

A Web Service Recommender System Using Enhanced Syntactical Matching

M. Brian Blake and Michael F. Nowlan

Department of Computer Science

Georgetown University

Washington, DC

{mb7, mfn3} @georgetown.edu

ABSTRACT

Service-oriented computing (SOC) enables organizations and individual users to discover openly-accessible capabilities realized as services over the Internet. However, service registries can potentially be very large preventing organizations from discovering services in real-time. In fact, consumers may not be aware of the services that can be of most benefit to them. In our work, we introduce a web service recommender system that proactively discovers and manages web services. This paper focuses on the underlying search and ranking algorithms that enable the recommendations. As an innovation, we have analyzed real, fully-operational web services currently available on the Internet and, as a result, have discovered insights into how real web service messages are defined. Using these general naming tendencies coupled with enhanced syntactical methods, we are able to aggregate services by their messages and accurately suggest candidate services to users as a part of daily routines.

Categories and Subject Descriptors

D.2.11 [Software]: Software Engineering: Software Architectures – domain-specific architectures.

General Terms

Design, Experimentation, Standardization, Languages

Keywords

Web Services, Discovery, Recommendation

1. INTRODUCTION

Web services play a central role in service-oriented architectures (SOA) and the SOC paradigm [2][18]. Web services can be defined as networked capabilities with openly accessible interfaces such that they can be discovered and executed by other machines, in real-time. Consumers using an individual web service or chains of services (e.g. as a result of web service composition [10]) may benefit from the knowledge of other, more qualified services that they may later substitute into their processes [3]. Alternatively, there may be other related services that may facilitate their tasks. As web services technologies increase in popularity, it is likely that the pool of available services on the Internet will become increasingly large. The increase of available services may present a significant problem if consumers want to find relevant services.

While accurate search and discovery (i.e. with a high level of confidence) require semantic approaches with technologies such as RDF, OWL-S, and WSDL-S [7][11][17][23], currently these technologies are not widely used in practice. In addition, searching repositories in real time using semantic approaches becomes increasingly more time-consuming as repository sizes increase and the number of reference ontologies or *upper ontologies* increases and becomes more complex.

In this work, we attempt to recommend candidate services to consumers proactively within their daily routine. Considering services currently accessible in practice, most do not contain semantic representations [9][16]. In fact, semantic processing would be difficult with respect to performance if the goal is to proactively deliver recommendations to a consumer within their daily routine. In addition, in the case of recommending services, human consumers can ultimately decide if the candidate services meet their needs. Of course, for automated substitution of services, semantics would be required. In this work, we propose and evaluate a syntactic approach to recommend relevant services to consumers. Syntactic approaches may be more practical in a recommendation scenario in the near-term, since current repositories do not contain semantics. Furthermore proactive recommendation scenarios have stringent performance requirements. In Figure 1, we illustrate a proactive recommendation scenario. As a first step, files and system messages are monitored during operational sessions. Secondly, individual strings are extracted from these files. The third step, and the focus of our work, is using the extracted strings to find relevant services. Finally, based on the extent to which a service is relevant to a captured file, the system suggests relevant web services.

In developing syntactical approaches for web services recommendation, we addressed several general problems:

1. *When are web services relevant to a consumer's operational routines?*
2. *How do you determine similarity between the data that a consumer is viewing/processing and the inputs, outputs, and service name of a web service?*
3. *How many matches between strings in consumer data and those within a web service are required for recommendation? Is this threshold domain-dependent?*

