Predicting Service Mashup Candidates Using Enhanced Syntactical Message Management

M. Brian Blake and Michael F. Nowlan
Dept. of Computer Science
Georgetown University
Washington, DC 20057
{mb7, mfn3}@georgetown.edu

Abstract

The descriptiveness of capabilities advertised on service-oriented architectures provides a promising platform for crafting new knowledge. Service mashup has been introduced as an approach for integrating the information provided from multiple web services into one common operational picture. In the future, scale will be a barrier to these types of approaches. With the entry and exit of large numbers of services on the Internet, it will be difficult to find and suggest the most relevant service candidates for new mashups. In this work, we present an efficient syntactical approach for actively discovering web service candidates for service mashups. This approach leverages the message naming characteristics of the developers and of the target service repository to inform search algorithms. Favorable precision results are described based on experimentation executed on an open repository of web service from the Internet.

Keywords
Web services, service mashup, service-oriented computing

1. Introduction

Service-oriented computing supports the notion of thousands (even millions) of modular software capabilities discoverable and usable by disparate organizations. The combination of the Web Service Description Language (WSDL) and the invocation methods supported by SOAP are important as they provide machine-interpretable interfaces and interactions [24]. Such open approaches provide an environment where atomic modules (i.e. web services) can be discovered and composed on demand. This sort of interaction, in a business setting, is optimal because it allows the underlying business units or divisions to act autonomously while, at the same time, facilitating active collaboration at a layer of granularity that each stakeholder can dictate and control. As such, a business unit can seek out relevant capabilities then consume and compose them in terms of their existing capabilities to create entirely new offerings.

Web 2.0 [22] is a paradigm that overlays the notion of service-oriented computing. In conjunction with the fundamental paradigms, Web 2.0 advocates for the individual user to be the prime stakeholder. Consumer-to-consumer collaboration technologies and market-oriented environments allow individual users to interact seamlessly. As such, the individual user is now beginning to exploit web services in a manner most appropriate to their daily activities. While business process composition is not a necessary action for the individual end user, the integration of the resulting data into a common view can be of importance. A service mashup is the simultaneous execution of two or more services to create an integrated tool that provides a more complete description about some object or characteristic. For example, web services from Google Corporation that provide mapping capability can be integrated with capabilities from the United Parcel Service (UPS) to understand the path of parcels that get lost in the mail. This example is illustrated in Figure 1. This integration of web services outputs is the general idea behind service mashup. The notion of service mashup currently receives a great deal of attention from academia and industry. Much of the current work involves tools and techniques for integrating web services information and furthermore the visualization of that information. In our work, we are interested in the discovery of candidate web services prior to the actual act of mashing them up (i.e. service mashup discovery). The technology for integrating service outputs (i.e. the fundamental requirement for service mashup) lies in the broader area of data integration. In this area, semantics and more specifically languages such as the Resource Definition Framework (RDF) [27] and the Web Ontology Language for Services (OWL-S) [23] play a significant role. However, we consider that open services randomly offered over the Internet are, at least currently, not described in terms of semantics. And, even if they use semantics, they would not adhere to a common ontology.
As such, our approach uses syntactical, natural language approaches to predict when the underlying web services messages are related. Furthermore, we use adaptive software to analyze characteristics of available service repositories to set the thresholds for our syntactical matching software. As such, we use human naming tendencies and the characteristics of service developers to inform our matching algorithms.

This paper continues in the next section with a discussion of related work in the areas of service composition and mashup. The subsequent section discusses how previous work on syntactical matching is leveraged for service mashup. The final sections detail our approach for predicting services mashup using syntactical matching and the corresponding experimentation and evaluation.

2. Related Work

The notion of comparing software interfaces for similarity has been explored in traditional software engineering venues [11][35]. However, our approach specializes the similarity search with regards to markup language interfaces in the service-oriented computing domain. As such, the approaches to service mashup are similar to the fundamental techniques for the discovery and composition of web services [14] [26]. Two common approaches to composition and consumption and perhaps mashup are semantic and syntactic techniques. Semantic approaches generally support the integration of web services by exploiting the semantic description of their functionality using ontological approaches [2][30][17][32]. Conversely, syntactic projects tend to concentrate on string manipulation and thesauri approaches to correlate services.

