

Using Naming Tendencies to Syntactically Link Web Service Messages

Michael F. Nowlan, Daniel R. Kahan, and M. Brian Blake

Department of Computer Science, Georgetown University,
234 Reiss Science Building, Washington, DC 20057-1232
{mfn3, drk8, mb7}@georgetown.edu

Abstract. Service-oriented computing (SOC) enables organizations and individual users to discover openly-accessible capabilities realized as services over the Internet. An important issue is the management of the messages that flow into and out of these services to ultimately compose higher-level functions. A significant problem occurs when service providers loosely define these messages resulting into many services that in effect cannot be easily integrated. State of the art research explores semantic methods for dealing with this notion of data integration. The assumption is that service providers will define messages in an unpredictable manner. In our work, we investigate the nature of message definitions by analyzing real, fully-operational web services currently available on the Internet (i.e. from the wild). As a result, we have discovered insights into how real web services messages are defined as affected by the tendencies of the web services developers. Using these insights we propose an enhanced syntactical method that can facilitate semantic processing by classifying web services by their message names as a first step.

1 Introduction

Web services are at the core of service-oriented architectures (SOA) and the SOC paradigm [15]. 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. Web service composition involves putting together a *chain* of multiple, related services. If, for example, an intelligent software component or software agent uses a Web service to purchase airline tickets, then the agent likely has enough information to submit queries to other applicable services such as hotels, rental cars, and entertainment events.

One of the aims of the WS-Challenge 2005 [21] was to demonstrate how such chains can be constructed by agents looking only at syntactical elements of Web Service Description Language (WSDL) documents and foregoing the semantic capabilities included with such technologies as DAML [5], OWL-S [8], and BPEL4WS [5]. The work presented in this paper further investigates that aim by analyzing real services in order to understand the actual *tendencies* of service providers when creating message names. By integrating these tendencies captured from real web services with straightforward syntactical approaches, we introduce a quick, *just-in-time* message management approach that effectively facilitates service discovery and composition without the processing overhead and many specifications required by full semantic

techniques. This approach *does not completely replace semantic approaches*, but can help to speed the overall discovery and composition process by filtering and aggregating services by their message names prior to semantic processing.

This paper proceeds in the next section with a discussion of related work in the area of service composition from both a syntactic and semantic point of view. The subsequent section describes the web services repository consisting of open services from the Internet. Section 4 describes the enhanced syntactical approach that we devised to help correlate messages and Section 5 describes an extension of those approaches to service composition. Finally in Section 6, we experiment with discovering services based on user documents to evaluate if our approaches can help suggest relevant services in the context of a user's daily routine.

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 [8] and Rao [17] 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] [19] [19] [10]. Conversely, syntactic projects tend to concentrate on string manipulation and thesauri approaches to correlate services [15]. Our approach is not related to the semantics approaches to discovery/composition but more closely related to the syntactic projects. Rocco [18] uses rigorous string manipulation software to help equate web services messages while Pu [15] uses an eXtensible Markup Language (XML) type-oriented rule-based approach.

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.

3 Message Naming Tendencies

In developing an algorithm that is capable of equating the messages of multiple web services, we took an applied, bottom-up approach. The first step was to understand the tendencies of the software developers that name the web services. In order to get an understanding of these tendencies, we manually downloaded and verified the functionality of as many services as we could identify on the Internet. Web services came from a number of web services repositories [21], Amazon Corporation [1], and random Internet searches. Two students spent approximately 40 hours downloading and verifying service functionality using Mindreef's Soapscope application [12]. From

this effort, we collected 490 WSDL documents. After 40 hours of investigation, it became difficult to find new services. Consequently, we believe that our repository is perhaps one of the biggest of its type (i.e. containing published functional services).

