We believe these inference rules to be of particular interest in the Semantic Web area, as well. Finally, the naming algorithm has well-defined properties to characterize the consistency levels for integrated interfaces. To the best of our knowledge, this is the first piece of work making such guarantees for a merged interface. The experiments on the 7 domains demonstrate that the formalism we provide is useful in pactice. There is substantial work left to be done. We aim to experimentally show that our framework is readily applicable to other areas of interest sensitive to labeling process, e.g., integrated concept hierarchies or HTML forms.

Acknowledgments: This work is supported in part by the following grants from the National Science Foundation: IIS-0414981 and IIS-0414939.

10. REFERENCES

[1] http://www.cs.uic.edu/∼edragut/QIProject.html.
[2] C. Batini, M. Lenzerini, and S. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 1986.
[3] M. Bright, A. Hurson, and S. Pakzad. Automated Resolution of Semantic Heterogeneity in Multidatabases. *ACM Transactions on Database Systems*, 19(2):212–253, 1994.
[4] P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In *EDBT*, 1992.
[5] C. Chang and H. Garcia-Molina. Mind Your Vocabulary: Query Mapping Across Heterogeneous Information Sources. *SIGMOD Record*, June 1999.
[6] K. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. In *SIGMOD Record*, 2004.
[7] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Mappings between Database Schemas. In *SIGMOD*, 2004.
[8] E. Dragut, W. Wu, P. Sistla, C. Yu, and W. Meng. Merging source query interfaces on web databases. In *ICDE*, 2006.
[9] C. Fellbaum. Wordnet: An on-line lexical database and some of its applications. 1998.
[10] B. He, K. Chang, and J. Han. Discovering complex matchings across web query interfaces: A correlation mining approach. In *SIGKDD*, 2004.
[11] H. He, W. Meng, C. Yu, and Z. Wu. Constructing interface schemas for search interfaces of web databases. In *WISE'05*.
[12] H. He, W. Meng, C. Yu, and Z. Wu. WISE-integrator: An automatic integrator of Web search interfaces for e-commerce. In *VLDB*, 2003.
[13] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, pages 251–262, 1996.
[14] D. L. McGuinness, R. Fikes, J. Rice, and S.Wilder. An environment for merging and testing large ontologies. In *Proc. KR*, page 483493, 2000.
[15] S. Melnik, P. Bernstein, A. Halevy, and E. Rahm. Supporting executable mappings in model management. In *SIGMOD*, pages 167–178, 2005.
[16] S. Melnik, E. Rahm, and P. Bernstein. Rondo: A Programming Platform for Generic Model Management. In *SIGMOD*, pages 193–204, 2003.
[17] N. Noy and M. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *AAAI/IAAI*, pages 450–455, 2000.
[18] Q. Peng, W. Meng, H. He, and C. Yu. Cluster: Clustering e-commerce search engines automatically. In *WIDM*, 2004.
[19] M. Porter. The porter stemming algorithm. Accessible at http://www.tartarus.org/ martin/PorterStemmer.
[20] R. Pottinger and P. Bernstein. Merging Models Based on Given Correspondences. In *VLDB*, 2003.
[21] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogenous and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.
[22] S. Spaccapietra and C. Parent. View integration: A step forward in solving structural conflicts. *TKDE*, 1994.
[23] W. Wu, A. Doan, and C. Yu. Webiq: Learning from the web to match query interfaces on the deep web. In *ICDE*, 2006.
[24] 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*, 2004.
[25] L. Xu and D. Embley. Discovering Direct and Indirect Matches for Schema Elements. In *DASFAA*, 2003.
[26] Z. Zhang, B. He, and K. Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *SIGMOD*, pages 107–118, 2004.
[27] Z. Zhang, B. He, and K. Chang. Light-weight domain-based form assistant: querying web databases on the fly. In *VLDB*, pages 97–108, 2005.