RC24424 (W0711-114) November 19, 2007
Computer Science

IBM Research Report

User Collaborative Semantic Web Service Matching for Legacy Application Integration – A Case Study

Hua Fang Tan
IBM Research Division
China Research Laboratory
Building 19, Zhouguancun Software Park
8 Dongbeiwang West Road, Haidian District
Beijing, 100094
P.R.China

Rama Akkiraju, Anca-Andreea Ivan, Richard Goodwin
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598 USA

Diane Knappenberger
IBM Mail Stop: 025A
6300 Diagonal Highway
Boulder, CO 80301 USA

User Collaborative Semantic Web Service Matching for Legacy Application Integration– A Case Study

Hua Fang Tan[1], Rama Akkiraju[2], Anca-Andreea Ivan[2], Richard Goodwin[2] and Diane Knappenberger[3]
[1]IBM China Research Lab, Building 19 Zhongguancun Software Park, 8 Dongbeiwang WestRoad, Beijing,P.R.C.100094
[2]IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY, 10532, USA
[3]IBM Mail Stop: 025A 6300 Diagonal Highway Boulder, CO 80301, USA
{tanhuaf@cn, akkiraju@us, ancaivan@us, rgoodwin@us, kberger@us}.ibm.com

Abstract

Legacy application integration is often tedious and laborious. IT consultants do not have many tools at their disposal that can help them in their efforts. Semantics-based intelligent tools can alleviate some of the burden on IT consultants. In this paper we present the results of a case study on the application of our semantic Web service matching system to a real-world legacy application integration problem. Experimental results on our initial data sets indicate that the system on its own is able to achieve close to an average of 50% recall and 54% precision measures in matching the parameters that define the interfaces of business applications. Additional performance enhancements of up to 95% recall and 72% precision measures were achieved with the usage of domain ontologies created by an expert. Therefore, we believe a user working with a semantics-based intelligence tool could gain significant productivity enhancements in IT projects rather than working alone. These productivity enhancements can be quickly translated into savings in time and money in business process integration projects.

1 Introduction

Business process integration is among the most important challenges faced by organizations today. Integrating disparate business applications is at the heart of this challenge. In the context of a large enterprise that was created through mergers and acquisitions, there are often thousands of applications and data repositories, each with associated services and interface definitions. Since these interfaces were created for different organizations, over different periods of time and for different initial purposes, it is unlikely that they use a common set of terms to name services and parameters. This leads to substantial heterogeneity in syntax, structure and semantics. For example, what one service interface in one system may encode as *itemID*, *dueDate*, and *quantity* may be referred to by another service interface in a different application as *UPC* (Universal Part Code), *itemDeliveryTime* and *numItems.* A substantial amount of developer time is spent in identifying these kinds of semantic ambiguities and resolving them.

In current practice, much of this interface mapping is done by consultants manually. Analysts and developers typically pour over large spreadsheets or XML documents that describe the interface elements of each application and manually create the mappings between source application parameters and target application parameters. This process is tedious and laborious. Semantics-based intelligent tools can alleviate some of the burden on IT consultants. The objective of semantic Web service matching research is to help develop (semi) automated tools to help resolve these types of syntactic, structural and semantic differences.

In this paper, we present the results of a case study on the application of our semantic Web service matching system to a real-world legacy application integration problem. The objective of the case study is to assess the performance improvements that can be noted when an IT consultant works with our semantic Web service matching tool in comparison to doing it manually herself. In our work, we model the interfaces of business applications that need to be integrated as semantic Web services represented in WSDL-S [WSDL-S]. We, then perform semantic matching of these Web services to find the semantic similarities between the interfaces of their corresponding business applications.

Considerable amount of work has been done on schema matching [Madhavan *et al.* 2001] [Rahm *et al.* 2001] and more recently on Web service matching [Sycara *et al.*, 1999] [Dong *et al.*, 2004] [Syeda-Mahmood *et al.*, 2005]. Specifically, previously, Syeda-Mahmood et. al [Syeda-Mahmood *et al.*, 2005] have shown that combining domain-independent and domain-specific ontologies to determine the semantic similarity between ambiguous concepts/terms yields better results while matching Web services interfaces than could be obtained using any one cue alone. In their work, the domain-independent relationships are derived using an English thesaurus after tokenization and part-of-speech tagging. The domain-specific ontological similarity is derived by inferring the semantic annotations associated with Web service descriptions using an ontology. Matches due to the two cues are combined to determine an overall similarity score. Our semantic matching engine is based on this work.