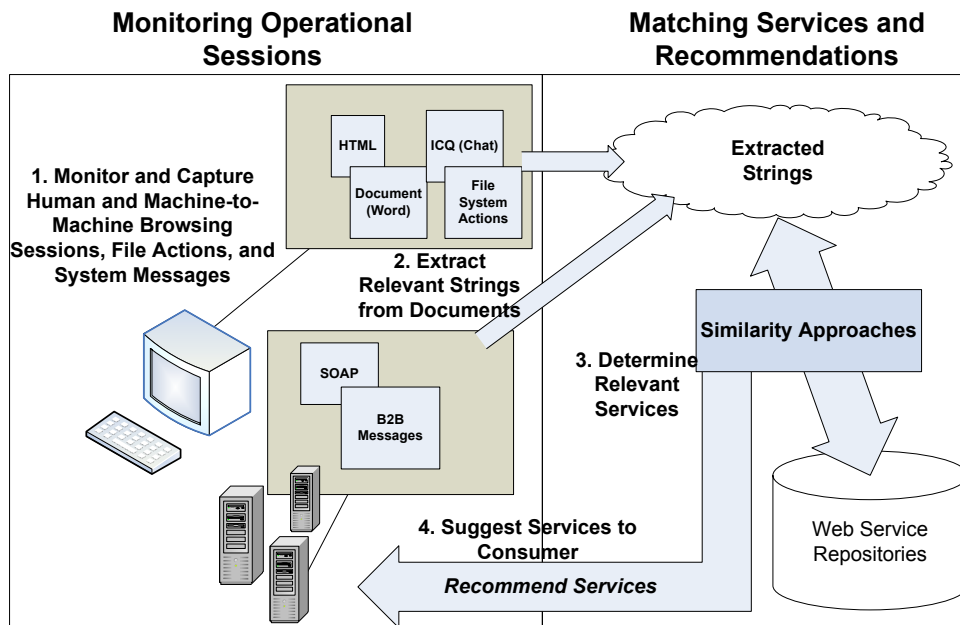


Figure 1. The Web Service Recommendation Scenario.

This paper proceeds in the next section with a discussion of related work. In Section 3, we discuss the trends in naming web services that can be used to enhance syntactical matching approaches when determining relevant service recommendations. In Section 4, we introduce a similarity approach that uses developers' naming tendencies. In Section 5, we show how a similarity score can be reached based on the number of matches between user data and attributes of a candidate web services. In the final section, we evaluate through experimentation how accurately our similarity approach recommends services relevant to a user's activity.

2. RELATED WORK

Techniques for the discovery and composition of web services are the target of many related projects for service-oriented computing. Srivastava and Koehler [10] and Rao [20] detailed the progress that has been made in the field of service composition and detailed the two competing approaches, *semantic* and *syntactic* techniques. Semantic approaches generally support the integration of web services by exploiting the semantic description of their functionality using ontological approaches [1] [22] [23] [12]. Conversely, syntactic projects tend to concentrate on string manipulation and thesauri approaches to correlate services [5][6]. Our approach is not related to the semantics approaches to discovery/composition but more closely related to the syntactic projects. Rocco [21] uses rigorous string manipulation software to help equate web services messages while Pu [19] uses an eXtensible Markup Language (XML) type-oriented rule-based approach.

Dong, Halevy et. al. [8] introduce a web service search engine named Woogle. The approach in this work uses syntactical methods associated with clustering algorithms to generate queries in a web services data repository. This approach is effective and performs favorable provided a user's query. Our approach differs in the fact that we try to scan consumer data to proactively

generate a query. Our approach also attempts to understand the how consumers name their services from the bottom-up. By using the tendencies of consumers for searching for web services, our approach may be able utilize human tendencies to create shortcuts in operations.

The innovation in our work differs from related projects in that we attempt to capture the tendencies of the software designers/developers that create the web services. By using these tendencies, we create lightweight approaches that combine the nature of message naming (as selected by software designers in operational environments) with standard string manipulation approaches. Unlike projects that evaluate their approaches through performance, our work uses perhaps the largest repository of functional web services to qualitatively determine the effectiveness of our approach to correlate web services messages on the open Internet. This paper expands prior work [16] with new letter pairing algorithms and new approaches to determining fidelity thresholds that can be determined in real-time.

3. UNDERSTANDING CONSUMER MESSAGE NAMING TENDENCIES

In developing an algorithm that is capable of recommending web services based user data, the first step was to understand the tendencies of the software developers that name the web services. This approach does not consider semantics in evaluating services, but the notion is that the naming trends exercised by the developers that build web services may be useful as a substitute. In addition, services performing similar functions are more likely to contain the same or similar part names (the stratification of the web service repository is discussed in later sections). The general notion of this approach is that the strings in a user file need to be examined to find out how similar they are to the inputs/outputs of a web service message and/or the operation name. The naming tendencies can be used to link strings in the user's files to the strings used in the definition of web services.

In order to gain an understanding of these tendencies, we manually downloaded and verified the functionality of as many web services as possible. Web services came from a number of web services repositories (i.e. Woogole, XMLMethods, Salcentral, BindingPoint, and WebServiceList) [4][24][25], Amazon Corporation [1], and random Internet searches. Two students spent approximately 40 hours downloading and verifying service functionality using Mindreef’s Soapscope application [14]. We believe our repository is one of the biggest available for benchmarking since we were able to determine repetition among our source repositories. From this effort we collected 590 services descriptions with over 5200 operations and just under 20,000 web services messages or *parts*. From this point on, we will refer to web services messages as *part names*.

In this work, we took a bottom-up an approach to determine naming tendencies. We analyzed services to characterize differences in messages that are essentially the same. We determined 4 significant tendencies from this analysis.

