A Framework for Domain-Specific Interface Mapper (DSIM)

Komal Kumar Bhatia1[†] and A. K. Sharma2[††],

YMCA Institute of Engineering, Faridabad, Haryana, INDIA

Summary
Large amount of on-line information resides on the invisible web (deep or hidden web). These web pages are generated dynamically from databases and other data sources hidden from the user, such pages not indexed by a static URL, are generated only when queries are asked via a search interface rendering interface matching a critical problem in many application domains, such as: semantic web, interface integration, data warehouses, e-commerce, etc. Many different matching solutions have been proposed so far. In this paper, a novel framework for interface matching is being proposed. It employs extensible domain-specific Library for quickly identifying regions in the interface repository comprising of important mappings.

Key words:
Invisible web (deep or hidden web), Search interface, Interface matching.

1. Introduction

From this section, input the body of your manuscript according to the constitution that you had. For detailed information for authors, please refer to [1]. A multitude of search engines (A .K. Sharma et al, 2002), such as *Google* (A. K. Sharma et al, 2003), *Yahoo, Infoseek and Altavista* etc, are available to retrieve information from the huge repository of diverse information called World Wide Web. In general, these engines crawl the web by following URLs that are embedded in the Web pages. The downloaded Web pages are stored indexed into the local databases of the search engines. When a request in the form of keywords arrives, the local databases are searched and the appropriate web pages are returned.
However, there is enormous amount of the Web contents that remains untouched by the traditional search engines i.e., "invisible web" or "deep web" (Bergman et al, 2001). Infact, hidden web represents information, stored in specialized databases, only accessible through specific search interfaces (A. K. Sharma et al, 2006) created by using CGI and HTML forms or Javascript etc. However, this type of data contents are a goldmine of information, as many databases contain detailed and specific information as indicated by the study in 2000 done by brightplanet.com (S. Raghavan et al, 2001). It is suggested that the invisible web contains about 400-550 times the information of the traditional, index able World Wide Web (Burner et al, 1997). Thus, hidden web contains documents into tune of

8000 terabytes of information, the access to which would provide the following benefits:
Ability to access the invisible web would be a tremendous boost for information retrieval over the Web.
Each database contains data from a specific domain (from car prices to court cases). Therefore, highly relevant information would be obtained.
Since a query interface (B. He et al, 2004) acts as an entry point (see Fig. 1) for accessing the hidden web, interface matching (B. He et al, 2004; Do H. et al, 2002; Jayant Madhavan et al, 2001; R. Dhamankar et al, 2004; Sergey Melink et al, 2000) becomes an essential activity towards mediating queries across deep services. In fact, interface matching discovers the necessary semantic correspondences of attributes across Web interfaces. The interface matching also assumes important, especially when domain-specific applications need to search alternative resources of data in the same domain.

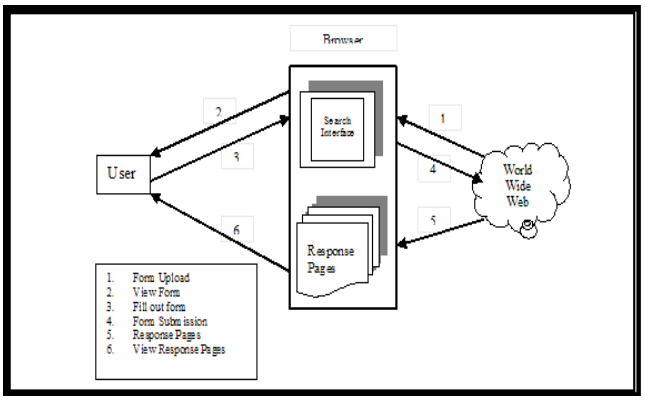

Fig. 1 User-Search Interface Interaction

In this paper, the problem of discovering semantic correspondences between attributes of different search interfaces, but in the same domain, has been identified and a framework to discover these mappings has been proposed. The paper has been organized as follows:
Section 2 describes the related work in the area of interface (or schema) matching; section 3 describes the

Manuscript received December 5, 2008
Manuscript revised December 20, 2008

proposed work i.e. a framework for Domain-specific Interface Mapper (DSIM) and explains the functionality of different components of DSIM; section 4 describes the experimental evaluation that is done over different domains and finally, section 5 draws the conclusion and describes the future research.