The rest of the paper is organized as follows. Section 2 introduces two accounting applications and their interfaces and calls out the interesting characteristics of these applications that have a bearing on automatic matching. In Section 3, we present an overview of our semantic matching engine so as to establish the context for the experiments to follow. We discuss the details of our experimental design in Section 4. In Section 5, we present the experimental results and draw the insights obtained from conducting the case study. Finally we conclude in Section 6 with a summary and future work items.

2 Matching the Interfaces of Accounting Applications – A Case Study Scenario

Our case study scenario consists of integrating two real-world accounting applications; Application A and Application B. Application A keeps track of various identifiers related to project and account controls. These identifiers are used within the general ledger as well as the systems that feed the ledger to track cost and revenue. They are also used for labor claiming and several types of reporting. Some of the types of these identifiers are: Customer Groups, Account Groups, Accounts, Billing Code, and Ledger Project Ids.

The client had implemented a new business management suite of applications. As part of this effort, the customer billing management is to be handled by another system called Application B. Application A and B share some common information related to customer accounts, billing codes and contractual information. Therefore, the key identifiers and overlapping information in these two systems have to be kept in synchronization to ensure data integrity across the entire business management suite of applications. At the heart of this data synchronization task is the challenge of mapping the interface parameters of these two systems. Since Application B has been developed by a vendor different from the one that developed Application A, there are both semantic and structural differences between the interfaces of the two systems. For example, what Application A calls as *AccountGroup* is referred to by Application B as a *Contract*. Similarly, a *Department* in Application A is called a *ServiceOffice* in Application B. A short list of such vocabulary differences is shown in Table 1. The task of an IT consultant is to find these commonalities and differences and write the glue code required to enable smooth data synchronization.

Application A	Application B
Account Group	Contract
Account	Worker Number
Department	Service Office
Customer Group Level 3	Country
Competency Code	Service Line

Table 1: Some terminology differences between Application A and Application B

In this case study, we are interested in applying our semantics-based intelligent tool developed using semantic Web services matching technology. The hypothesis of this effort is that an IT consultant working with a semantic Web service matching tool can be more productive in IT integration projects than doing it manually herself. Detailed evaluation criteria are discussed in the Experimental Design and Results sections. To investigate this, we selected three transactions that trigger the sending of messages from Application A to Application B. These transactions typically involve Application A sending a subset of information to Application B and in return receiving some updated information and confirmation codes from Application B.

We model these transactions as three Web services represented in WSDL [Christenson *et al* 2001] format, and then match the interfaces of these sets of Web services using our semantic Web service matching technology. Below, we present some of the characteristics of the interfaces of the two systems that have an impact on automatic matching.

1) 80% of the parameters are composed of multi--part-words (eg: DATE_OPEN, START_DATE etc.).
2) 70% of the parameters have one or more than one abbreviation (eg: CUST_ID for Customer Identifier and, CLNT_CDE for Client Code etc.).
3) Many superficial differences in semantics exist. Some of such differences are outlined below.
   a) Ordering differences: eg: “DATE_LAST_UPD”-“Last_update_date”.
   b) Synonyms: “DATE_BEGIN”-“START_DATE”.
   c) Terminology differences: Such as the ones listed in table 1.
4) Some of semantic information has been omitted when developers named the parameters. For example, a transaction that gets a billing code (called getBillingCode()) takes a parameter called ‘description’. In this case, the parameter ‘description’ can be complemented with contextual information (we refer to this as semantic annotation) such as “BillingCodeDescription”.
5) Some of semantic information that is omitted by developers can not be complemented by the contextual information. For example, in a parameter that is meant to represent “TOTAL_HOURS” ‘TOTAL’ has been omitted and it was called ‘HOURS’. Similarly, a parameter that is meant to represent “estimated_hours”, ‘estimated’ was omitted and it was simply called ‘hours’. This could lead to false positives during matching. Once eliminated this type of contextual information cannot be inferred unless it is explicitly stated in a domain model.