Tendency 1 (Subsumption Relationships). There is a strong tendency for web service developers to use part names based on common names. When using common names, similar messages tend have strong subsumption relationships. For example, we found equivalent services where $name = lname$, $name = first_name$, and $name = user_name$.

Tendency 2 (Common Subsets). Similar to subsumption relationships, some web service part names are related by having common subsets. For example, we found equivalent part names $first_name = user_name$.

Tendency 3 (Abbreviations in Naming). Another strong tendency was for common names to be shortened into abbreviations. For example, $building = bldg$ or $country = cntry$.

Tendency 4 (Size constraints). We found that strings that were shorter than 3 characters and longer than 15 characters are ineffective for matching part names.

4. EXPLOITING NAMING TENDENCIES TO FIND RELEVANT MESSAGES

Based on the previously defined tendencies, we developed an algorithm that exploits those tendencies to discover when the web service part names are equivalent. The following sections describe the individual syntactical approaches used to develop the recommendation algorithm.

4.1 Equality, Subsumption Relationships and Size Constraints

Probably the most straightforward comparison for two part names is to check to see if they are equal. There were a number of occurrences when developers of different services did decide to use the same name for a similar type of message. However, a more likely occurrence is the many cases where one developer used a two-word description of an idea, while another used one word to mean the same idea. Approaches to exploit tendencies 1 and 4 are equally straightforward, programmatically. Web service part names are evaluated to see if the first string is a subset of the second string or second string is a subset of the first string. In addition, this approach disregards part names that are smaller than 3 characters or larger than 15 characters.

4.2 Using Levenshtein Distance, Letter Paring for Tendencies 2 and 3

Several syntactic approaches were used to exploit Tendency 2 and 3 when matching services. The Levenshtein distance (LD) (also called the *edit distance*) is a measure of similarity between two strings [13][15]. The LD is the smallest number of deletions, insertions, or substitutions required to transform a source string, s , into a target string, t . The greater the LD, the more different the strings are. For example:

- If $s = \text{“test”}$ and $t = \text{“test”}$, then $LD(s,t) = 0$, because no transformations are needed. The strings are already identical.
- If $s = \text{“test”}$ and $t = \text{“tent”}$, then $LD(s,t) = 1$, because one substitution (change "s" to "n") is required to transform s into t .

In our work, we adapted implementations of the LD algorithm from several sources [10][14]. The LD algorithm is effective when evaluating abbreviations with full strings. In addition, LD is also effective for similar strings that are changed to create uniqueness. There were a number of occurrences where zeros are substituted for the letter O . In the later sections, we will describe how we determine how many transformations are required as a threshold. Although LD is effective for similar strings, it is not effective for strings that have similar subsets that do not have true subsumption relations as in Tendency 1. For example, $last_name$ and $surname$ are equivalent but neither is a subset of the other. To account for instances of this nature, we employed the use of Letter Pairing. The Letter Pairing (LP) approach is an algorithm that can be used to match strings that have common subsets. Using the LP algorithm, two string are separated into *letter pairs*. STR1 would be separated in ST , TR , and $R1$, and STR2 would be separated into ST , TR , and $R2$. Two multiplied by the number of identical pairs between the two strings, is divided by the total number of pairs. This calculation results into a percentage. In this case the LP similarity would be $4/6$, because there are two pairs shared by both strings and double two is four. The total number of pairs is 6, three from each string, and so the percentage is 66.7%. An innovation of our work is adjusting the threshold for this percentage depending on the category of the web services.

4.3 The Combined Algorithm: Tendency-Based Syntactic Matching - Levenshtein Pairing (TSM-LP)

We introduce a matching algorithm called *Tendency-Based Syntactic Matching-Levenshtein Distance and Letter Pairing (TSM-LP)*. This algorithm exploits the four previously mentioned tendencies using LD and LP. The TSM-LP algorithm is defined as a combination of all of the syntactical approaches defined in Sections 4.1 and 4.2. The core of TSM-LP is the Tendency-Based Thresholds, F_{T1} and F_{T2} , that we use to govern the LD algorithm, L_D , and LP algorithm, L_P , when comparing a two strings, S_i and S_j . TSM-LP is governed by two thresholds that are closely tied to the uniqueness or *Sensitivity* of the category of web services. The TSM-LP algorithm is defined in Figure 2. In summary, if the LD and LP are within the Tendency-Based Thresholds or if either of the strings are a subset of the other and the strings are both greater than 3 and less than 15, then the strings are considered similar.