Our approach to service mashup is generally related to the area of syntactic information management. Rocco [28] uses rigorous string manipulation software to help equate web services messages while Pu [25] uses an eXtensible Markup Language (XML) type-oriented rule-based approach. Dong, Halevy et al. [10] introduces a web service search engine named Woogle. Most of these approaches use syntactical methods associated with string manipulation, case-based rules, and clustering algorithms to infer similarity in web service message parts. Our approach attempts to understand how developers name their services from the bottom-up. By categorizing services and understanding the characteristics of messages in a particular domain, we can build specialized algorithms to predict when two independent web service messages are related. The innovation in our work is one that 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 [5][3].

Although the area of data integration has had a longstanding background, the usage of these techniques to accomplish mashups has only just recently started to be addressed. A majority of the work in this area address the tools and environments that support the visualization environment that presents mashup results [12]. Other projects describe enabling techniques for preparing services for mashup [15][29]. These are also projects that investigate the policy for protecting data in mashup environments [36] and instituting enterprise policy for modernizing systems using the information resulting from successful mashups [9].
3.1 Discovering Trends in Web Service Message Naming

Our approach uses specialized parsing tools to evaluate services from the bottom up. We sought to discover trends across the open web services. As a result, approximately one dozen trends were discovered. The three most common trends are listed below.

**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 to have strong subsumption relationships. For example, there are equivalent services where \( \text{name} = \text{first name} \), \( \text{name} = \text{first name} \), and \( \text{name} = \text{user name} \).

**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 \( \text{first name} = \text{user name} \).

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

3.2 Using Natural Language Approaches to Exploit Trends in Message Matching

Perhaps, the most straightforward comparison for two part names is syntactic equivalence, or if one part name includes the other. For those cases, special natural language approaches are not necessary. However, several syntactic approaches were used to exploit Trend 2 and 3 when matching services. The Levenshtein distance (LD) (also called the edit distance) is a measure of similarity between two strings [20]. LD is the smallest number of deletions, insertions, or substitutions required to transform a source string into a target string. The greater the LD, the more different the strings are. In this work, we adapted implementations of the LD algorithm from pre-existing sources [18][20]. 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 \( O \) or numbers are added to the end of a message name.

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, \( \text{last name} \) and \( \text{surname} \) 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 strings are separated into letter pairs. STR1 would be separated into \( \text{ST}, \text{TR}, \text{and R1} \), and STR2 would be separated into \( \text{ST}, \text{TR}, \text{and R2} \). A ratio of the number of identical pairs and the total number of pairs is calculated. This percentage is used to evaluate the similarity of the two strings.

An innovation of our work is adjusting the thresholds for the LD and the LP percentage based on the category of the web services that is being analyzed. Again this is described in detail in earlier work.

3.3 Various Approaches for Syntactical Message Matching

The overarching matching algorithm is called Tendency-Based Syntactic Matching-Levenshtein Distance and Letter Pairing (TSM-LP). This algorithm exploits the previously mentioned tendencies using LD and LP. However as a part of the service mashup experimentation we will evaluate with the following 4 methods of syntactical matching.
Equality. This method checks to see if two message parts are syntactically equivalent.

Subsumption. This method evaluates whether two message parts have a subsumption relation (i.e. message part 1 is a subset of message part 2 or vice versa).

Levenshtein Distance (TSM-L). This method adds to equality and subsumption by also checking if the message parts are similar based on edit distance. Three thresholds based on varying rigor govern the algorithm. The three thresholds are determined based on the degree of self-similarity for the particular category for which the service is associated [3].

Letter-Pairing (TSM-P). This method adds to equality and subsumption by evaluating if message parts are similar based on letter pairing. Strictness thresholds are based on repository similarity and align with TSM-L.

Levenshtein Distance/Letter-Pairing (TSM-LP) This method combines all of the previously mentioned methods.

4. Discovering Service Mashups

A web service requires specific input messages and provides relevant output messages. An effective mash-up blends two services that are not similar in function, but rather similar in the messages that they provide. In this way, identifying a similar message part, either input or output, between two services is the first step in our approach to identifying a mashup. Figure 3 demonstrates how mashing-up services involves finding similar or equal part names among services.