These characteristics of the problem have a bearing on the performance of the heuristics and algorithms that are implemented in our semantic matching engine. To help set the context for some of the results presented in Section 4, we introduce our semantic matching engine in Section 3 below.

3 An Overview of Our Semantic Matching Engine

Our semantic Web service matching engine is based on [Syeda-Mahmood *et. al* 2005]. We model both a client's request and a service provider's service as Web services. These Web services can be optionally semantically annotated with contextual information. We use WSDL-S [Akkiraju *et al.* 2005] mechanism to add semantic annotations to WSDL documents. The semantic matching engine uses domain-independent and domain-specific ontologies to determine the semantic similarity between ambiguous concepts/terms in a WSDL document. The domain-independent relationships are derived using an English thesaurus after tokenization, part-of-speech tagging and abbreviation expansion. The domain-specific ontological similarity is derived by inferring the semantic annotations associated with Web service descriptions using an ontology. Matches due to the two cues are combined to determine an overall similarity score. Figure 1 presents a component view of our semantic matching engine.

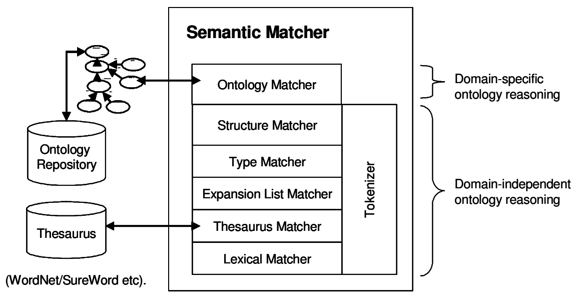

Figure 1: A component view of our semantic matching engine.

Below we discuss those parts of the system that are relevant to matching the interfaces of Application A and Application B of our case study. More details about the inner workings of the system are presented in the paper [Syeda-Mahmood et. al 2005].

- *Lexical Matching (L)*: Words are matched for their exact lexical similarity._Eg: 'country_cd' on one side matches with 'COUNTRY_CD' on the other side.
- *Word Tokenization (T)*: Words are tokenized based on the common naming_conventions used by programmers such as underscore, spaces, and camel case letters etc. For example, the "BillingCode" will be tokenized into two tokens "Billing" and "Code" respectively. This allows for semantic matching of the attributes.
- *Abbreviation expansion (X)*: The abbreviation expansion uses domain-independent as well as domain-specific vocabularies. It is possible to have multiple expansions for a candidate words. All such words are retained for later processing. Thus, a word such as "CustPurch" will be expanded into CustomerPurchase, CustomaryPurchase, etc.
- *Using domain-independent ontologies (D)*: We used the WordNet dictionary/thesaurus [16] to find matching synonyms and other related concepts to words. For example, the term FINISH in the multi-part word DATE_FINISH is a synonym to the term END in END_DATE.
- *Using domain-specific ontologies (O):* Domain-specific ontologies can be created by an expert to provide additional contextual information to interface parameters. For example, in our case study, an expert could create an ontology relating the corresponding terms given in table 1 and annotate the parameters of the interfaces accordingly. These annotations help in making matches that could not otherwise have been made.

At a high-level, the semantic matcher works as follows. The domain independent reasoning module takes the multi-term words of a given Web service interface and parses them into tokens. Abbreviation expansion is done for the retained words if necessary, and then a thesaurus is used to find the similarity of the tokens based on synonyms. The resulting synonyms are assembled back to determine matches to candidate multi-term word parameters from another Web service interface that it is being matched with (whose parameters are also treated using the same approach). Scoring is done as follows in the case of domain independent reasoner.

The semantic similarity between parameters p_{Ai} and p_{Bj} (that belong to Application A and B respectively) consisting of tokens m and n respectively, where k tokens are matched is given by: Score (p_{Ai}, p_{Bj}) = *Min* {(k/m), (k/n)}. For example, the semantic similarity score between the parameters "Control_Country_CD" (say p_{Ai}) and "Country_CD" (say p_{Bj}) would be 0.67. This is so because two of the three tokens in p_{Ai} match with the two of the two tokens in p_j. Therefore, Score (p_{Ai}, p_{Bj}) = *Min* {(2/3), (2/2)}.