$TSM-LP(S_i, S_j)$:	TSM-L Function
$L_D(S_i, S_j)$:	Levenshtein Distance function
$F_{T1}(S_i)$:	Tendency-Based Threshold
$F_{T2}(S_i)$:	Tendency-Based Threshold for Letter Pairing
S_i, S_j :	Two strings for comparison
$Length()$:	String length functions
C_S	Web Service Category (e.g. Business)
$F_{T1}(S_i)$	
$temp = [(Length(S_i) * 2) / 3] - 2$	
return $temp$	
$F_{T2}(S_i)$	
$temp = Sensitivity(C_S)$	
return $temp$	
$TSM-LP(S_i, S_j)$	
if $(L_D(S_i, S_j) \leq F_{T1}(S_i))$ or	
$(L_P(S_i, S_j) \geq F_{T2}(S_i))$ or	
$(S_i \subseteq S_j \text{ or } S_j \subseteq S_i)$ and	
$(S_i > 3 \text{ and } S_j > 3)$ and	
$(S_i < 3 \text{ and } S_j < 15)$	
return $TRUE$	
else	
return $FALSE$	

Figure 2. The TSM-LP Algorithm.

5. CALCULATING A RECOMMENDATION SCORE USING TSM-LP

A candidate web service is relevant to a consumer's file or data messages when a sufficient number of matches are made between the strings that compose the consumer's file and the strings composing the web service part names and operation names. Formally, each operation, O_w , of a WSDL file is evaluated separately. The operation consists of an operation name N_O , the name of the WSDL file, N_W , and the set of part names $\{P_1, P_2, \dots, P_N\}$. The consumer file, F_C , consists of the set of relevant strings, $\{S_1, S_2, \dots, S_N\}$, that are contained within it. There are two major components of our recommendation approach.

1. Determining the similarity thresholds based on the nature of the repository
2. Creating a recommendation score for a web service based on a specific consumer-based file or operation.

5.1 Similarity Thresholds for TSM-LP

TSM-LP determines syntactically if two strings are equivalent. As such, TSM-LP is the centerpiece of our approach. Determining when messages descriptions are similar to a consumer's files establishes relevancy. TSM-LP is not a static

evaluation approach. TSM-LP is customized quantitatively based on the nature of the web services repository that is being used for recommendation. The major attribute of the repository that determines the sensitivity of the TSM-LP algorithm is the *uniqueness* of the messages. After categorizing all the web services into specific areas (Calendar, Business and Econ, Graphics, News, Communication, Technology, Military, Conversion, and Entertainment), a function is executed that determines how *self-similar* the repository (i.e. category) is to itself. In determining self-similarity, we determine the percentage of total number of similar part names in a particular category to the total number part names in repository. Figure 3 shows the categories (we created similar categories to [8]) and the self-similarity percentage.

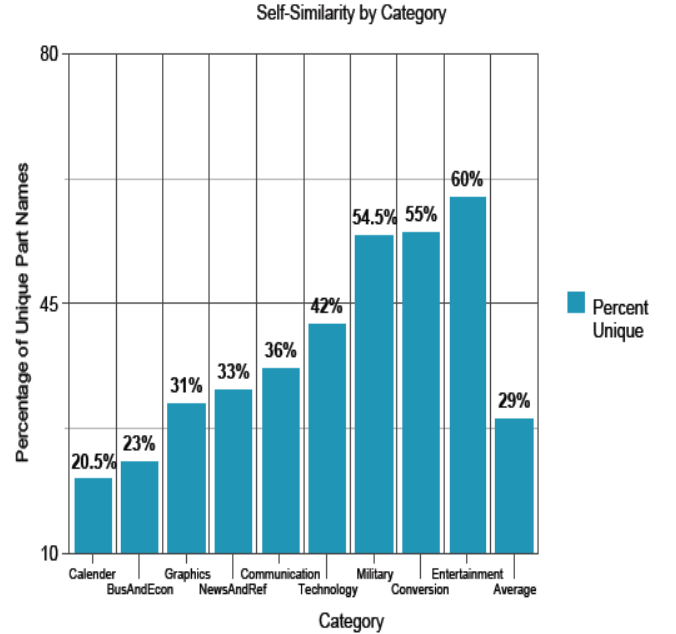


Figure 3. Service Self-Similarity by Category.

The self-similarity percentage is used to determine ranges with respect to the TSM-LP sensitivity. In previous work [16], we found that the average sensitivity for LD could be defined by a specific calculation. In this work, we have refined that sensitivity such that it can be customized based on the consumer's operational domain. This means that while a user works in an area where Business and Economy web services may be beneficial, our recommending system would take into account the low uniqueness, or high self-similarity, in the naming tendencies of the Business and Economy services as a whole when determining which services to suggest. A summary of self-similarity percentages, sensitivity ranges, and corresponding LD and LP thresholds are shown in Table 1. The reader should note that a category with a self-similarity greater than 75% would be too unique to make any reasonable matches. Likewise, a homogeneous repository that is too similar (i.e. under 12.5%) has mostly all the same message names and likely all the exact same services. Using Table 1 and Figure 3 reveals how TSM-LP is customized based on the specific category.

