Web Service Matching for RESTful Web Services

Reihaneh Rabbany Khorasgani, Eleni Stroulia, Osmar Zaiane
Computing Science Department
University of Alberta
Edmonton, Canada
rabbanyk,stroulia,zaiane@ualberta.ca

***Abstract*—There is a growing number of web services available on the Internet, providing a wide range of functionalities. This diversity introduces a variety of new challenges in the field of software engineering - service discovery, integration, and composition, all of which require, to some extent, "service matching". Web-service matching (or alignment) is the task of mapping the functionalities of two web services, assuming that these functionalities overlap somewhat.**

In this paper we propose a novel graph-theoretic approach, called Semantic Flow Matching (SFM), for matching REST web services, specified in WADL (Web Application Description Language). The method builds a heterogeneous network of WADL elements and semantically related terms, and uses this network to match similar functionalities of different web services. The method is implemented in a prototype tool that consists of two modules: a converter and a mapper; where the converter wraps the REST web services in WADL format and the mapper module matches web services based on their semantics extracted from the WADL interface build by the converter. We demonstrate the potential of the approach with a small case study.

***Keywords*-Web Service Matching, REST, WADL**

I. Introduction

There are a multitude of web services and APIs available on the web. These APIs provide a wide range of different services and there usually is a substantial semantic overlap between them, with many web services providing essentially the same functions. Such overlapping functionalities enables redundancy in the web-service ecosystem, and gives developers the opportunity to migrate from one API to anther when the API originally used becomes unavailable or insufficient for their needs. To support the discovery of similar APIs, several new methods are being developed for web-service discovery, including query-based methods relying on keywords and identifiers [14], [1], clustering [2] and more detailed structure matching [3], [4].

There are a variety of activities that require web-service matching. Let us consider, for example, service migration: clients that are using different web services for the same task, should easily be able to share their data, migrate from one service to the other in the case of service deprecation and also have a single point of access to their data distributed over different web services. As an example, consider the Del.icio.us social bookmarking web service, which is currently for sale by its owner Yahoo! and there are many users worried about their data. As a result several other bookmarking sites provide means to import Del.icio.us data. Here if one could automatically build a matching between Del.icio.us and any other similar service, we could simplify the migration. This would also enable users to use Del.icio.us with Diigo, Zootool and Hbookmark (other social bookmarking service), *etc* at the same time and with a single point interface.

In principle, service-matching techniques rely on interface matching, behaviour matching or a combination of both (which is more likely to produce better result) [4], [3]. Interface matching methods usually rely on the assumption that services are described in a common interface-description language, such as WSDL (Web Service Description Language) for example [14], where behaviour-matching approaches also consider the usage context around the services, as specified in client applications or BPEL (Business Process Execution Language) descriptions [4].

Motivated by the increasing trend of Web 2.0 applications from SOAP-based web services to RESTful, here we consider web services described in WADL instead of WSDL. WADL (Web Application Description Language) is a description for RESTful web services and an alternative for WSDL descriptions of SOAP services. Focusing on interface matching, we propose a novel method to match functionalities of two given RESTful web services based on their WADL description/interface by building a semantic network to capture their semantic relation. Our proposed method incorporates linguistic knowledge and domain-specific heuristics to automatically match given WADLs. It creates a heterogeneous network of WADL elements and related semantically synonym terms, then searches for the most similar methods (of different web services) by looking for the ones with maximum semantic flow between them.

In the rest of this paper, we first overview RESTful architecture and WADL description followed by an overview of currently available methods for Web Service Matching (Section 2). Then we elaborate on our proposed method for matching RESTful web services based on their WADL description (Section 3). This is followed by our results for matching different real world social bookmarking web services (Section 4). Finally we conclude our work, enumerate the main challenges and possible future directions (Section

5).

II. Background and Related Work

Although the problem of web-service matching is related to the problem of similarity-based search of web services (as both are relevant to the task of service discovery), methods developed for the latter task usually only measure the similarity between different web services but do not necessarily provide a complete matching of their operations.

Generally, the problem of API matching in web services is similar to several matching problems in other fields: *schema matching* in Databases, *text document matching* in Information Retrieval and *software component matching* in Programming Languages. However, there are some differences that make techniques developed in these other domains not quite suitable [1].