If the parameters p_{Ai}, p_{Bj} have semantic annotations from a domain ontology, then the reasoner computes the similarity score as follows. Relationships *subClassOf(*p_{Ai}, p_{Bj}*)*, *subClassOf(*p_{Bj}, p_{Ai}*)*, and *typeOf (*p_{Ai}, p_{Bj}*)* all are given a score of 0.5, *sameAs(*p_{Ai}, p_{Bj}*)* is given a score of 1 and no relationship gets a score of 0. For example, if both '"Control_Country_CD" and "Country_CD" had the same semantic annotation say 'domainOntology#CountryCode', then they would match exactly and get a score of 1 according to the domain reasoner. In cases where we can compute a score from domain-independent reasoner as well as from a domain-specific reasoner, we use a winner-take-all approach (our system implements other schemes for score combination as well but we use this winner-take-all scheme here).

Therefore, the parameters "Control_Country_CD" and "Country_CD" match with a score of 1.

In the next section, we describe the experimental design.

4 Experimental Design

As was noted, the objective of this case study was to assess the productivity enhancements that an IT consultant can get by using a tool to achieve application interface matching in comparison to doing it manually herself. We have selected three sets of interfaces that needed be matched between Application A and Application B, where each interface pair consisting of dozens of parameters (altogether about 85 parameters in total). While noting the possible productivity enhancements that can obtained, we were also interested in comparing the incremental benefit of each approach/heuristic in our engine. Therefore, we have structured the experiments based on the heuristics introduced in section 3 represented by the letters L, T, X, D, and O.

1. Experiment L: Matching interfaces using lexical matching approach alone.
2. Experiment L+D: Matching interfaces using lexical matching and domain-independent dictionary/ontology approaches.
3. Experiment L+T+D: Matching interfaces using lexical matching, tokenization, and domain-independent dictionary approaches.
4. Experiment L+T+D+X: Matching interfaces using lexical matching, tokenization, domain-independent dictionary and abbreviation expansion approaches.
5. Experiment L+T+D+X+O: Matching interfaces using lexical matching, tokenization, domain-independent dictionary, abbreviation expansion and domain-specific ontology approaches.

To evaluate these experiments we define the following:
P_A = {the set of all parameters $p_{A1}....p_{An}$ in a Web service interface of Application A}
P_B = {the set of all parameters $p_{B1}....p_{Bn}$ in a Web service interface of Application B}
match pair (p_{Ai}, p_{Bj}) = a mapping between the parameters p_{Ai} in a Web service interface of Application A and p_{Bj} in a Web service interface of Application B. (Please note that a match pair represents the existence of a link between p_{Ai} and p_{Bj}. It does not say anything about whether it is correct or not.)
Generated Matches (P_A, P_B) (G) = {the set of all match pairs generated by the semantic matching engine between P_A and P_B}
Expected (E) = {the set of match pairs labeled by the expert between P_A and P_B i.e., 'ground truth'}
Correct (C) = {G ∩ E} i.e., the number of match pairs which are generated by our semantic matching engine between P_A and P_B that are also in the set E.
Incorrect (I) = {G – E} i.e., the number of match pairs which are generated by our semantic matching engine but not in the set E.
Missing (M) = {E – G} i.e., the number of match pairs which are in the set E but not generated by our semantic matching engine.
Precision (P): C/(C+I) = C/G
Recall (R): C/(C+M) = C/E
F-measure: 2*P*R/(P+R） i.e.，the weighted harmonic mean of precision and recall

To obtain (E), we worked with an IT consultant who had worked on the actual integration of Applications A and B for the client and gathered the expected results. This served as our 'ground truth'. *Precision* measure gives an indication of the quality of matches while *recall* measure helps measure the quantity of matches. F-measure is a harmonic mean of precision and recall metrics. It is indicative of both the quality and the quantity of matches that a semantic Web services system is able to make. Higher precision, recall and F-measures indicate that the system is able to find most number of expected (correct) matches while matching the interfaces of applications, while keeping the number of incorrect matches as low as possible. This would be encouraging because it would mean that an IT consultant doesn't have to find these matches manually. Therefore, higher precision and recall measure indicate the possible productivity enhancements that can be obtained.

One has to note that our approach is for a user/IT consultant to collaboratively with the system to achieve these results. It is not a fully automated system. A consultant is expected to work with the system to provide abbreviation expansions and domain ontologies where needed to achieve this. Such an initial effort on creating abbreviation expansions and domain ontologies etc. would be justifiable if multiple transactions occur between any given applications that need to be integrated. The initial effort pays-off in the long run because once provided, this information can be reused in matching the multiple interface pairs of the applications.