Table 1. Categorical Sensitivity Rankings.

Self-Similarity Percentage	TSM-LP Sensitivity	LD Threshold	LP Threshold
12.5 - 25%	High	$\lfloor \frac{Length(S_i) * 2}{3} \rfloor - 3$	55.0%
25 - 50%	Medium	$\lfloor \frac{Length(S_i) * 2}{3} \rfloor - 2$	47.5%
50 - 75%	Low	$\lfloor \frac{Length(S_i) * 2}{3} \rfloor - 1$	40.0%

5.2 Creating the Recommendation Score

Determining a recommendation score is a straightforward process. The recommendation score is the number of times individually the set of strings within a consumer’s data file is relevant to the web service description name, operation name, and the set of part names for a specific web service. Formally, the recommendation score, R_S , is defined as:

$$R_S = (TSMPLP(S, N_w) + TSMPLP(S, N_o) + TSMPLP(S, P))$$

where TSM-LP is the matching algorithm. In early experimentation, we have arbitrarily limited the recommendations to the most highly scored services to determine accuracy. In future work, we intend to use the characteristics of the domain (similar to determining TSM-LP sensitivity) to determine how many services to recommend (i.e. a recommendation score threshold).

6. EVALUATION AND DISCUSSION

In evaluating our recommendation approaches, we were interested in two measures. The first measure was to determine how accurate TSM-LP is for determining when messages, by their part names, of independent web services are relevant. We define relevancy loosely (i.e. through manual visual inspection, we examine the two part names to see if one part name could be used to supply data to the second part name.).

6.1 Impact of the Tendencies

After implementing these various syntactic measures we needed to examine the effectiveness of our approach in declaring strings similar. Naturally, equivalent strings were declared to be similar, but more important to us were the cases when strings were not equivalent syntactically, but exhibited some similarity. In the repository of services, there were 89,503,269 instances when two strings were tested for similarity. In 1,054,137 (about 1%) of those comparisons, one or more of the “similarity assessments” deemed the strings similar. The rest of the information regarding similarity is represented in Figure 4 according to the number of times each similarity check found similarity between strings. An interesting fact is that the total number of similar matches found (1,054,137) is only about 25% less than the sum of the similar matches of each method (1,322,988). More specifically, there is a relatively small amount of overlap. This result suggests that developers describe services in slightly different ways, and that more than one tendency exploitation is needed to be fully comprehensive. These results suggest that each of the three tendencies provide their own unique contribution to the determination of similarity.

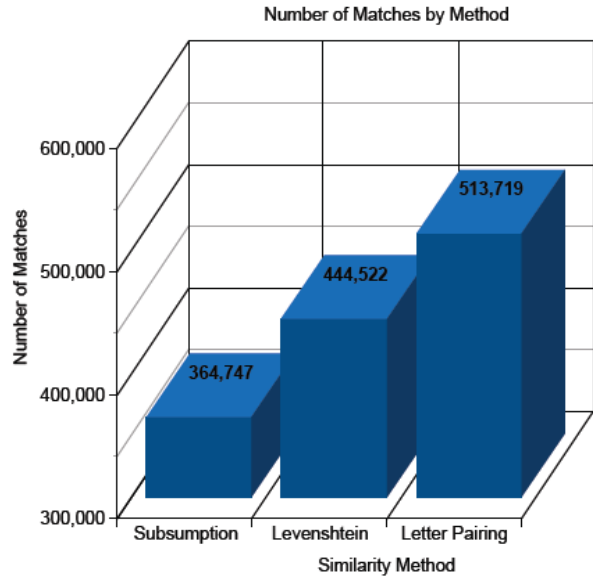


Figure 4. Impact of Each Tendency in Determining Similarity

The initial experiment only determined what strings were considered similar by TSM-LP, but not how accurate the approach is. To examine the accuracy of all three tendency checks, the only accurate method was to perform a manual verification. However, a manual verification of accuracy for over 5,000 part names is not feasible. As such, we looked at the top 50 most commonly appearing part names in the repository. Figure 5 shows the percentage of valid matches returned by each method in the top 50 most commonly used part names (and implicitly shows the false positive percentage). Although the “Letter Pairing” similarity method had the lowest false positive rate (highest accuracy), it is important to note that all three methods are needed for comprehensive similarity matching because each method specializes in determining similarity for different developer naming tendencies.