The 490 service descriptions can be decomposed into 12,187 total part names (i.e. part names are the independent parameter strings that comprise a web service message). 2490 part names remain once duplicate entries (i.e. part names occurring multiple times in the same WSDL file) are removed. Finally, once all duplicates are removed, there are 957 unique part names across all services in the repository. Intuitively there are more input part names than output part names. Another interesting result is that only 27% of output part names (after removing duplicates) are unique across services, while the inputs are 43.9% unique (i.e. the outputs are more homogeneous than the inputs). Since the developers are more likely to use the same or similar part names for output messages, this suggests that, when accessing the Universal Description, Discovery and Integration (UDDI) registries, a user should first search by web services outputs then by inputs in order to increase search speed. Table 1 lists the quantitative details of the repository.

Table 1. Detailed Information about the Repository

Statistic Description	Inputs	Outputs	Total
Number of Web Services			490
Gross Number of Part Names			12,187
-----	-----	-----	-----
Number of Part Names (Unique within each WSDL)	1816	674	2,490
Overall Unique Part Names (23 names overlap inputs and outputs)	798	182	957

3.1 Common Message Names

In order to scope our experiment we decided to focus on the top 30 most common part names, because 30 best reflected the names with the number of similar names over 5 occurrences although range was from 536 occurrences to 5 occurrences of a particular part name. We loosely classified the top 30 part names as *ambiguous*, *descriptive*, or *more descriptive*. The part names are listed in Table 2. The authors recognize that these classifications are subjective and use them in an attempt to stratify the results.

Table 2. Top 30 Most Common Part Names

Ambiguous	Descriptive	More Descriptive
Parameters, Body, LicenseInfo, ResponseInfo, SubscriptionInfo, Header, Result, Return, Symbol, Password, Identifier, Fault, IdentifierType, Text, Type	LicenseKey, Username, Name, Height, Width, Style	StartDate, EndDate, Year, AsOfDate, City, Email, Country, State, Month

3.2 Ambiguous and Descriptive Message Names

We have found that the most ambiguous part names in messages are those that use the actual section names of the SOAP message (i.e. *parameters*, *body*, *header*, *fault*, etc.). Even using semantics, these types of strings would be difficult to correlate as they could be related to almost anything. We, however, later found that some of the ambiguous names have specialized meanings within certain web services development environments that depart from the WSDL standards. After analyzing the repository, we found that 15 of the top 30 most common part names are ambiguous strings as shown in Figure 1. There are 1200 occurrences of the most common part names out of 2490 possible part names (unique per service). In other words, almost 50% of the input/output messages use only 3.1% (i.e. 30 of the 957) of the unique names whereas 82% of the common part names are ambiguous. The percentage of common part names by type considering all occurrences is illustrated in Figure 2.

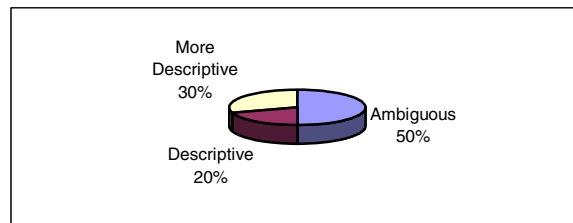


Fig. 1. Percentage of Common Names by Type

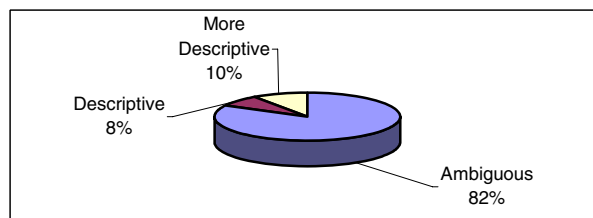


Fig. 2. Percentage of Common Names by Type and by Occurrence (1200 out of 2400)

Ambiguous part names, which are shown to be a common practice among web services developers, are counter-productive to automated approaches to both service discovery and composition. Developers use ambiguous names perhaps based on shortsightedness. Although their services are openly available, the developers still build their services as if the services were only internal to their intranet. Machine-to-machine interactions are either not considered, or perhaps our analysis suggests that the service-oriented paradigms are not thoroughly adopted by those developers that are building the services. However, the immaturity of the tools to support web services development may also play a part. In the remaining experiments, we focus on the more descriptive message names in developing matching algorithms. Experimentation results with ambiguous names are shown for comparison.

