Critical Points for Interactive Schema Matching

Guilian Wang[1], Joseph Goguen[1], Young-Kwang Nam[2], and Kai Lin[3]

[1] Department of Computer Science and Engineering, U.C. San Diego, {guilian, goguen}@cs.ucsd.edu
[2] Department of Computer Science, Yonsei University, Korea, yknam@dragon.yonsei.ac.kr
[3] San Diego Supercomputer Center, U.C. San Diego, klin@sdsc.edu

Abstract. Experience suggests that fully automated schema matching is infeasible, especially for n-to-m matches involving semantic functions. It is therefore advisable for a matching algorithm not only to do as much as possible automatically, but also to accurately identify the critical points where user input is maximally useful. Our matching algorithm combines several existing approaches, with a new emphasis on using the context provided by the way elements are embedded in paths. A prototype tested on biological data (gene sequence, DNA, RNA, etc.) and on bibliographic data, shows significant performance improvements from user feedback and context checking. In non-interactive mode on the purchase order schemas, it compares favorably with COMA, the most mature schema matching system in literature, and also correctly identifies critical points for user input.

1 Introduction

Many scientific and commercial applications use multiple distributed information sources with metadata, of which schemas are the most important. Discovering semantic correspondences between different schemas, called **schema Matching**, is a critical step in many such applications [5, 2, 4, 13], but can be very difficult, since even schemas for the same entities from different sources can have very different structural and naming conventions, and may also use different data models. Similar, or even the same, labels may be used for schema elements having totally different meanings, or having subtly different semantics, due to differences in the unit, precision, resolution, aggregation, measurement protocol, etc.; this is extremely common in domains like environmental science, ecology, biology, and commerce.

There are problems beyond differences in naming and structure. For example, in the bibliographic domain, price units may differ, items may be missing (e.g., price, publication date, or publisher name when the source consists of that publisher's books), author names may be full or separated into first and last, and some may have a different scope, e.g., including articles as well as books. Scientific domains often have indirect matches among different schemas, e.g., in ecology, the density of a species in a community is species count divided by area, so one schema could have a `density` column, another may have `count` and `area` columns, although both schemas are for biodiversity. Hence a single element in one schema may correspond to multiple elements in another, and it is also possible for multiple elements in one schema to correspond to multiple elements in another; such matches are called n:m.

Moreover, n:m matches can have different meanings in different contexts. For instance, the `PurchaseOrder1/OrderHeader/Contact` from one company may relate to `PurchaseOrder2/BillTo/Contact` and to `PurchaseOrder2/ShipTo/Contact` from another company, and contact information from the first may be the simple union of two contacts from the second. These are 1:2 matches that can be simplified to two 1:1 matches, `([Contact, BillTo/Contact], sim)` and `([Contact, ShipTo/Contact], sim)`, and hence can be treated like other 1:1 matches. On the other hand, the correspondences of full author name to concatenation of first and last names, and of species density to count divided by area involve conversion functions, and should be represented as 1:2 correspondences, `([name, first concat last], sim)`, `([density, count div area], sim)` respectively. Unfortunately, most existing schema matching tools treat all these cases the same way, considering `[name, firstname]` and `[name, lastname]` as correct matches, often called "global n:m matches" when $n = 1$ and $m = 2$.

When one schema element is related to multiple elements in another, whether a real multiple match exists, and whether it is appropriate, can be very difficult to determine, due to various forms of context dependency. In the purchase order schema matching problem, `Contact` in the purchase order header of one schema may correspond to both billing `Contact` and shipping `Contact` in another, although the `Contact` for the purchase order is different from the billing or shipping `Contact`. On the other hand, matching `Contact` in the purchase order header to the supplier `Contact` is not desirable. Also, although `Bib/Book/Author` in one schema is very similar to both `arts/article/author` and `arts/book/author` in another, only the `Bib/Book/Author` to `arts/book/author` correspondence makes sense. Such examples strongly suggest that totally automatic matching is infeasible, since some correspondences are correct only under subtle conditions that are impossible to infer from just the schema and/or straightforward auxiliary information.

Existing automatic tools mainly help discover simple 1:1 matches, without considering data semantics, or how the generated mapping will be used, and thus often require significant manual effort to correct incorrect matches and add missing matches. In practice, schema matching is still done manually by domain experts, usually with a graphical tool [9, 11], and is very time consuming when there are many data sources or when schemas are large and/or complex.