In Database *schema matching*, schemas are matched based on their predicted semantics and is useful in information integration. The schema matching methods mostly contain some linguistic and structural analysis. They also use the domain knowledge and previous mapping experiences to boost their results [6]. There are two main issues with these approaches that make them unfavourable in the context of web service matching. The first problem is that schema matching and web service matching have different granularities. In web services we are trying to map functionalities of two schemas while in schema matching we are trying to find related components of two schemas which compares to the granularity of matching parameters of web services rather than their methods. The other reason is that in schema matching, schemas are highly matching which is usually not the case in web services. Web services usually have small overlaps and often rather than substitute they complement each other.

Document matching is a long-studied problem in Information Retrieval where term/doc frequencies are used for matching [7]. This reliance on term frequencies, make these approaches unsuitable in web service matching because web service descriptions do not contain adequate text and also there is rarely text description of web services. On the other hand, web services contain other structural information that would be ignored by these techniques.

Another closely related problem is *software component matching*, an important topic in software reuse [8]. This problem is concerned with signature matching of different procedures considering their data types, parameter names, parameter types, parameter order *etc*. Some specification mapping variants also consider program behaviour and post conditions. Despite apparent similarity of signature matching and web service matching, better scrutinization shows different level of expressiveness and data structures, makes naïve application of methods in one domain inefficient in the other, however there has been successful adaptations in the similar problems [9], [10]. In [9], Stroulia *et al.* computed structural and semantic similarity of web services for the web service discovery, leveraging from mainly signature matching methods.

Web-service discovery is a problem closely related to web-service matching. Web service discovery is about searching the web services repository (UDDI – Universal Description Discovery and Integration) for a desired web service. The common web service search systems are based on keyword search which results in some shortcomings. For example these basic keyword search methods can not return a web service that contains 'zip' or 'postal code' when the user searches for 'zipcode' [1]. Some recent research in this area has focused on using semantics to get better results. This is achieved by

- adding semantic metadata to the web service descriptions [2] and providing new semantic languages to describe web services *e.g.* OWL-S, WSMO [11], [12]; and
- inferring semantics by clustering similar web services [13], [1], [5] or using information retrieval techniques [5].

One of the most related works in service discovery is Woogle by Dong *et al.* [1]. Woogle is a web service search system that looks for similar web service operations for a given operation. Woogle clusters parameter names in web services into semantically meaningful concepts and compares inputs and outputs of operations with these semantic concepts to measure similarity of operations. Clustering is based on the co-occurrence of names together with the assumption that "parameters tend to express the same concept if they occur together often" [1]. However we found this assumption too strong and not surprisingly invalid in practice.

Motivated by the automated web-service discovery and use, a recent line of research in web service matching has emerged. Wang *et al.* [14] proposed a flexible interface-matching suite of methods, based on the structure of their data types and operations and the semantics of their natural language descriptions and identifiers, as described in their WSDL specifications. Given only a textual description of the desired service, a semantic information-retrieval method can be used to identify and order the most relevant WSDL specifications based on the similarity of the element descriptions of the available specifications with the query. If a (potentially partial) specification of the desired service behaviour is also available, this set of likely candidates can be further refined by a semantic structure-matching step, assessing the structural similarity of the desired vs the retrieved services and the semantic similarity of their identifiers. In this paper, we describe and experimentally evaluate our suite of service-similarity assessment methods. Motahari-Nejad *et al.* [3] introduced a semi-automatic interface matching approach which incorporates the ordering constraints imposed by business protocol definitions on service operations. Both Mikhaiel *et al.* [4] and Motahari-Nejad *et al.* [3] argue in favour of using both Interface Matching (*e.g.* WSDL) and Protocol Matching (*e.g.* BPEL) to have a more precise and effective web-service matching. Motahari-Nejad *et al.*'s method, has in its core, XML schema matching for interface matching. They also incorporate protocol level information into this, both by considering the depth of the operation in the protocol description and by an iterative reference-based similarity propagation through protocol and already matched operations.

The majority of methods proposed so far for web-service matching, assume that services are described in WSDL. However there is a trendy migration of the Web 2.0 (participatory Web) applications from SOAP based web services, defined by WSDL, towards RESTful services which do not necessarily use WADL. This makes matching RESTful web services an interesting problem [15], [16], [17].

REpresentational State Transfer [18] is a software architecture style that incorporates clients sending specially structured HTTP requests to servers that include the specification of how to access information resources, and servers that process the request and return the response. A client is either at "rest" which means it is able to interact with its user, or it is in transition. This happens when the client sends a request and awaits the response. These requests and responses are the transfer of representations of resources as they includes the current and intended state of resources.