2. Related Work

With the easy access to information stored in different application specific databases, the problem of semantic heterogeneity (W. Wu et al, 2004) is becoming more severe, especially when the information is accessed from different resources. For example, a recent project at the GTE Telecommunications Company sought to integrate 40 databases that have a total of 27,000 attributes of relational tables. The project planners estimated that, without the database creators, just finding and documenting the semantic mappings among the elements would take more than 12-person year's (Ipeirotis, P. et al, 2001) . Nevertheless, *schema (or interface)* mapping is the task of finding semantic correspondences between elements of two schemas and plays a central role to solve the problem of *semantic heterogeneity*. In the past years, following systems and approaches have been developed to determine schema matching:

Cupid (Jayant Madhavan et al, 2001) uses a hybrid matching approach by combining a name matcher with a structural match algorithm, which derives the similarity of elements based on the similarity of their components. Therefore, this technique emphasizes on the name and data type similarities that exists at the finest level of granularity.

LSD (Learning Source Description) (A. Doan et al, 2001) and its extension GLUE represent powerful composite approaches for combining different matchers. Both techniques use machine-learning techniques for each matcher. Machine learning used by these methods is a promising technique for evaluating data instances to find out element similarity. On the other hand, the accuracy of the element similarity depends on a suitable training, which requires some manual efforts.

In COMA (Hong-Hai Do et al, 2004), the information related to previous schema matching i.e. schema mappings are stored in a *reuse library*. Given two schemas *S1* and *S2* that are to be matched, the *reuse component* tries to find a schema *S* in its reuse library for which it has stored matches between *S* and *S1*, and between *S* and *S2*. This stored information is combined to produce a new match.

In SemInt matching technique (Li W, et al,2000), semantic mappings between individual attributes of two schemas are generated by exploiting up to 15 constraint-based and 5 content-based matching criteria. Neural networks are used by this technique to determine match candidates.

A critical look at the available literature indicates that the following issues need to be addressed while designing the framework for fully automatic schema matching technique:

As the number of data sources is growing continuously at very high rate, it is very tedious, time consuming and error-prone to perform the interface matching manually in web-based applications. Therefore a faster, less error-prone and more efficient approach is required that uses fully automated approach for schema matching.

An effective schema matching method requires a combination of many matching techniques, such as linguistic matching (W. Wu et al, 2004) of names of schema elements, comparison of their data instances, considering structural similarities between schemas, and using domain knowledge and user feedback.

In this paper, a Domain Specific Interface Mapping Scheme called DSIM is being proposed. DSIM provides an extensible domain specific library of match functions to support multi-strategy matching approach. It also uses mapping knowledge base to leverage previous matching experiences.

3. Proposed architecture of Domain Specific Interface Mapper (DSIM)

The proposed *Domain Specific Interface Mapper* (DSIM) finds the semantic mappings between the components of different web interfaces of the same domain i.e. all the interfaces belong to the same domain such as airline domain. The main inputs to this Interface mapping system, as shown in Fig.2, are two interfaces A and B comprising of a number of components i.e. $\{n_1, n_2 \ldots n_p\}$ and $\{n'_1, n'_2 \ldots n'_q\}$ respectively.

DSIM uses a *Search Interface Repository* that is the repository for domain-specific search interfaces. It also provides an extensible domain-specific matcher library to support multi-strategy match approach. The multi-strategy match approach uses different matching strategies like *fuzzy matching, domain-specific thesaurus* etc that are executed independently. The SVM Generator in DSIM is used to create matrices of mapping that are identified by the matching library. The *SVM Selector* generates the appropriate mapping which can be used as the output. The DSIM also uses a *Mapping Knowledgebase* which stores the important semantic mappings so that they can be used further when after sometime our search interface repository would be updated.

The multi-strategy interface matching is carried out by the DSIM in three phases: *parsing, semantic matching and semantic mapping generation*. The working of each phase is explained as follows:

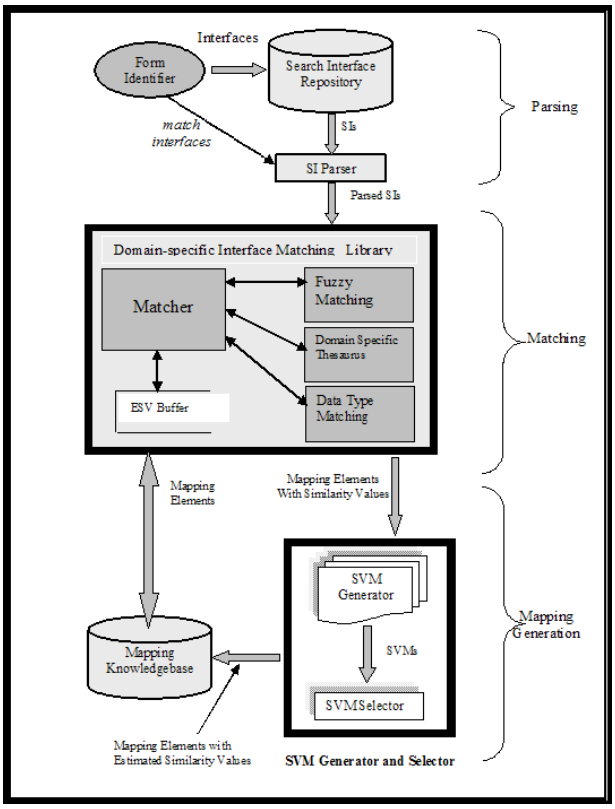

Fig 2. The Domain-specific Interface Mapper

3.1 Parsing

The parsing phase extracts the interfaces from Search Interface Repository and parses them in to an ordered tree. It uses following two components:

Search Interface repository: Form Identifier (A. K. Sharma et al, 2006) module identifies the domain specific search interfaces and stores them in the Search Interface Repository. This repository is further used by DSIM and is updated with domain-specific interfaces after a regular interval of time. Form Identifier sends a signal *"match interfaces"* to SI Parser whenever interfaces are available in the repository for parsing.

SI Parser: On receiving the signal *"match interfaces"* from form identifier, the parser extracts the interfaces from the Search Interface repository and parses them to obtain the structure of a query interface represented by using hierarchical schema as shown in Fig. 3. In fact, it is an ordered tree of elements where each leaf correspond to a field in the interface, each non-leaf node correspond to a group or super-group of the field. In fact, the order among the sibling nodes within the tree resembles the order of fields in the interface in a left-to-right and top-to-bottom fashion. For example, *<attribute 11>, <attribute 12>, <attribute 2>, <attribute 3>, <attribute 41>, <attribute 42>……..<attribute n>* represent leaves in the interface, *<attribute 1>* and *<attribute 2>* represents non-leaf nodes and root represents the form name.

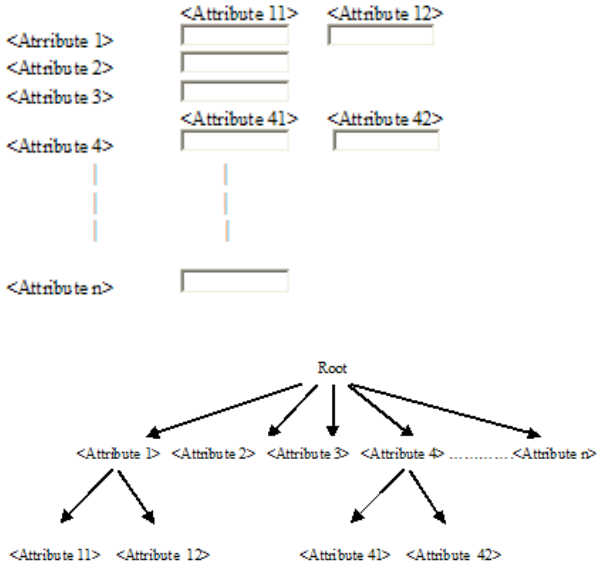

Fig. 3 Hierarchical representation of a query interface in a particular domain

Consider the example given below:

Example *Fig. 4 shows a typical example of two query interfaces in the books domain and its corresponding hierarchical representation.*

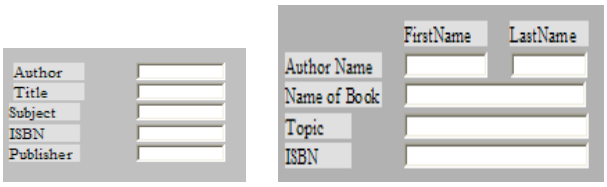

Fig. 4 Two query interfaces in book domain

It may be observed that Fig. 5(a) and Fig. 5(b) have two and three levels respectively. The root represents the name of the interface, each leaf node corresponds to a field in the interface and each non-leaf node corresponds to a group or super- group.

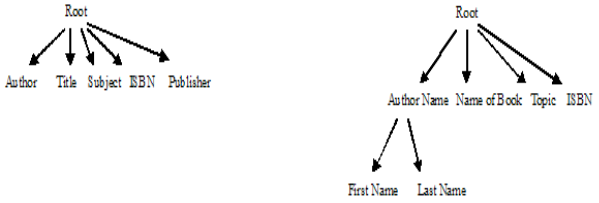

Fig. 5 (a) & (b) Hierarchical representation of two query interfaces in book domain

3.2 Semantic Matching

This phase of DSIM matches the components of two different interfaces by using a *Domain Specific Matching Library*, which is an extensible matcher library to support multi-strategy match approach. New matching strategies can be easily included in the library and used. Currently, Domain-specific Matching Library uses three types of matching strategies as given below:

3.2.1 Fuzzy Matching:

This Fuzzy Matching strategy uses a single element matcher called node name matcher (NNM). The NNM matcher is very similar to Similarity flooding's *StringMatch* and *EditDistance* matcher in COMA (Hong-Hai Do et al, 2004). The node name matcher can be implemented using the *CompareStringFuzzy* function. The *CompareStringFuzzy* function compares two strings and returns a similarity index in the range [0, 1]. The similarity index is computed based on character substitution, insertion, exclusion and transposition. Examples of the computed name similarity index are given in the third column of Table 1.