5 Experimental Results

For each service interface pair that needed to be matched we ran five experiments introduced earlier namely L, L+D, L+T+D, L+T+D+X, and L+T+D+X+O. For each experiment the number of correct and incorrect match pairs obtained was noted. Figures 2-10 represent the recall and the precision values computed for each of the three experiments, when increasing the service matching score threshold. We varied the service matching score threshold because the number of 'close' matches obtained depends on the amount of semantic ambiguity that was allowed to be resolved. For example, the parameters 'Control-Country-CD' and 'Country-CD' will be considered a match only if the semantic similarity threshold is set to below 0.6. This is so because the threshold allows only those match pairs whose match score is above the set threshold. Since these two parameters have a score of 0.67[1], they will be considered a match at this threshold. If the threshold is set to 0.7, these parameters will not be considered a match. Thus by varying the semantic threshold, we obtain different match pairs and thereby different precision and recall measures. The intuition is that as the service matching score threshold increases the number of matches decreases. This, in turn, increases the number of missing matches. Therefore recall ratio decreases. Whereas precision increases because higher thresholds have the effect of lowering incorrect matches. This intuition is confirmed by our experiments (as shown in Figures 2-10). The reason for using this semantic similarity score threshold in our experiments is that we are interested in finding out the optimal threshold at which optimal precision and recall can be obtained for our data sets. Next, we will explain some of the interesting results obtained during the experiments.

In Figures 8 and 9, both the lexical (L) and the dictionary (L+D) approaches have a recall and a precision of 0 at all thresholds. The lexical approach failed because the two interfaces were developed by different users, thus the chances of using exactly the same words to denote identical notions were very small. The dictionary approach failed because in most cases, users define multi-word tags to better explain the concept defined by a tag and a dictionary does not usually contain multiword tags. However, this is not always the case.

Figures 3 and 6 suggest that the lexical and dictionary-based approaches yield very good precision at all thresholds, i.e., the number of incorrect matches found is relatively small compared to the number of correct matches found. Examination of data in service pairs 1 and 2 reveal that there are indeed some parameters that can be matched using lexical and dictionary-based approaches. These are the parameters that matched exactly (or those did not use any abbreviations). For example: 'INDUSTRY_CLASS' in Application A matched with 'Industry_Class' in Application B. These types of matches contributed to good precision. But it is to be noted that in this case, the number of correct matches found is very low compared to the total number of expected matches i.e., the number of missing matches is high. Therefore, the recall is very low (i.e., recall of 0.06 and 0.13 in Figures 2 and 5).

All figures suggest that the most significant increase in performance (both recall and precision) is obtained through tokenization. Furthermore, the performance increases even more by applying word expansion and ontology-based matching techniques.

During our experiments, we noticed that the precision is not always increasing with the service match score threshold (see Figure 6). The reason is that the number of correct matches dropped with the increase of the threshold, while the number of incorrect matches remained constant. This break in monotonic rise of precision can be noticed between thresholds 0.6 and 0.8. The details are shown in table 2.

Threshold	# correct matches	# incorrect matches	precision
0.6	25	8	0.76
0.7	15	2	0.88
0.8	11	2	0.84

Table 2: Table 2 explains the fluctuations in monotonic increase in the precision in the precision curves in figure 5.

Another observation is that L+D+T+X+O approaches have the best performance. In particular, Figure 8 shows the ideal case when recall is 1. In this particular case, the ontology and the ontology-annotations were built together with an expert; thus, the experiment obtained perfect results. Even though this might not be always the case (see Figures 2 and 5), we strongly believe that users can highly improve the quality of the matches if they take the time and effort to build a meaningful ontology.