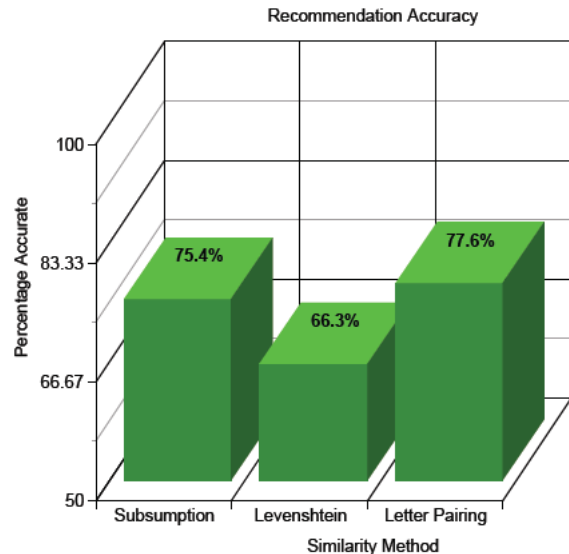


Figure 5. Positive Matches by Tendency Check

6.2 Comparing TSM-LP to Other Methods

Figure 6 shows the three different matching methods and the breakdown of the similarity matching between the user’s file and the top 100 most strongly recommended operations. As the chart shows, all three approaches return the same number of exact *hits* (about 5,700) within the top 100 most common part names. A “hit” is declared when the strings are exactly equal. This is and should be the same for each case, because basic string equality is the foundation of each method. A “match” is declared when two strings are either exactly equal or deemed similar by the similarity method. All hits are matches, but not all matches are hits. Figure 6 shows the breakdown by method for the number of hits, exact equivalencies, and the number of similar matches. The first method shows that when using a basic equality test, 5700 hits are returned, as expected. It also shows that for this method there are no similar matches found because by the nature of the method, only exact hits can be returned. On the other hand, TSM-LP returns about 17,000 similar matches beyond the original 5,700 within the top 100 most common part names. The last measure is using LD with a static number of transformations (i.e. 5) as a threshold, which produces 97,000 relevancy matches. The authors also tested a static transformation allowance of 11 and a transformation allowance equal to the string length, but these methods performed worse than the threshold of 5 and thus are not shown here. Figure 7 shows that TSM-LP is 78% accurate with regards to relevancy. Considering there were about 15% of the part names with ambiguous naming conventions, this percentage is a positive result.

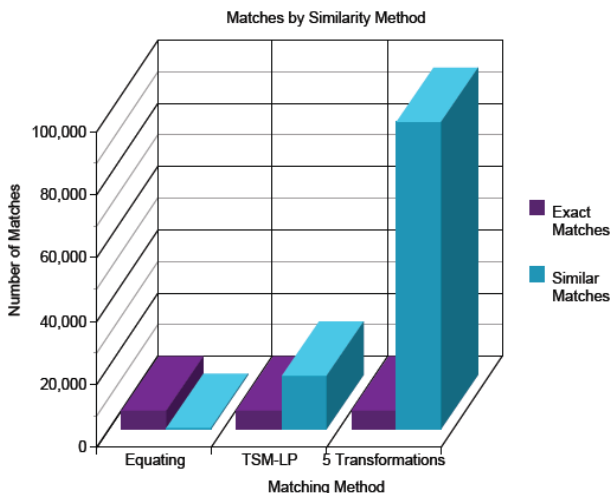


Figure 6. Exact and Similar Matches by Method.

TSM-LP is a similarity matching method capable of capturing the matches beyond the exact string equivalencies, but not to the extent that it recommends completely unrelated web services. Figure 7 examines only those matches returned by the methods that are NOT hits (i.e. strings that are not identical but found to be similar by the matching method). It is for this reason that equating (i.e. exact equivalence) cannot return any such matches.

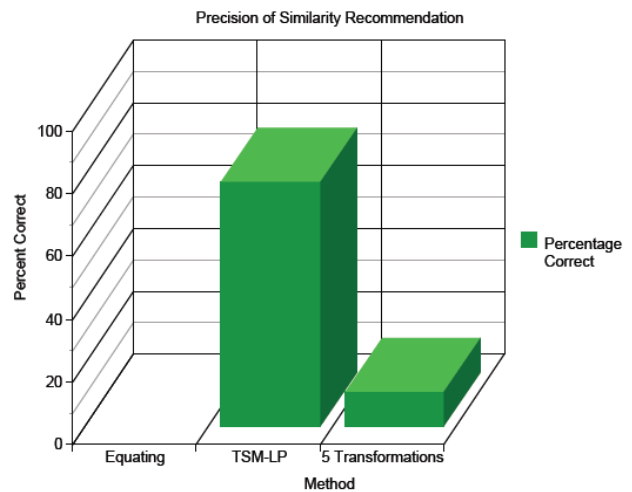


Figure 7. Precision of Similarity Recommendations.