Table 1 Examples of using *CompareStringFuzzy* and *NNM* functions.

Interface Components		CompareStringFuzzy		NNM
n	n'	(n,n')	Reversed n, n'	n, n'
"Name"	"Naming"	0.86	0.00	0.86
"Name"	"Company"	0.25	0.25	0.25
"Name"	"FirstName"	0.00	0.80	0.80
"Name"	"Time"	0.55	0.83	0.83

Note that in the third row, contrary to expected, the similarity between elements "Name" and "FirstName" is zero. This is because *CompareStringFuzzy* function gives more significance to a mismatch at the beginning of a string than to a mismatch at the end. It compares not only the original strings (column 3 in the table), but also their reverse strings, i.e., "emaN" and "emaNtsriF"(column 4 in the table). The larger of two similarity indexes becomes the final similarity index (column 5 in the table).

3.2.2 Domain Specific Thesaurus:

Identifying semantic relationships between concepts or objects is very important in database schema integration and Web source integration. To facilitate component matching, the domain-specific matching Library of proposed DSIM contains domain-specific thesaurus. By using domain-specific thesaurus, the following three types of semantic relationships between attribute names or elements have been identified: *Synonymy*, *Hypernymy* and *Meronymy*.

- *Synonymy*. Two attributes *A1* and *A2* are synonyms if they have similar meanings.
- *Hypernymy.* Attribute *A1* is a *hypernym* of attribute *A2* if *A1* is more generic than *A2.* For example, *tree* is a hypernym of *mango*.
- *Meronymy.* Attribute *A1* is a *meronym* of attribute *A2* if *A1* is a part of *A2.* For example, *first name* is a meronym of *name*.

In DSIM's *Domain-specific Thesaurus*, hypernymy and meronymy relationships of two terms using the information in the interface representations are also identified (see Table 2).

Table 2 Examples of Domain-specific Thesaurus for *Books* domain

Attribute Name (T1)	Interface Name	Attribute Name (T2)	Interface Name	Relationship
"Name"	"Books"	"First Name"	"RefBooks"	Meronym (T2, T1)
"Name"	"Books"	"Last Name"	"RefBooks"	Meronym (T2, T1)
"Format"	"Books"	"Hardcover"	"RefBooks"	Hypernym(T1, T2)
"Name"	"Books"	"Author Name"	"Refer"	Synonymy(T1, T2)
"Title"	"Books"	"Book Name"	"Refer"	Synonymy(T1, T2)

Let us consider two interfaces consisting of attributes *"hardcover"* and *"format"* respectively. If the value of *format* is hardcover, obviously, the attribute *format* becomes hypernym of *hardcover* (see row 3). Thus, the existence of hypernym in an interface helps in mapping process. Also, the *part* relationships of elements may be used to discover meronyms. Consider a search interface that contains an attribute *author* with two parts, *first name* (row 1) and *last name* (row 2). From this interface, both *first name* and *last name* can be identified as meronym of *author*.

3.2.3 Data Type Matching

This matching strategy is used to match the data types of the attributes of two different interfaces. It considers only two data types: *numeric and string*. The data types of two attributes of different interfaces would match only if both attributes are of same type (see Table 3).

Table 3 Examples of Data Type Matching in Books domain

Attribute Name (T1)	Data Type of attribute T1	Interface Name	Attribute Name (T2)	Data Type of attribute T2	Interface Name
"Name"	String	"Books"	"First Name"	String	"RefBooks"
"Format"	String	"Books"	"Hardcover"	String	"RefBooks"
"Total Pages"	String	"Books"	"Pages"	Numeric	"Refer"
"Rate"	Numeric	"Books"	"Price"	Numeric	"Refer"

For example, for attributes A1 and A2 in two different interfaces, the matcher can return the following similarity values in each case:

While using, **fuzzy matching** strategy, the matcher compares two strings and returns a similarity value in the range [0, 1]. For example, for attributes "author" and "first Name" of two different interfaces (see Table 1), the matcher returns 0.80 as the similarity value.

While using, **Domain-specific Thesaurus**, the matcher returns either 0 or 1 as the similarity value. The matcher returns 0 if there exists no relationship between attributes A1 and A2 of two different interfaces. If any relationship (synonymy, hypernymy or meronymy) exists between two attributes of different interfaces, the matcher returns 1. For example, for attributes "author" and "first name" of two different interfaces "Books" and "RefBooks" (see row 1 of Table 2) respectively, the matcher returns 1 as the similarity value.