The goal of schema matching systems is to save manual effort, but the total manual effort includes work performed to prepare schema and auxiliary information, as well as to guide the match process, and edit the results to get a correct mapping. However, existing metrics do not measure total manual effort. The most common metrics only compare accuracy and completeness of match results. However, a system that can identify the critical points where input can most help the matching process could save more manual effort than another system having the same quality metrics; a system that asks users for input where matches cannot be automatically determined can save more manual work than a system that generates wrong matches at those places, even if it provides some guidance. Moreover, because schemas are often added to or deleted from distributed systems, and because schemas themselves often evolve, a matching system that incrementally updates existing mappings could save much more effort than a system that always treats matching as a new task.

Our approach is to discover simple correspondences using a combination of existing matching techniques, prompting the user to provide input at critical points when no adequate match can be determined, e.g., when there is more than one almost equally good match, or when the contexts embedded in possibly corresponding paths are not consistent with each other, or where n:m correspondences and/or functions may be needed. Context information provided by the paths of schema elements is explicitly used to improve simple match accuracy and detect critical points for user input. A research prototype has been tested on bibliographic, biological, and purchase order data, with encouraging results, though it should be noted that not all the features we discuss in this paper have as yet been fully implemented.

2 Related work

An extensive review of existing techniques and tools for automatic schema matching is given in [13]. LSD and GLUE use machine learning to evaluate data instances and train matchers, and then predict element similarities by combining their results [1, 3]. Cupid combines name and structural matching, and predicts element similarity based on the similarity of the name and data type of their components (sub-elements and attributes in XML DTD or Schema), which can easily lead to incorrectly identifying atomic elements (leaves in schema tree) [4]. SF [10] and Rondo [7] use a versatile graph matching algorithm that is also used in our prototype for structural matching, and they introduce more realistic metrics for match accuracy, which are also used by COMA and in our system. COMA supports flexibly combining different schema matching techniques, reuse of previous matching results, and user input during matching; it has been carefully tested for real purchase orders schemas, showing that reuse of existing mappings can help a lot [2]. More approaches [17] emerged after [13]. All these tools only help find 1:1 matches, and have great difficulty with matches that involve conditions or conversion functions. Further, n:m matches for n or m > 1, cannot be discovered automatically at all. [15] and [14] use ontologies to discover some indirect matches that involve composition or decomposition of multiple elements based on their ontological descriptions, e.g., sample values and keywords, but require the user to provide the ontologies. [14] also argued that user interactions are desirable for deciding many issues in its source schema to target ontological view mapping framework.

3 Our approach

The most novel aspects of our approach are its support for n:m matches that involve semantic functions, and its focus on providing maximum support for users, rather than total automation. It enhances, combines, and reuses algorithms for linguistic, structural, and graph matching from prior work to discover as many good matches automatically as possible, and more importantly, to detect where user input may be most valuable and to identify what specific input to request from users. Its structural matching considers closely connected nodes, as in ARTEMIS, COMA, Cupid, DIKE, and Rondo. It has a series of matching stages using different methods, as in COMA and Cupid; results from different methods are combined by weighted summation. Linguistic similarity of elements is based not just on their tag strings, but also on their path strings, as in COMA, so that hierarchical schema structure is treated as linguistic, and paths provide context dependent matches for shared elements, as in Cupid. The following are some more technical differences from prior work:

a. Schemas are represented in both tree and graph formats: the tree format is convenient for context and sub-tree matching, while the graph format accommodates additional schema information, including element type (attribute, atomic, or context), data type, and constraint.

b. Specific user input is interactively requested at critical points, not just pre- and/or post-match, and user supplied matches are not modified later; this makes post-match editing much easier, since bad guesses made without user guidance propagate less.

c. Graph matching gives new matches based on user supplied and/or high similarity value matches from previous stages, without changing these "good matches" in its fix-point computation and later stages.

d. Since tag meanings can vary with context, and context is given by preceding (i.e., higher) elements, we seek to identify **core** contexts and attributes, which are the most important contextualizing elements for tags within subtrees, by using heuristics and user input; then threshold and context are checked for them. For example, in matching two book DTDs, the contexts `/arts/book` and `/arts/article` are found to be core contexts, with title, author, publisher as their main attributes. Previous steps found that the best match with `/arts/article` is `/bookstore/book`, but its similarity value is lower than the core threshold, so that matches in its subtree are not reliable, and user input is needed. Matching `/arts/book/publisher/name` with `/bookstore/book/author/name` fails because they are in different contexts, even though they have the same tag, and there is no better match for `/arts/book/publisher/name`.

e. Matching two schema trees is divided into matching their sub-trees rooted at core context nodes. This improves manageability, and provides a general framework for discovering and maintaining n:m matches.