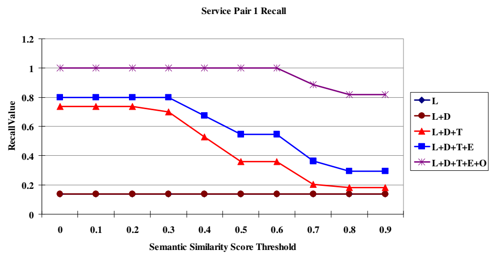

Figure 2: Recall curves for Service Pair 1

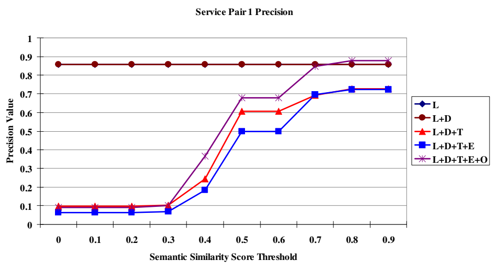

Figure 3: Precision curves for Service Pair 1

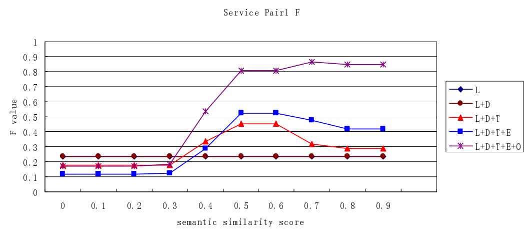

Figure 4: F curves for Service Pair1

[1] Section 3 discussed our score computation scheme that shows how we obtain a score of 0.67 between the parameters 'Country_Country_CD''and 'Country_CD'.