Similarly, for **Data type matching** strategy, the matcher also return 1 as similarity value only if data type of two attributes of different interfaces are same else returns 0. For instance, for attributes "author" and "first name" of two different interfaces "Books" and "RefBooks" (see row 1 of Table 3) respectively, the matcher returns 1 as both attributes are of type *string.* But for attributes "Total pages" and "pages" (row 3 of Table 3), the matcher returns 0 as both attributes are not of same type.

The overall similarity value also called estimated similarity value is computed as an average of the similarity values obtained from the matcher for each mapping by using three matching strategies: fuzzy matching, *Domain-specific Thesaurus and Data Type Matching.* The estimated similarity value is further used by the Similarity Value Matrices (SVMs) as explained in the next section.

3.3 Semantic Mapping Generation

This phase generates the necessary semantic mappings between the attributes of interfaces and stores them in mapping knowledge base for future reference. It uses following functional components for this purpose:

3.3.1 SVM generator and Selector

The state of system is represented by a triplet (S, O, SVM), where S is the set of attribute of one interface, O is the set of attributes of another interface, and SVM is a matrix, called Similarity Value Matrix (SVM), with a row for every attribute of one interface and a column for every attribute of another interface. The schematic diagram of a SVM is shown in Fig. 6 (a). Let variables s and o denote the attributes of one interface and another interface, respectively. An entry SVM[s, o] denotes the Estimated Similarity Value (ESV) between attribute s of first interface and attribute o of another interface. Fig. 6 (b) shows a SVM for interfaces F1 and F2 and Fig. 6 (c) shows a SVM for interfaces T1 and T2.

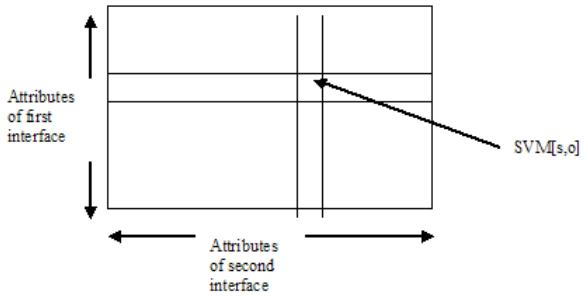

Fig. 6(a) Schematic diagram of Similarity Value Matrix

	Author	Name of Book	Topic	ISBN
Author	1.0	0.3	0.5	0.1
Title	0.2	0.8	0.5	0.15
Subject	0.35	0.45	0.75	0.12
ISBN	0.2	0.3	0.4	0.99
Publisher	0.3	0.1	0.35	0.4

Fig. 6 (b) SVM for interfaces F1 and F2

	FName	Book Name	ISBN	Name of Topic
Author	0.85	0.3	0.5	0.1
Title	0.2	0.8	0.5	0.15
Subject	0.35	0.45	0.5	0.12
ISBN	0.2	0.3	0.99	0.27
Publisher	0.3	0.1	0.35	0.4

Fig. 6 (c) SVM for interfaces T1 and T2

Fig. 6 Example of two Similarity Value Matrices

In fact, SVM generator identifies the estimated similarity values as returned by the *Domain-specific Matching Library* and generates the different Similarity Value Matrices (SVMs). Each entry in the SVMs shows interface mappings between the attributes of two interfaces. In a real system, some of interface mappings could be irrelevant and of no importance to the user. Since a good interface mapping would not spread over the repository and will remain located in rather small region of the repository, DSIM employs a SVM Selector which checks all the SVMs and their estimated similarity values. It also looks for mappings which are more valuable, instead of using all the mappings that are given by the Domain-specific Matching Library, thereby improving the efficiency of interface matching.

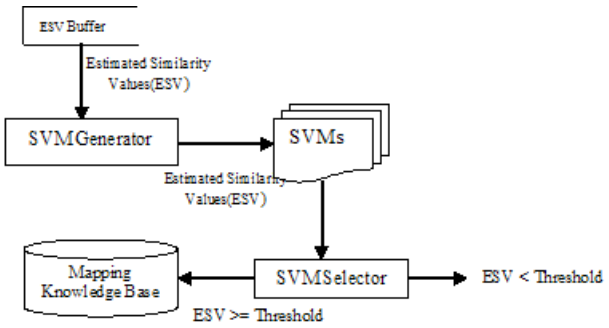

Fig. 7 Detailed Schematic Diagram of SVM Generator and Selector module

To find the more valuable mappings, the SVM Selector uses a threshold value as the selection parameter. The threshold value is compared with the estimated similarity values for each SVM to find the more valuable mappings. The mappings having estimated similarity values greater than the threshold values are being stored in the Mapping Knowledge Base for future reference (see Fig. 7). The mappings having estimated similarity value below a particular threshold value would be ignored. For example, if an attribute *author* of first interface is matched with attributes *first name, ISBN and subject* of the second interface (see Fig. 8), the matcher returns estimated similarity values for each of three mappings i.e. for *author* and *First Name*, *author* and *ISBN* and for *author* and *subject*.