Figure 1 shows our structure for matching a source schema S and a target schema T. One or more iterations may be executed, depending on whether the mode is automatic or interactive. In interactive mode, matching strategy selection and match candidate determination are done interactively, requesting specific user input at critical points, while in automatic mode, everything is done with default strategies. Each iteration has 5 steps: optional user interaction, linguistic and data type matching, structural matching, optional context check, and combination of match results; the last four steps also have optional user interactions. Match processing and the input preprocessing are described in the following subsections.

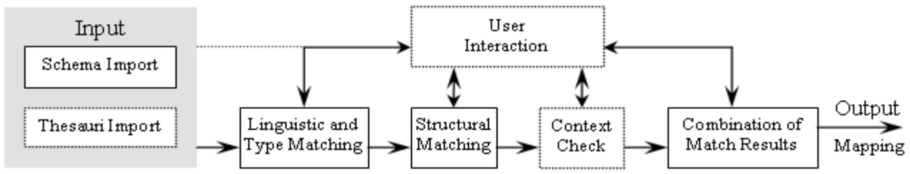

Fig. 1. Diagram of match processing

3.1 Input Preprocessing

The input consists of target and source schemas, and optional domain thesauri. The **Schema Import** module produces a tree and a graph representation of each schema. In the tree representation, each element and attribute of a schema is a node, and sub-elements are child nodes. Each node has its type (attribute, leaf, or parent, which is possibly a context), data type, height, constraints, two sets of matches (n_j, *sim*) for node n_j in the other tree, for pre-context-check and post-context-check, respectively, etc.

The graph representation uses the RDF (directed labeled graph) model. Each element and attribute that is a node n_i becomes an RDF resource, res_{ni}. Each node type nt_i and data type dt_i become RDF resources res_{nti} and res_{dti} respectively. Corresponding edges are created for each node and added to the graph, two for its parent if it is not the root node, (res_{ni}, $parent\_height_{ni}$, $parent_{ni}$) and ($parent_{ni}$, $child\_height_{ni}$, res_{ni}), one for its node type, (res_{ni}, *nodeType*, res_{ntni}), one for its data type, (res_{ni}, *dataType*, res_{dtni}), etc.

3.2 User interaction

Users can optionally provide some good matches and some mismatches, before the initial iteration. If the target element and the source element have significantly different names in a specified good match, then users are asked for input to update the domain thesauri.

3.3 Linguistic and data type matching

In the current prototype, linguistic matching includes: (1) terminological matching, in which element name and path are normalized through domain specific thesauri containing a synonym table, an abbreviation table and a hypernym table; and (2) syntactic matching, which is purely based on string similarity, computed according to the number of substrings they have in common; for this, the implementation in [7] is borrowed. For any pair of nodes (n_i, n_j) in the two trees, both normalized name similarity value $ns_{i,j}$ and normalized path similarity value $ps_{i,j}$ are computed by

$$ns_{i,j} = Coeff \times StringMatch(Normalize(name_{ni}, name_{nj}))$$
$$ps_{i,j} = Coeff \times StringMatch(Normalize(path_{ni}, path_{nj}))$$

where $Coeff = 0.8$ if $name_{ni}$ or $name_{nj}$ happens to be hypernymic to the other, otherwise $Coeff = 1.0$.

$dts_{i,j} = Compatibility(dt_{ni},\ dt_{nj})$

The linguistic and data type similarity value, $ls_{i,j}$, is the weighted sum of name, path and data type similarities, $ls_{i,j} = ns_{i,j} \times w_{n\_l} + ps_{i,j} \times w_{p\_l} + dts_{i,j} \times w_{dt\_l}$, where w_{n_l}, w_{p_l} and w_{dt_l} are the weight values for name, path and data type similarity respectively, at the linguistic and data type matching steps, and $w_{n\_l} + w_{p\_l} + w_{dt\_l} = 1.0$.