4 Correlating Message Names to Enhance Syntactic Processing

The size of our web services repository unfortunately is not adequate for performing significant composition experiments. However, we attempt to provide insight into composition by understanding the similarity of message names across all services. In other words, even if two services do not naturally integrate sequentially, it is important to our experimentation to discover that some subset of their messages have the same meaning. We conducted this experimentation by using a specific text similarity algorithm coupled with the naming tendencies perceived in the repository. Naming tendencies were extracted from the more descriptive part name strings that were occurred most frequently within our repository.

4.1 Tendency-Based Syntactic Matching

We introduce a matching algorithm called Tendency-Based Syntactic Matching-Levenshtein (TSM-L). This algorithm combines several naming tendencies with the Levenshtein distance (LD) (also called the *edit distance*) which is a measure of similarity between two strings. 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 [8][12]. In addition the LD algorithm, the following naming tendencies are considered:

- Part names that are not common but that are similar to the common part names tend to be supersets or subsets of the common part name (for example; endDate is similar Date).
- Exceedingly lengthy strings and strings less than 2 characters are ineffective for message management.
- Setting a threshold for similarity distance is most effective when the LD threshold is not static but some function of the strings that are being compared.

Considering the fact that the LD algorithm is an established algorithm created in 1966 [12], it is not reiterated in this paper. Instead, the TSM-L algorithm is defined as an extension of the LD algorithm. The center of TSM-L is the threshold, or the Fitz Threshold, F_T , that we use to govern the LD algorithm, L_D , when comparing a two strings, S_i and S_j . The TSM-L algorithm is defined in Figure 3. In summary, if the edit distance (LD) is less than or equal to the Fitz Threshold or if either of the strings are a subset of the other and the strings are both greater than 1 and less than 15, then the strings are considered similar.

$TSM-L(S_i, S_j)$:	TSM-L Function
$L_D(S_i, S_j)$:	Levenshtein Distance function
$F_T(S_i)$:	Fitz Threshold Parameter Function
S_i, S_j :	two strings for comparison
$Length()$:	string length functions

$F_T(S_i)$ $temp = [(Length(S_i) * 2) / 3] - 2$ return $temp$
$TSM-L(S_i, S_j)$ if $(L_D(S_i, S_j) \leq F_T(S_i))$ or $(S_i \subseteq S_j$ or $S_j \subseteq S_i)$ and $(S_i > 1$ and $S_j > 1)$ return $TRUE$ else return $FALSE$

Fig. 3. The TSM-L Algorithm

4.2 Repository Similarity Using TSM-L

As a first evaluation of the TSM-L algorithm we evaluated the similarity of the 30 most common part names in our repository. The results of this experiment are illustrated in Figure 4. The top 15 most common message names are ordered by the number of times they occur in the repository. Therefore, the chart shows that the number of similar matches is independent of the number of appearances of the actual value. For example, “parameters,” which appeared 536 times, is the first data point on the *Ambiguous* line and it only has 1 similar match, according to the TSM-L algorithm. Whereas, the 12th highest total for Descriptive parameter names, *State*, only had 9 occurrences in the repository, yet it returned 12 similar matches using the TSM-L algorithm. The 7th highest total for Ambiguous names is *Result*, and it returned to highest number of matches in either category with 17 similar matches. The authors recognize the fact that *Result*, by definition, has different meanings for different services and sufficient matching depends on the context of the service.

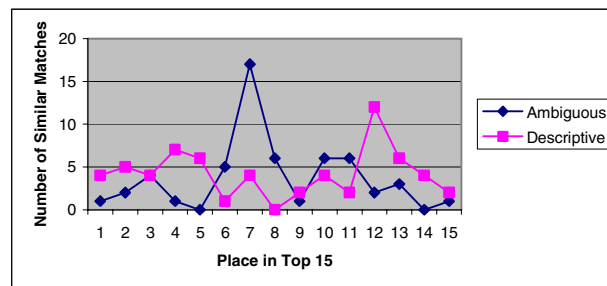


Fig. 4. Total of Unique Similar Strings by Top 30 Most Common Message Names

Table 3. Representative similarity results

String	Similar Strings (<i>false positives in italic></i>)
State	StateAbbrev, <i>StartDate</i> , state1, state2, <i>SQLStatement</i> , StateCode, USState, <i>StartYear</i> , <i>StartTime</i>
Identifier	id, <i>ie</i> , Identifiers, IdentifierType, IssuerIdentifier, IssuerIdentifierType
Password	<i>word</i> , phoneAlertPassword, lcPassword, password1, password2, host_password,
Country	CountryCode, CountryName, country1, country2, strCountry, <i>County</i> ,