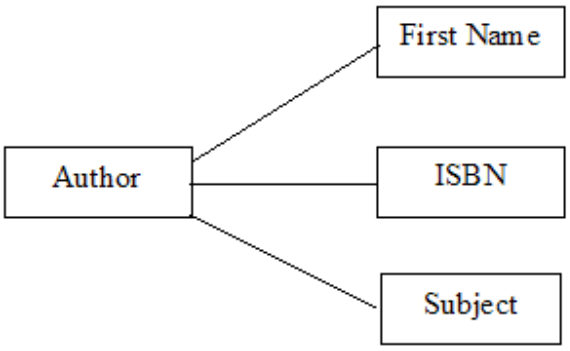

Fig. 8 An Example of Matching process

The estimated similarity values of these three mappings are compared with a threshold value. The mappings having estimated similarity values greater than threshold value are treated as important mappings and are stored in Mapping Knowledge Base for future reference and all other mappings having estimated similarity values less than threshold value are discarded. The algorithm for SVM Generator and SVM Selector is shown in Fig. 9.

Algorithm SVMGenSelector*(Iface1, Iface2)*

/ This algorithm stores the estimated similarity values in the SVMs. */*

/ The attributes of the first and second interfaces are represented by rows and columns respectively. */*

/ It calls SVMSelector Algorithm for selecting valuable mappings. */*

Begin

Store the estimated similarity values for attributes of Iface1 and Iface2 in SVM_{m*n}*;*

*SVMSelector (SVMm*n);*

end;

Algorithm SVMSelector*(SVMm*n)*

*/*This algorithm finds the most valuable mappings by comparing the estimated similarity values in SVMs with the appropriate threshold value and store them in a Mapping Knowledge Base if the mappings are not already there.*/*

begin

for i = 1 to m do

for j = 1 to n do

begin

if (SVM[i, j] >= Threshold Value) then

if (SVM[I,j] does not exists in Mapping Knowledge Base) then

```
        Store SVM[i, j] for Iface1 and Iface2 in Mapping Knowledge Base
        with attributes names;
            else
                continue;
        end;
    end;
```

Fig. 9 Algorithm for SVM Generator and Selector

3.4 Mapping Knowledgebase

The large numbers of mappings produced by SVM Selector are stored in Mapping Knowledge Base (see Fig. 10). It contains five fields: *attribute I, interface I, attribute II, interface II, estimated similarity value.* In order to avoid duplicacy of match effort and redundancy of storage, the matcher consults the Knowledge Base in search / insert fashion. Before starting the next match cycle, the matcher searches the mappings in the Knowledge Base. If the mapping entry is found then they are discarded else the match process is continued in order to derive new semantic mappings. The mappings so obtained are inserted in the Knowledge Base.

Attribute I	InterfaceI	Attribute II	InterfaceII	EstimatedSimilarity Value
Author	F1	Author	F2	0.99
Author	T1	First Name	T2	0.81
Subject	F1	Topic Name	F2	0.72
ISBN	F1	ISBN	F2	1.00

Fig. 10 Structure of Mapping Knowledge Base

4. Experimental Evaluation

Extensive experiments were conducted over several domains of sources on the Web with the goal to evaluate accuracy of matching and the contribution of different components of DSIM towards the same. Although results differ for different domains, the DSIM has shown fairly general behavior with consistent results.

4.1. Performance Metrics

Similar to (B. He. Et al, 2003), the performance of field matching has been measured via three metrics: precision, recall, and F-measure. *Precision* is the percentage of correct mappings over all mappings identified by the system, while *Recall* is the percentage of correct mappings identified by the system over all mappings as given by domain experts. Suppose the number of correctly identified mappings is C, the number of wrongly identified mappings is W and the number of unidentified correct mappings is M, then the precision of the approach is given by the expression given below

$$P = C/(C + W) \quad (1)$$

and the recall, R, of the approach is

$$R = C/(C + M) \quad (2)$$

F-measure incorporates both precision and recall. F-measure is given by

$$F = 2PR/(P + R) \quad (3)$$

where : precision P and recall R are equally weighted.
In DSIM, schema trees were generated from the query interfaces collected from five domains. The schema trees have been used to conduct the experiments. For every domain average precision, average recall and average F-measure were computed by the expressions given below:

$$\text{Average Precision} = \sum P_i / N \quad (4)$$

$$\text{Average recall} = \sum R_i / N \quad (5)$$

$$\text{Average F-measure} = \sum (\text{F-measure})_i / N \quad (6)$$

where N is total no. of interfaces matched and i range from 1 to N.