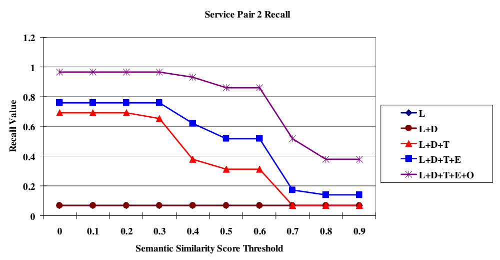

Figure 5: Recall curves for Service Pair 2

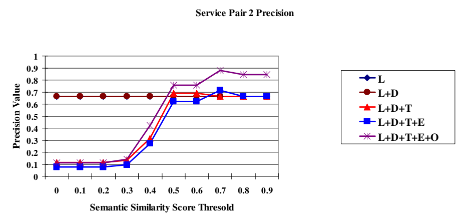

Figure 6: Precision curves for Service Pair 2

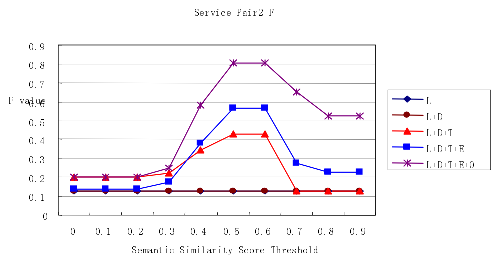

Figure7: F curves for Service Pair 2

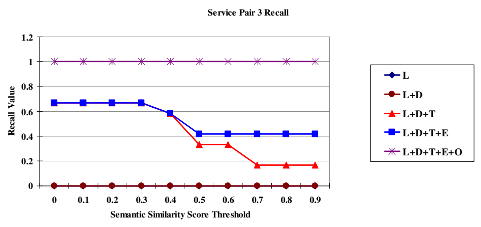

Figure 8: Recall curves for Service Pair 3

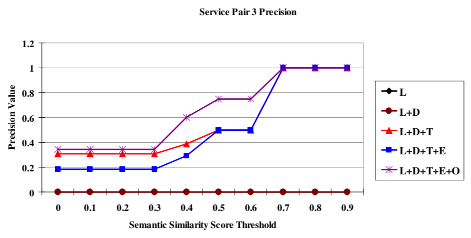

Figure 9: Precision curves for Service Pair 3

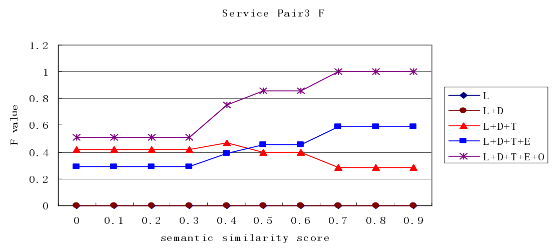

Figure10: F curves for Service Pair3

F-measure is a harmonic mean of precision and recall and in general, is preferred over an arithmetic mean since it better represents the inverse relationship between precision and recall. Since precision and recall typically hold an inverse relationship with one another, an ideal plot of F-measure resembles a bell curve. The objective of matching is to maximize both precision and recall which typically occurs at the point where precision and recall curves intersect. This point is also the point at which an F-measure is highest for those given precision and recall metrics. This point at which an F-measure is maximized gives us an optimal semantic threshold that works for that problem domain. It is to be noted that identifying the optimal threshold is a tough problem in itself and it may vary from problem to problem. Figures 4, 7 and 10 show plots of F-measures plotted against semantic similarity score threshold for the three service pairs under consideration. As expected, figures 4 and 6 show bell-shaped curves for F-measures for most experiments except for the L, and L+D experiments. The F-measures for L and L+D stay constant hovering between 0.1 and 0.3 for both service pair experiments at all semantic thresholds because of the small # of exact matches that could be found with lexical and dictionary based matching which improved the precision significantly while keeping the recall very low (because of the number of missing matches). In the case of service pair 3, F-measures when plotted don't show a drop as expected in a pronounced manner because in this experiment human expert's annotations have helped match all the match pairs thereby making the recall 100% even while the precision was improving steadily as the semantic threshold increased. While achieving 100% recall is not always possible even while keeping the precision high, the F-measure nevertheless shows us that peak for all experiments, on an average, occurs at around a semantic threshold of 06. In all three service pair experiments, maximum F-measure appears to be occurring between the semantic thresholds 0.5-0.7.

At semantic threshold 0.6, on an average, L+D+T+X yields 49% recall and 54% precision. On the other hand, when domain ontologies are added, L+D+T+X+O yields 95% recall and 72% precision. These results are very encouraging.

6 Related Work

The problem of automatically finding semantic relationships between schemas has been addressed by a number of database researchers lately including [Madhavan *et al.*] [Melnik *et al.* 2002] [Rahm *et al.*]. The notion of elemental and structural level schema matching has been present in the METEOR-S project [Patil *et al* 2004], where the engine can perform both element and structure level schema matching for Web services. The element level matching is based on a combination of Porter Stemmer for root word selection, WordNet dictionary for synonyms, abbreviation dictionary to handle acronyms and NGram algorithm for linguistic similarity of the names of the two concepts. The schema matching examines the structural similarity between two concepts. Both element match score and schema match score are then used to determine the final match score.

Recently, clustering and classification techniques from machine learning are being applied to the problem of Web service matching and classification at either the whole Web service level [Hess and Kushmerick 2003] or at the operation level [Dong et al 2004]. In [Hess and Kushmerick 2003] for example, all terms from portTypes, operations and messages in a WSDL document are treated as a bag of words and multi-dimensional vectors created from these bag of words are used for Web service classification. Although this type of classification retrieves matches with higher precision that full-text indexed search, the overall matches produced, however, do not guarantee a match of operations to operations, messages to messages, etc. The paper by Madhavan et al [Madhavan *et al.* 2001] addresses this aspect by focusing on matching of operations in Web services. Specifically, it clusters parameters present in input and outputs of operations (i.e. messages) based on their co-occurrence into parameter concept clusters. This information is exploited at the parameter, the inputs and output, and operation levels to determine similarity of operations in Web services. All these approaches use simplistic Web services available on the Web. Since there are only a handful of parameters per operation, and only partial matches of parameters can be expected in realistic Web services. Thus it is not clear how this method scales to industrial strength Web services that have few parameters each modeled though by complete XSD schemas wherein lies the true information for matching parameters. Also, this work does not consider ontological reasoning and human annotations. Our system [Syeda-Mahmood *et al.*, 2005], which is used in this case study, is among the first of the systems to the best of our knowledge that combines information retrieval and ontological reasoning with an indexing method to efficiently scale to deal with matching large industrial strength Web services. By combining multiple approaches (information retrieval and ontology matching), we show that better relevancy results can be obtained for service matches, than could be obtained using any one cue alone.

Related work in the area of application of (semi) automatic Web service matching for industry problems can be classified into business-2-consumer and business-2-business categories. From our experience matching the interfaces of business-2-business applications is very different and much harder than matching the interfaces of services such as those of travel agency services [Zaremba *et al* 2006] or office utility services [Rcal 2002]. This is so because in business-2-business setting companies tend to use many acronyms and abbreviations since most of the parameters typically originate in databases. Also, in legacy application integration, the structure of the applications that are being integrated could be significantly different from one another. These characteristics pose interesting challenges that are typically absent in business-2-consumer domains. Some products such as IBM's Rational Data Architect [RDA 2006] and possibly from other vendors as well provide schema matching tools that are grounded in information retrieval techniques. However, such approaches do not allow for ontological reasoning and human annotations. To the best extent of our knowledge, our case study is the first of its kind that applies a combination of information retrieval and ontological reasoning techniques to real-world legacy application integration problems.