4.3 Evaluation

To evaluate the effectiveness of the TSM-L function, we chose 4 other approaches for comparison. The baseline approach, *basic equating*, simply looks for strings that completely match one another (i.e. using a string.equals method). Two other approaches consider a static number of LD transformations as threshold without considering naming tendencies. We chose 5 because that was the average string size of all message part names. We chose 11 because it was more than twice the average. Finally, we consider a dynamic threshold for LD whereas the threshold is equal to the number of characters in the message name being analyzed. The results show that the TSM-L approach consistently produces more matches than the baseline (basic equating) for both ambiguous message names (Figure 5) and descriptive names (Figure 6). The TSM-L approach produces on average 35% more matches than the baseline (this may be difficult to see in the Figure 5 and 6). The other approaches get much larger numbers of similar messages; however most of the matches are false positives as shown in Figure 7 and Figure 8. The TSM-L approach performs close to the baseline approach that by definition should not have any false positives.

Figure 9 summarizes the accuracy of the TSM-L approach with respect to the previously-defined approaches. 98% of the message names that were deemed similar accurately describe the same type of information. (i.e. ~2% were false positives). The TSM-L approach performs much better than the static approaches to string transformation which do not consider naming tendencies.

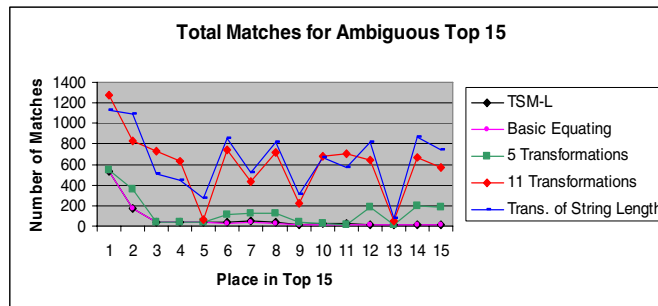


Fig. 5. Matching for Ambiguous Messages

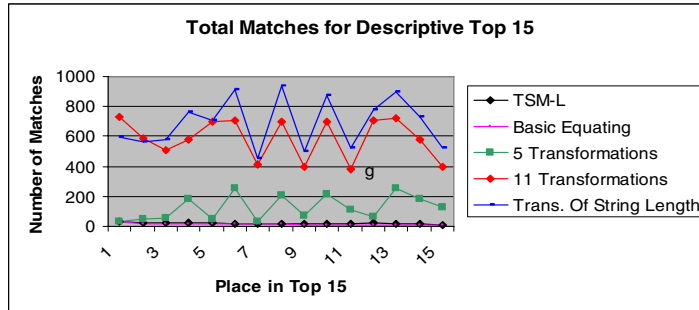


Fig. 6. Matching for Descriptive Messages

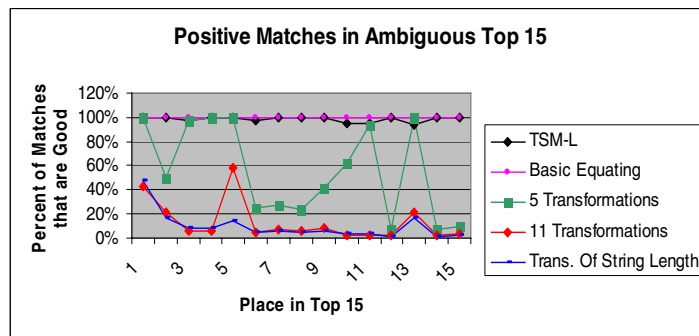


Fig. 7. Percentage of Positive Matches for Ambiguous Messages

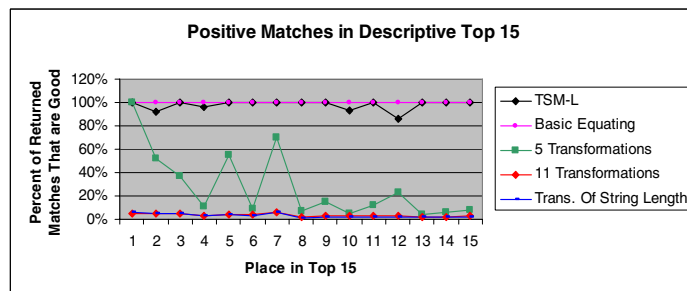


Fig. 8. Percentage of Positive Matches for Descriptive Messages

Obviously, this evaluation does not consider the context or format of the message, but relegates these lower-level issues to related semantic processing techniques. However, in a SOC scenario, the number of candidate services can be significantly decreased prior to semantic processing using our approach.

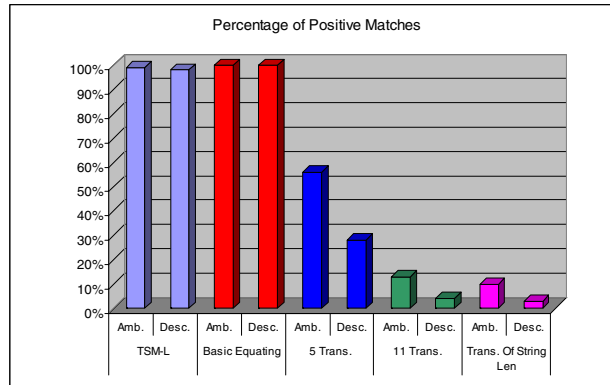


Fig. 9. Summary of Percentages of Positive Matches

5 Conclusion