4.2. Data Set

For experimental evaluation of the proposed work, query interfaces of the sources available on the deep Web for the following five domains have been considered:

- *Airfare*
- *Automobile*
- *Book*
- *Job*
- *real estate*

For each domain, 20-30 query interfaces were collected by utilizing the online directories: invisibleweb.com (now profusion.com) which maintains a directory of hidden sources along with their query interfaces and the Web directory maintained by yahoo.com. Yahoo.com houses both hidden and visible web. The query interfaces of hidden web were obtained and manually transformed into schema trees.

4.3. Experiments

In the proposed work, SVM generator identifies the estimated similarity values as returned by the *Domain-specific Matching Library* and generates the different Similarity Value Matrices (SVMs). The SVM Selector uses a *threshold value* as a selection parameter which is compared with the estimated similarity values for each SVM to find the more valuable mappings. The mappings with estimated similarity values greater than a particular threshold are stored in the Mapping Knowledge Base. The various strategies of *Domain Specific Matching Library (DSML)* were employed to perform same set of experiments for each domain using different threshold values. *Average Threshold* of a particular domain is computed by the following expression

$$Average\ Threshold = \sum T_i / N \qquad (7)$$

where N is total number of interfaces matched and i range from 1 to N.
The results of the experiments are shown in Table 4.

Table 4. *Average Threshold* for each domain

S. No.	Domain	Average Threshold
1	Book	0.81
2	Airfare	0.79
3	Auto	0.80
4	Job	0.78
Overall Average Threshold		**0.795**

It may be observed that the *Overall Average Threshold* for all the five domains is about *0.80.* The *Average Threshold* values listed in Table 4 and expressions (4), (5) and (6) were used to compute *Average Precision, Average Recall*and *Average F-measure* for every domain and the results are tabulated in Table 5.

Table 5. Experimental results in five domains

Domain	Average Precision	Average Recall	Average F-measure
Book	92.5	91.3	91.8
Airfare	92.2	90.4	91.2
Auto	89.8	92.7	91.2
Job	84.6	89.3	86.8
Overall Average	**89.7**	**90.9**	**90.2**

From Table 5, it may be observed that overall average precision of matching process is high i.e. ranges from 84% to 93.5%, overall average recall of matching process is also high i.e. ranges from 85.5% to 96% and overall average F-measure of matching process is also quite high i.e. about 90%.
If the estimated similarity values of mappings are higher than threshold values, the semantic mappings thus found by the matching process are stored in the *Mapping Knowledgebase*. In fact, these mappings could be used if after sometime search interface repository would be updated or new search interface are inserted in the *Search Interface Repository*.

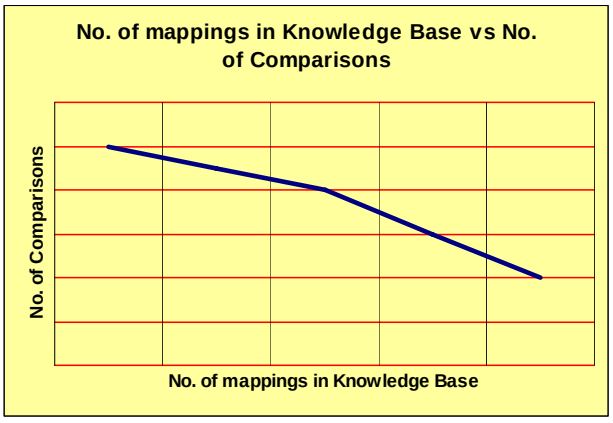

Fig. 11 No. of mappings in Knowledge Base vs No. of Comparisons

It has been found (see Fig. 11) that the number of comparisons done in the matching process decrease as the number of mappings in the mapping Knowledge Base increase indicating that the framework proposed in this paper is both scalable and efficient. Moreover, the efficiency of DSIM increases as the number of mappings in the mapping knowledge base increases.

5. Conclusion and Future research

DSIM quickly identifies regions in the interface repository comprising of important mappings. It further improves by discarding the less important mappings as SVM Selector uses a *threshold value* as a selection parameter. The loss mostly occurs among the mappings which rank low, an acceptable trade off.

Though the domain-specific Interface Matching Library currently supports three matching strategies, but it is extensible in the sense that newer and better strategies can be easily added later on.