Our approach to service mashup is a systematic approach for discovering web service operations that share syntactically-equivalent message parts or share a slightly syntactically-different message parts but with equivalent meanings. The approach starts with the selection of an initial operation for which we discover other candidate services for mashup. Next, the chosen operation is compared to all other operations that are not contained within the same WSDL file. In the comparison, message parts of the initial web service operation are evaluated for equivalence or similarity to any of the message parts of each of the other web service operations. If equivalent or sufficiently similar (based on the strictness thresholds), a potential mash-up is declared. The current goal of this work is to determine how many shared output message parts are most relevant to the prediction. In future work, we plan to determine the type of messages, either input or output, that are most predictive of mashup.

Our approach also checks the overall similarity of the two services being evaluated. If they share a high number of input and output message parts, then there is a high probability the services do the same thing or are the same service. In these cases, the algorithm does not declare a potential mashup, particularly if two web service operations have more than 75% of their message parts related. The approach for the mashup prediction algorithm is shown in Figure 3. The reader should note that the string-to-string comparison specified in Figure 3 is TSM-LP, although all of the similarity methods are evaluated in our experimentation.

Figure 2. Approach for Discovery Potential Service Mashup Candidates.

<table>
<thead>
<tr>
<th>Mash(OP₁, OP₂)</th>
<th>Mashup Prediction Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TSM-LP(Pn₁, Pn₂)</td>
<td>Similarity Function (Section 3)</td>
</tr>
<tr>
<td>OPₓ</td>
<td>Web Service Operation</td>
</tr>
<tr>
<td>Pnₓ</td>
<td>Message Part</td>
</tr>
<tr>
<td>match</td>
<td>Number of Similar Matches</td>
</tr>
<tr>
<td>size</td>
<td>Number of parts in an operation</td>
</tr>
</tbody>
</table>

Mash(OP₁, OP₂)
  forAll(Pn₁, Pnₙ)
  forAll(Pn₂, Pnₙ)
  if(TSM-LP(Pn₁, Pn₂))
    match++
    break
  endFor
  if(match / OP₁.size < .75)
    return true
  else
    return false

Figure 3. The Systematic Approach to Predicting Service Mashup Candidates.
5. Experimentation and Evaluation

Our approach was evaluated on our open repository of nearly 600 real services (containing over 5,000 web service operations). The only method with confidence for assessing that two services represent a valid mashup is through manual inspection. Considering the large number of web service operations, we took a random sample of 100 operations for our experimentation. The random samples were chosen because they contained a higher number of message parts than the average for the repository. Additionally, we reused our similarity software (i.e. TSM-LP) to ensure that the chosen services had relatively diverse input/output messages. As a result, we believe that the operations that were chosen for experimentation represent relatively diverse input/output message parts to reflect the diversity in the repository.

In the experiment, each of the 100 web service operations was evaluated independently against the other operations in the sample set. In the case of each operation, each independent message part was evaluated against the message parts of other evaluative operations. Figures 4 – 7 illustrate the quantitative and qualitative results of using the various similarity matching approaches for predicting service mashup candidates.

5.1 Qualitative and Quantitative Experimentation

As described previously, the TSM-L, TSM-P, and TSM-LP string-to-string similarity methods have various strictness thresholds that determine what magnitude of mashup predictions would result from varying the similarity methods. Sensibly, the results reflect that there are far fewer mashup predictions made when the message parts had to be completely syntactically equivalent. However, finding 65 potential service mashups based on syntactically equivalent message parts is perhaps effective in practice. We were, in fact, surprised to find so many in the open repository of random services from the Internet. 65 potential mashups represents just over 1.1% of the operations in the entire repository. The method that returned the most mashup predictions utilized TSM-L method. Interestingly enough, this suggests that there are many cases of abbreviations within the message parts.

In the next experiment, we investigated how the number of predicted mashups changes as we vary the number of similar message parts. Most specifically we ask the question, “How does the number of predictions change as the requirement for shared message parts is increased?” As expected the number of predictions decreases as the requirements for shared message parts is made more stringent. Figure 6 shows that the number of predictions have a polynomial decrease with relation to the increase in required shared message parts. In addition, the decrease in predictions is consistent irrespective of the similarity method used.