A second important evaluation is to determine how accurately the algorithm recommends web services. In a second experiment, three HTML files were randomly pulled from the Internet. These HTML pages represented the subject matter of *Books*, *Sports*, and *Finance*. The relevant strings from each of the files were extracted and submitted to the recommendation algorithm. We evaluated the top 10% of the recommendations to determine how many of the services belong to the relevant categories, as shown in Figure 3. Figures 6, 7, and 8 show the results of this experiment. In each case, services in relevant categories were the prominent recommendations. The strongest result was from the Finance file where more than 66% of the services recommended came from the Business and Economics category. An interesting result was that the Books file yielded most of the results from the Calendar category. Finally, the Sports file has an even distribution between the Calendar, Business and Economics, and News categories, and percentages are given.

6.3 Actual Operation Recommendations

We have on-going experimentation to determine the appropriate recommendation score and the most effective number of web service operations that should be recommended for any given case. However, in order to truly see the effectiveness of TSM-LP for service recommendation, we examined the first three suggested web service operations after sending in five different files pulled randomly from a user’s desktop. Although in real-practice it is possible that only the highest scoring web service operation will be recommended, we displayed the results for the highest three operations to show the comprehensiveness of our approach. Table 2 shows the suggested operations for various files. Note that a higher Relevancy Score for an operation means that there is a greater probability of similarity between that operation’s definition and the user’s file.

Table 2. Actual operation suggestions made using TSM-LP.

Type of File	Operation Name	Relevancy Score
Itinerary generated from Travel website	GetStations	2350
	IsValidExchange	2350
	IsExchangeOpen	2200
Currency conversions webpage	GetSearchTerms	1050
	NumberToDollars	1050
	Search	1000
Random book search from online bookseller	ListBooks	1600
	BooksInfo	1400
	WishlistSearchRequest	1250
Finance homepage on Yahoo.com	IsValidExchange	1200
	GetCurrentMortgageIndex	1150
	IsExchangeOpen	1100
Sports homepage on msn.com	GetSportNews	1850
	WorldCupFootball	1650
	GetBriefings	1200

7. CONCLUSION

In this paper, we introduce a syntactical approach for aggregating and analyzing web service messages to facilitate the recommendation of capabilities to consumers. An innovation of this approach is the alignment of the algorithm with real human tendencies when naming web service messages. We have determined several approaches for customizing recommendations based on the distinctiveness of the consumer’s domain and the characteristics of the web services repository being used for search. In future work, we intend to extend the ability to characterize recommendations based on real-time evaluation of the environment. In addition, we plan to incorporate these algorithms within an intelligent agent architecture that can be used to federate search among multiple distributed web services registries.

8. ACKNOWLEDGEMENTS

The service repository and certain parts of the service discovery software used in this work were partially funded by the National Science Foundation under award number 0548514. We also acknowledge the early conversations and contributions of Daniel R. Kahan.

Top 10% of Suggested Operations for Books File

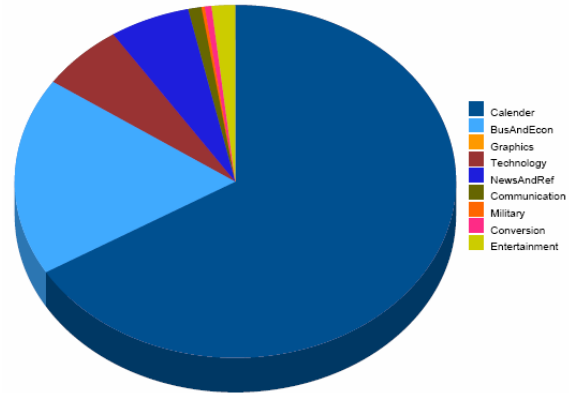


Figure 8. Distribution of Recommendations on Books File.

Top 10% of Suggested Operations for Finance HTML File



Figure 9. Distribution of Recommendations on Finance File.

Top 10% of Suggested Operations for Sports HTML File

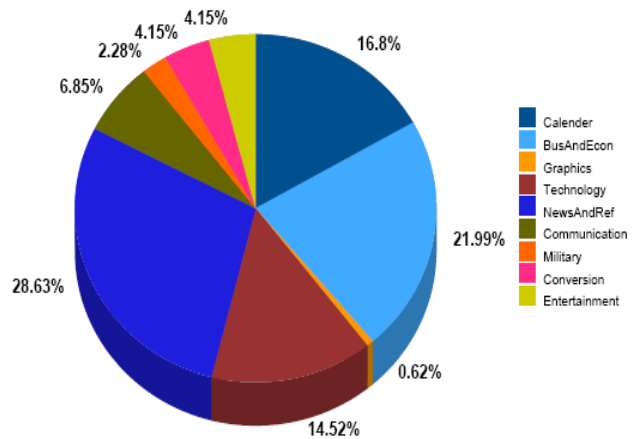


Figure 10. Distribution of Recommendations on Sports File.