Future research includes: (1) establishing a tighter control over selection of the important mappings – more insight into the effects of certain selection parameters on the efficiency/effectiveness trade off allows for better tuning, (2) ordering the mappings in the Mapping Knowledge Base – a measure of mapping's quality can be used to decide which mappings have better chances to produce good mappings. In this way, the time-to-first good mapping can be improved, (3) extending the match library and improving the learning capability.

Acknowledgments

Insert acknowledgment, if any.

References

[1] Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to Map between Ontologies on the Semantic Web. In *WWW*, 2002.

[2] A. Doan, P. Domingos, A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. In SIGMOD Record, 2001.

[3] A.K.Sharma, J. P. Gupta, "An Architecture for Electronic Commerce on the Internet", Journal of Continuing Engineering Education, Vol. 2, pp 10-15, Roorkee, July 2002.

[4] A.K.Sharma, J. P. Gupta, D. P. Agarwal, "A novel approach towards Volatile Information Management", Journal of CSI, Vol. 33 No. 1, pp 18-27, Sept' 2003.

[5] A.K. Sharma, J.P. Gupta, D. P. Agarwal, "An alternative approach for generation of document fingerprints for static documents", Journal of CSI, Vol. 35 No. 1, pp 18-27, Mar 2005

[6] A. K. Sharma, Komal Kumar Bhatia: "Automated Discovery of Task Oriented Search Interfaces through Augmented Hypertext Documents" Proc. First International Conference on Web Engineering & Application (ICWA2006).

[7] Alexandros Ntoulas Petros Zerfos Junghoo Cho, "Downloading Hidden Web Content", UCLA Computer Science, fntoulas, pzerfos, chog@cs.ucla.edu He, K. Chang, and J. Han. Discovering complex matching

[8] across web query interfaces: A correlation mining approach. In *SIGKDD*, 2004.

[9] He, K. Chang, and J. Han. Statistical schema matching across web query interfaces. In *SIGMOD*, pages 217–228, 2003.

[10] Bergman, M.K., *The Deep Web: Surfacing Hidden Value*. 2000,

[11] BrightPlanet.com,Sullivan, D., *Search Engine sizes.* The Search Engine Report, 2001.

[12] Do H.-H., Melnik S., and Rahm E.: Comparison of Schema Matching Evaluations, Proc. GI Workshop "Web and Databases", Erfurt, Oct. 2002.

[13] *Google.com*., www.google.com

[14] H. He, W. Meng, C. Yu, and Z. Wu. WISE-integrator: An automatic integrator of Web search interfaces for e-commerce. In *VLDB*, 2003.

[15] Hong-Hai Do, Erhan Rahm, COMA-A system for flexible combination of schema matching approaches. In Proc. 28th VLDB Conference, 2004.

[16] Ipeirotis, P., Gravano, L. & Mehran, S. (2001), 'Probe, count, and classify: categorizing hidden web databases', ACM SIGMOD 30(2), 67 – 78.

[17] Jayant Madhavan, Philip A. Bernstein, Erhard Rahm, Generic Schema Matching with Cupid, VLDB 2001.

[18] Li W, Clifton C, Liu S (2000) Database integration using neural network: implementation and experiences. Knowl Inf Syst 2(1): 73-96.

[19] Mike Burner, "Crawling towards Eternity: Building an archive of the World Wide Web" Web Techniques Magazine, 2[5], May 1997.

[20] N. Noy and M. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *AAAI/IAAI*, pages 450–455, 2000.

[21] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. iMAP: Discovering Complex Mappings between Database Schemas. In *SIGMOD*, 2004.

[22] S. Raghavan and H. Garcia-Molina, "Crawling the Hidden Web", VLDB Conference, 2001.

[23] Sergey Melink, Hector Garcia-Molina, Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. In Proc. 18th International.Conf. On Data Engineering, San Jose CA, 2002.

[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] *www.invisibleweb.com*.

Komal Kumar Bhatia received the B.E. and M.Tech. degrees in Computer Science Engineering with Hons. from Maharishi Dayanand University in 2001 and 2004, respectively. Presently, he is working as Assistant Professor in Information Technology department in YMCA Institute of Engineering, Faridabad. He is also pursuing his Ph.D in Computer Engineering and his areas of interests are Search Engines, Crawlers and Hidden Web.

Prof. A. K. Sharma received his M.Tech. (Computer Sci. & Tech) with Hons. From University of Roorkee in the year 1989 and Ph.D (Fuzzy Expert Systems) from JMI, New Delhi in the year 2000. From July 1992 to April 2002, he served as Assistant Professor and became Professor in Computer Engg. at YMCA Institute of Engineering Faridabad in April 2002. He obtained his second Ph.D. in IT from IIIT & M, Gwalior in the year 2004. His research Interest include Fuzzy Systems, Object Oriented Programming, Knowledge Representation and Internet Technologies.