The final experiment was performed to determine the precision of the predictions with regards to suggesting valid service mashups. Since the experiments were performed on an open repository of web services, manual inspection is the only approach for verification. We declared a mashup when there was significant valid overlap in the output message parts of two operations. As previously mentioned, we used a subset of the repository consisting of 100 web service operations. Even with this subset, there were, in some cases, over 1000 predictions. As a result, in determining precision, we wrote software that randomly selects 30 suggested mashups for each method and for each variation in the requirement for shared message. Figure 7 illustrates favorably that the average precision was approximately 80% across all methods. Considering the level of precision and the large number of predictions, the results suggest that TSM-L produces the largest number of valid service mashup predictions. Another interesting result was the impact of increasing the requirement for shared message parts. Although the increase in required shared message parts increased the precision, the increase was not particularly substantial. In most cases, the precision increase from requiring 1 shared message part to 3 shared message parts was 5% to 15%. However, Figure 6 shows that the corresponding decrease in the number of predictions was a magnitude greater.
5.2 Performance Assessment

To assess the variance in performance across the prediction methods, we recorded the average time for each method over 5 different executions. Figure 8 shows the average time it takes for the method to use one web service operation and predict service mashup candidates from a repository of approximately 100 diverse operations. All methods can complete the prediction process in less than 1 second. This is a favorable result with regards to making on-demand predictions of mashup services.

5.3 Sample Predictions and Common Shared Message Parts

An assessment of the content of the set of valid service mashups shows a large number of mashups between weather services and address services. Also, many stock symbol look-up services had valid mashups with other financial services such as company information or currency converters. In some cases, our approach inadvertently predicted valid mash-ups that were suggested through ambiguous message parts such as “message” or “code”. There were several message parts that occurred frequently. Table 2 lists the 6 most common message parts.

<table>
<thead>
<tr>
<th>Top 6 Most Common Message Parts for Predicting Mashups</th>
<th>Percentage of Top 6 Used for Predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>State, City, Name, Date, Time, Zip</td>
<td>29%</td>
</tr>
</tbody>
</table>

These message parts represent the base string for our similarity approaches, although in most cases, the web
service operations use similar strings. As further validation to our work, the most common message parts, perhaps as expected, revolve around the dimensions of person, location, and time.

Our approach concentrates on evaluating two services for mashup compatibility. However, the next logical and straight-forward step is to chain together pairs of mashup predictions to create mashups of multiple operations. In order to demonstrate this, several mashups were connected as predicted by the TSM-LP algorithm that could be executed simultaneously. There were several particularly interesting mash-ups produced. One mashup, illustrated in Figure 9, can be used to describe an individual based on their phone number, street address, email address, credit card history, airport, weather, and email information. In this case, mashup editing software, as described in related work, is required to integrate the outputs and in some cases the inputs. In the case of these four services, the mashup is straightforward. As described earlier, the dimensions of person, time and location inherently attach all message parts.

6. Discussion

In our early work, we analyzed existing web services on the Internet to gather insight about how services are developed and how service-based messages are named. As a result, we developed several natural language processing approaches (i.e. TSM-L, TSM-P, TSM-LP) that mirror the nature of the open web services. In this work, we explored how these approaches could be applied to the domain of predicting service mashups. Results show that the TSM-L method provides the largest number of valid predictions. In addition, the recommendation performance is favorable with regard to making real-time recommendations. The TSM-L method is most effective on messages names that utilize abbreviations. This suggests that the open repository contains many abbreviations as a part of service messages.

In future work, we plan to assess the ability to predict service mashups using a combination of inputs and outputs. In addition, we plan to aggregate the most useful message parts to develop core objects for new web service developers. An interesting future application is the development of a distributed web services development environment that leverages the knowledge of existing services. Web service developers may, in real-time, be recommended web services that might be consumed in the new operations that they develop. Another future project would be the integration of our approach for use as the pre-processor for emerging service mashup editors and visualization environments. In other words, our approach can suggest mashups, and subsequently the editors and visualization tools can verify the output.

7. Acknowledgements

This work was partially funded by National Science Foundation Grant Numbers 0548514, 0634302, and 0723990.

References