9. REFERENCES

- [1] Amazon Web Services (2007): www.amazon.com/gp/aws/landing.html
- [2] Benatallah, B., Dumas, M., and Sheng, O.Z. Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. *Distributed and Parallel Databases* 15(1):5-37, January 2005. Kluwer Academic Publishers
- [3] Blake, M.B., Kahan, D., Fado, D.H., and Mack, G.A. "SAGE: Software Agent-Based Groupware Using E-Services" *ACM GROUP 2005*, Sanibel Florida, November 2005
- [4] Blake, M.B., Tsui, K.C., and Wombacher, A. "The EEE-05 Challenge: A New Web Service Discovery and Composition Competition", *Proceedings of the IEEE International Conference on E-Technology, E-Commerce, and E-Services*, Hong Kong, March 2005
- [5] Bosca, A., Ferrato, A., Corno, D., Congui, I., and Valetto, G. "Composing Web Services on the Basis of Natural Language Requests", *Proceedings of the 3rd IEEE International Conference on Web Services (ICWS 2005)*, pp 817-818, Orlando, FL, June 2005
- [6] BPEL4WS (2007): <http://www.ibm.com/developerworks/library/specification/ws-bpel/>
- [7] DAML (2007): <http://www.daml.org>
- [8] Dong, X., Halevy, A.Y., Madhavan, J., and Nemes, E., Zhang, J. *Similarity Search for Web Services*. *VLDB* 2004
- [9] Kahan, D.R., Nowlan, M.F. and Blake, M.B. "Taming Web Services in the Wild", 4th IEEE International Conference on Web Services (ICWS 2006), Chicago, IL, 2006
- [10] Koehler, J. and Srivastava, B., "Web Service Composition: Current Solutions and Open Problems" *Proceedings of the Workshop on Planning for Web Services* in conjunction with ICAPS03, 2003
- [11] McIlraith, S., Son, T. and Zeng, H. Semantic web services. *IEEE Intelligent Systems*, 16(2):46{53), March/April 2001.
- [12] Medjahed, B., Bouguettaya, A. and Elmagarmid, A. K. Composing Web services on the Semantic Web. *The VLDB Journal*, 12(4), November 2003.
- [13] Merriam Park Software (2007): <http://www.merriampark.com/ld.htm>
- [14] Mindreef Soapscope (2007): <http://www.mindreef.com/products/soapscope/index.php>
- [15] NIST Levenshtein Distance (2007): <http://www.nist.gov/dads/HTML/Levenshtein.html>
- [16] Nowlan, M.F., Kahan, D.R., and Blake, M.B. "Using Naming Tendencies to Syntactically Link Web Service Messages", in *Data Engineering Issues in E-Commerce and Services (DEECS)*, Lecture Notes in Computer Science, Vol. 4055, pp. 90-99, Springer-Verlag, June 2006
- [17] OWL-S (2007): <http://www.daml.org/owl-s/>
- [18] Papazoglou, M. "Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of WISE '03*
- [19] Pu, K., Hristidis, V., and Koudas, N. "A Syntactic Rule Based Approach to Web Service Composition", *Proceedings on the International Conference on Data Engineering (ICDE'06)*, Atlanta GA, USA
- [20] Rao, J. and Su, X. "A Survey of Automated Web Service Composition Methods", In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition, SWSWPC 2004*, San Diego, California, USA, July 6th, 2004
- [21] Rocco, D, Caverlee, J., Liu, L. and Critchlow, T. "Domain-specific Web Service Discovery with Service Class Descriptions", *Proceedings of the 3rd IEEE International Web Services* (2006): <http://www.w3.org/2002/ws/desc/>
- [22] Sirin, E., Hendler, J., and Parsia, B. "Semi-automatic composition of Web services using semantic descriptions", In *Proceedings of Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003*, 2002.
- [23] Williams, A.B., Padmanabhan, A., and Blake, M.B. "Experimentation with Local Consensus Ontologies with Implications to Automated Service Composition", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 7, pp 1-13, July 2005
- [24] WS-Challenge (2007): <http://www.ws-challenge.org/>
- [25] XMethods (2007): <http://www.xmethods.com/>