If there are multiple source nodes with big subtrees having almost the same linguistic and data type similarity to a target node that also has a big subtree TST, and if these subtrees are also similar, based on their share of linguistically similar nodes, then this might indicate that the target subtree TST corresponds to multiple source subtrees. Questions about whether there exist multiple match subtrees for this target subtree, which of the similar ones are good ones, and whether there are more, should be posed to users. After user input on matching subtrees, SST_i (0 $\leq$ i $\leq$ k), for target subtree TST, the original problem of matching source schema tree S and target schema tree T is divided into k+1 smaller matching problems: matching $T - TST$ with $S - \sum_i SST_i$, matching TST with each SST_i for $0 \leq i \leq k$.

3.4 Structural matching

The Similarity Flooding algorithm for graph matching in [10] was extended to structural matching for our prototype. Taking as input graphs G_t, G_s, and a set of initial similarity values, *InitMap*, between the nodes of the graphs, it iteratively propagates initial similarity values of nodes to surrounding nodes, using the intuition that neighbors of similar nodes are similar, and finally returns the structural similarity $gs_{i,j}$ of any node n_i in G_t and any node n_j in G_s. The original algorithm was modified to freeze input mappings from the user and/or of high similarity from previous steps for some nodes. The initial mapping *InitMap* includes all user input matches, and the best candidate with $max\_ls_i \leq th_{l\_g}$ for each target node from the linguistic and data type matching step, where th_{l_g} is a similarity threshold. Thus

$gs_{i,j} = GraphMatch(G_t, G_s,\ InitMap)$.

The quality of *InitMap* has a very important influence on the output of the structural matcher; users can also improve quality by confirming good matches, denying bad ones, and adding new ones. How we modified the algorithm is shown in the extended version [12] of this paper. After this graph matching step, the similarity between two tree nodes is the weighted sum of similarity values of name, path, data type and structure:

$s_{i,j} = ns_{i,j} \times w_{n\_c} + ps_{i,j} \times w_{p\_c} + dts_{i,j} \times w_{dt\_c} + gs_{i,j} \times w_{g\_c}$ (*)

where w_{n_c}, w_{p_c}, w_{dt_c}, and w_{g_c} are the weight values for combining the name, path, data type and structural similarities to identify the best match candidates for each node right before the following context check step, and $w_{n\_c} + w_{p\_c} + w_{dt\_c} + w_{g\_c} = 1.0$.

3.5 Context check

Matches are refined by checking and re-matching subtrees rooted at core elements, as discussed in point (d) above. After core contexts are identified, they are sorted in ascending order of height from the root, for top-down matching refinement. For each node n_i, check if its best matching node has maximum similarity bigger than a relatively high similarity threshold. If not, for this important target node, there may be no good match from the combined matching steps, so that its subtree matches also might not be good, and user interaction is requested; or in automatic mode, some matches with very low similarity within the subtree are thrown away and child nodes that are core nodes are checked further. If a good match is found, refinement of subtree matches to the best matching node n_j's subtree starts by computing initial mappings, $InitMap_{subtree\_ni,subtree\_nj}$ which are the best so far within these two subtrees (but not the best at whole trees instead) and with similarity bigger than a threshold. Then subtree graphs $G_{subtree\_ni}$ and $G_{subtree\_nj}$ are generated as input with $InitMap_{subtree\_ni,subtree\_nj}$ together for *GraphMatch* to compute again the structural similarities, $gs_{h,k}$ of nodes n_h in n_i's subtree and nodes n_k in n_j's subtree, by

$$gs_{h,k} = GraphMatch(G_{subtree\_ni}, G_{subtree\_nj},\ InitMap_{subtree\_ni,subtree\_nj}).$$

More checking is needed to see whether any almost equally good multiple matches with very high similarity values from the linguistic and data type matching step for these core nodes are still competing candidates after context check. If they are very different, this might indicate that some multiple matches would be missed. User input on specifying whether or not they are multiple matches can avoid missing multiple good matches. Then the core node subtree is matched to the multiple matching subtrees as discussed above, and a new similarity value for n_h and n_k is computed using formula (*), but the weight values for similarities from name, path, data type and graph matchers may change. Then for each target node n_i, another set of matches (n_j, sim), where n_j is a node in the source tree, is computed as the matching result of the context check matcher.