7 Conclusions

In this paper we have presented the results of a case study on the application of our semantic Web service matching system to a real-world legacy application integration problem. Experimental results on our initial data sets indicate that the system on its own (with minimal help from an IT consultant in terms of abbreviation expansions) is able to achieve close to an average of 50% recall and 54% precision measures in matching the parameters that define the interfaces of business applications. Additional performance enhancements of up to 95% recall and 72% precision measures were achieved with the usage of domain ontologies created by an expert. Such an initial effort on creating abbreviation expansions and domain ontologies etc. would be justifiable if multiple transactions occur between any given applications that need to be integrated. The initial effort pays-off in the long run because once provided, this information can be reused in matching the multiple interface pairs of the applications. These initial results are encouraging. Based on this, we believe that an IT consultant's productivity can increase significantly when semantics-based intelligent tools are used to integrate seemingly disparate but semantically similar applications. These productivity enhancements can be quickly translated into savings in time and money in business process integration projects. Currently, we are conducting experiments on large data sets to verify our results. As part of our future work we are also developing tools that can create domain ontologies (semi) automatically using machine learning and natural language processing techniques. This can further reduce the burden on the consultant in providing the initial domain ontologies.

7 References

[Akkiraju *et al*. 2005] R. Akkiraju, J. Farrell, J. Miller, M. Nagarajan, A Sheth, K. Verma. "Web Services Semantics - WSDL-S. A W3C submission". 2005. http://www.w3.org/Submission/WSDL-S/

[Christenson *et al* 2001] E. Christenson, F. Curbera, G. Meredith, and S. Weerawarana. "Web services Description Language" (WSDL) 2001. www.w3.org/TR/wsdl

[Dong *et al*., 2004] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity search for web services. In Proceedings of VLDB, 2004.

[Hess and Kushmerick 2003] A. Hess and N. Kushmerick, "Learning to attach metadata to Web services," in Proc. Intl. Semantic web conference, 2003.

[Madhavan *et al* 2001] J. Madhavan et al, "Generic schema matching with cupid" in Proc. VLDB 2001.

[Melnik *et al*. 2002] S. Melnik, H. Garcia-Molina and E. Rahm, "Similarity flooding: A versatile graph matching algorithm and its application to schema matching," in Proc. ICDE, 2002.

[Miller 1983] G. Miller. Wordnet: A lexical database for the english language. In Comm. ACM, 1983.

[Patil *et al* 2004] A.Patil, S. Oundhakar, A. Sheth, K. Verma. "Meteor-s Web service annotation framework", in Proc. WWW conference, pp. 553-562, 2004.

[Payne *et al* 2002] T. Payne, R. Singh, K. Sycara, "Rcal: A Case Study on Semantic Web Agents", in the Proc of AAMAS'02 July 15-19, Bologna, Italy 2002.

[Rahm *et al*. 2001] E. Rahm and P. Bernstein, "A survey of approaches to automatic schema matching," in the VLDB Journal, vol.10, pp.334-350 2001.

[RDA 2006] IBM Rational Data Architect http://www-306.ibm.com/software/data/integration/rda/

[Sycara *et al*., 1999] K. Sycara et al. "Dynamic service match making among agents in open information environments," in Jl. ACM SIGMOD Record, 1999.

[Syeda-Mahmood *et al*., 2005] T. Syeda-Mahmood, G. Shah, R. Akkiraju, A. Ivan, and R. Goodwin. Searching service repositories by combining semantic and ontological matching. In Proceedings of 2005 IEEE International Conference on Web Services, 2005.

[Zaremba *et al* 2006] M. Zaremba, M. Moran, T. Haselwanter Applying Semantic Web services to Virtual Travel Agency Case Study. Third European Semantic Web Conference, Budava June 2006. http://www.eswc2006.org/poster-papers/FP04-Zaremba.pdf