In this work, we propose an approach that would facilitate semantic processing by classifying web services by their message names. We introduce a new syntactical matching algorithm, TSM-L, that extends the Levenshtein Distance algorithm by adding conditions that emulate the message naming tendencies of real-world web services developers. The true innovation of our work is this bottom-up concept of incorporating human software development tendencies into automated string processing techniques. TSM-L is a preliminary approach to validate that the combination of string manipulation algorithms and human concerns can be effective. Although the results show that the algorithm can make message name matches with a high degree of fidelity, there is no time-effective approach to evaluate *false negatives* (i.e. the message names that should have been discovered but were missed). In future work, we plan to investigate other analysis approaches to incorporate a false negative evaluation. We believe that such an evaluation can be used as feedback to enhance the TSM-L algorithm.

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.

References

- [1] Amazon Web Services (2006): 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 (2006): <http://www.ibm.com/developerworks/library/specification/ws-bpel/>
- [7] DAML (2006): <http://www.daml.org>
- [8] 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
- [9] McIlraith, S., Son, T. and Zeng, H. Semantic web services. *IEEE Intelligent Systems*, 16(2):46{53}, March/April 2001.
- [10] Medjahed, B., Bouguettaya, A. and Elmagarmid, A. K. Composing Web services on the Semantic Web. *The VLDB Journal*, 12(4), November 2003.
- [11] Merriam Park Software (2006): <http://www.merriampark.com/ld.htm>
- [12] Mindreef Soapscope (2006): <http://www.mindreef.com/products/soapscope/index.php>
- [13] NIST Levenshtein Distance (2006): <http://www.nist.gov/dads/HTML/Levenshtein.html>
- [14] OWL-S (2006): <http://www.daml.org/owl-s/>
- [15] Papazoglou, M. "Service-oriented computing: Concepts, characteristics and directions. *In Proceedings of WISE '03*
- [16] 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, (to appear)
- [17] 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
- [18] 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/>
- [19] 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.
- [20] 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
- [21] WS-Challenge (2006): <http://www.ws-challenge.org/>
- [22] XMethods (2006): <http://www.xmethods.com/>