Core nodes might not group at only one or two levels in a schema tree, and they might not be identified precisely by heuristics. User input on specifying some or all core nodes can help the system concentrate on checking important subtree matches. When it is hard for the system to decide good matches for a core node, user feedback can significantly improve whole subtree matches, and significantly reduce total user effort.

3.6 Combination of match results

After performing the above matching steps, each target node gets a set of candidates from combining the results from the four individual name, path, data type and structural matchers, and another set of candidates from the context check matcher. Selecting top matches from one set, or from combination of both sets, depends on how they differ from each other. If the context check results win significantly with comparison based on the best match similarity value, the context check results will be selected, otherwise the pre-context-check results are selected. If they are similar, user input is requested, or by default the top candidates from both sets are selected by sorting them together non-interactively.

The direction of matching also affects the quality of match results, according to our experiments and prior work [2]. Users can choose selecting match results from a single direction or both directions, based on their knowledge of the schemas. Both-direction means matching target to source and also matching source to target, if a correspondence (n_t, n_s) in the target to source mapping is selected only if target node n_t is one of top k (e.g., k = 2) almost equal candidates for source node n_s in the source to target mapping. Both-direction is the default strategy, since it performs better on the average.

User input can also help decide how many best match candidates should be selected (by default only the best one is selected), the threshold for selecting or throwing matches with lower similarity values for each target node, and what delta value for deciding multiple candidates with similar similarity values almost equally in order to detect real multiple matches for output, etc.

Significant post-match manual work can be saved by for asking user input to organize matches in the right format for further applications, and to identify necessary operations if they could not be automatically recognized, or have not been specified earlier, especially for n:m matches. Since without formal metadata, ontological information or content information, it is nearly always infeasible to recognize precise conversion functions for n:m matches, effort to develop high quality user interfaces to support conveniently specifying them seems more helpful than asking users for formal metadata or domain ontologies as hints to discover them. We have not yet provided a good interface for delivering user input conveniently, but this should be done soon.

4 Examples and results

Three application domains have been used to evaluate our approach: 1) three book DTDs obtained from the web, arts, bookstore, and a mediated one; 2) three bio-molecular DTDs, GAME, BSML and BIOML, trimmed to remove some branches that are huge but irrelevant to sequence encoding, plus a mediated DTD for gene sequence encoding only; and 3) five XML Schemas for purchase orders, CIDX, Excel, Noris, Paragon, and Apertum, used by COMA [2]. The characteristics of relevant schemas are available in the extended version of this paper [12].

A partial evaluation of the quality of our matching process can be obtained using the same measures used in [2, 10], derived from the information retrieval and data mining fields. However, as argued earlier in the paper, a metric that counts the total user effort would provide a much more appropriate evaluation, since our aim is to minimize effort, rather than to compete directly with systems that are less interactive. The manually determined real matches R for a match task are compared with the matches P returned by automatic matching process. The following are counted: the correctly identified matches T, the wrong matches $F = P - T$, and the missed matches $M = R - T$. These correspond to true positives, false positives, and false negatives respectively. Then the following three measures are computed:

$Precision = T/P = T/(T + F)$
$Recall = T/R$
$Overall = 1\text{-}(F + M)/R = (T - F)/R = Recall \times (2 - 1/Precision)$

The first gives the reliability of the mapping, the second gives the share of real matches found, and the third is a combined measure for mapping quality, taking account of the post-match effort needed for both removing wrong and adding missed matches.

5 Summary and future work

Test results in the extended version of this paper [12] show that our approach works well compared with the purchase order schema tests used by COMA [2] and Cupid [4]. The context check step seems especially helpful in 1-direction match, improving the overall score significantly. For example, in the task of matching target arts DTD with source bookstore DTD (*a-st* in Figure 5–6 of the extended version of this paper [12]) in only one direction (from the target to the source, trying to match target nodes as much as possible), the arts DTD is about both books and articles, context check help prune all mismatches for the nodes under article subtree. But without context check, `arts/article/author`, `arts/article/title` get matched to `bookstore/book/author`, `bookstore/book/title` respectively, with relatively high similarity values. For simpler tasks with few multiple-to-multiple matches, in non-interactive mode, both-direction works as well as context check. In the above example, `arts/article/author` is not an almost equal candidate to `bookstore/book/author` in the bookstore to arts mapping, so it and the others in article subtree are filtered out. User interactions help identify n:m matches with semantic functions where good 1:1 matches cannot be determined, and certify many matches with relatively low similarity values, during the matching process. Some of the overall measures are still very low because the difference of price units for book DTDs can't be detected without analyzing the data content or having more semantic metadata, or because necessary conditions can't be found, or even determined to be needed for matching, using only the available information.

Many issues remain to be studied. One is representing the generated mappings to facilitate user editing, correct false matches, add missed matches, attach restructuring and semantic conversion functions, making sure that all heterogeneities are resolved for the next step, query discovery, which is extremely important for data integration. Second, semantic metadata, including formal ontologies, could make semantic matching more convenient and reliable, for example, to help with finding n:m matches; we are now studying this problem for ecological data integration and analysis problems. Match composition and reuse also deserve further research for the above issues, and have been touched upon for the 1:1 case by [2, 6]. Finally, we wish to conduct experiments with groups of users, to validate our proposed total user effort metric for evaluating our system in its interactive mode.

Acknowledgments We thank Bertram Ludäscher for many valuable discussions, and we also thank Hong-Hai Do, Erhard Rahm and Sergey Melnik for generously sharing their data with us. This material is based upon work supported by the National Science Foundation under Grant No. ITR 0225676.

References

1. Doan. A, P. Domingos, and A. Y. Halevy: Reconciling schemas of disparate data sources: A machine-learning approach. SIGMOD'01 (2001).
2. Do, H. and E. Rahm: Coma - A System for flexible combination of schema matching approaches. Proc. 28th Conf. on Very Large Databases (VLDB) (2002).
3. Doan, A. Thesis. http://anhai.cs.uiuc.edu/home/thesis/anhai-thesis.pdf (2003)
4. Madhavan, J., P. A. Bernstein, E. Rahm: Generic Schema Matching with Cupid. Proc. 27th Int. Conf. on Very Large Data Bases (VLDB) (2001).
5. Li, W., C. Clifton: SemInt: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural network. Data and Knowledge Engineering **33:1** (2000) 49–84.
6. Madhavan, J., P. A. Bernstein, P. Domingos, & A.Y. Halevy. Representing and Reasoning About Mappings between Domain Models. In *18th National Conference on Artificial Intelligence (AAAI)*, 2002.
7. Melnik, S., E. Rahm, P. A. Bernstein: Rondo: A Programming Platform for Generic Model Management. Proc. ACM Intl. Conference on Management of Data (SIGMOD) (2003) San Diego.
8. Milo, T. and S. Zohar: Using Schema Matching to Simplify Heterogeneous Data Translation. VLDB (1998) 122–133.
9. Miller, R., L. Haas, and M.A. Hernandez: Schema Mapping as Query Discovery. VLDB (2000) 77–88.
10. Melnik, S., H. Garcia-Molina, E. Rahm: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. ICDE (2002).
11. Nam, Y. K., J. Goguen, and G. Wang: A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases. Proc. Intl. Conference on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems, Springer, Lecture Notes in Computer Science, **2519** (2002) 1332–1344.
12. Wang, G., J. Goguen, Y. K. Nam and K. Lin. Critical Points for Interactive Schema Matching. Technical Report. Dept. Computer and Engineering, UCSD (2003).
13. Rahm, E. and Bernstein, P. A.: On Matching Schemas Automatically. Techn. Report (2001) Dept. of Comp. Science, Univ. of Leipzig. http://dol.uni-leipzig.de/pub/2001-5/en
14. Biskup, J. and D. W. Embley: Extracting information from heterogeneous information sources using ontologically specified target view. Information Systems **28** (2003) 169–212.
15. Li Xu and D. Embley: Using Domain Ontologies to Discover Direct and Indirect Matches for Schema Elements. Proc. Semantic Integration Workshop (2003) Sanibel Island, Florida.
16. Do, H.H., Melnik, S., Rahm, E.: Comparison of Schema Matching Evaluations. Proc. GI-Workshop "Web and Databases", Erfurt, LNCS 2593, Springer-Verlag (2002).
17. He, B. and K. C. Chang: Statistical Schema Matching across Web Query Interfaces. In ACM Intl. Conference on Management of Data (SIGMOD) (2003) San Diego.