In the REST style, every resource has a global identifier (*e.g.*, a URI in HTTP) and the client just needs to know this identifier and the required action. It also needs to understand the format of representation, which is typically an HTML, XML or JSON (JavaScript Object Notation) meta-data. RESTful web services, *a.k.a.* REST APIs, specify a collection of resources which consist of three components: URI of the web service, type of the data supported: JSON, XML, YAML, *etc.* and operations supported using HTTP methods (*e.g.*, POST, GET, PUT or DELETE). A REST API may be described using WSDL and use SOAP over HTTP, or it could be an abstraction purely on top of SOAP (*e.g.*, WS-Transfer) or without using SOAP at all [19]. The W3C standard for describing a REST web service is WADL which is a REST alternative to WSDL [20].

WADL, the Web Application Description Language, is a machine processable XML description for REST. In WADL, a service is described by a set of resource elements, where each resource consists of a set of hierarchically structured elements (sometimes specified in a XML Schema Definition; the parameters that describe the input to the methods through which the resource is accessed and the methods' responses are defined in terms of these elements. A request specifies how to represent the input, what types are required and any specific HTTP headers that are required. A response describes the representation of the service's response [21].

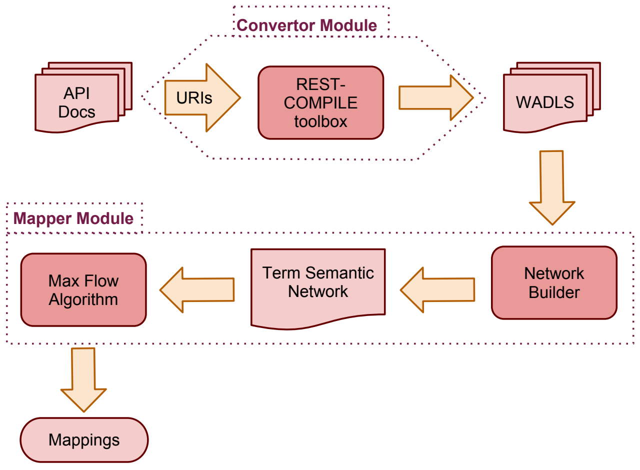


Figure 1. Overall Flow of the REST API Alignment Method. The converter module (on the top) wraps the REST API docs into WADL format. The mapper module (on the bottom) matches web services using their WADL interfaces. It consists of two submodules – a network builder and a maximum flow algorithm.

III. The Semantic Flow Matching Method

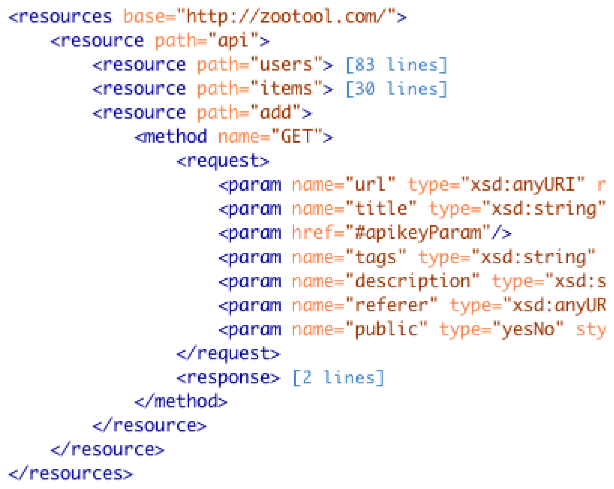
Figure 1 illustrates the overall process of our Semantic Flow Matching (SFM) method, which consists of two main modules – a converter and a mapper. The converter wraps the REST web services in WADL format and the mapper module matches web services based on their semantics extracted form WADL interface and using the SFM approach. The converter module (described in III-A) actually prepares our dataset of WADLs for the SFM approach (described in III-B). This converter module is necessary since WADL is not a commonly used description and not always present where the WADL specifications are required for the mapper module.

A. *The WADL-Converter Module*

The converter module transforms the HTML documentations of REST APIs into WADL specifications. We have used a semi-automatic method for this transformation, using the Google REST Describe toolbox [22] [1]. The Google "REST Describe" tool creates WADL descriptions of web applications by analyzing sample request URIs, where one could manually add or modify metadata to improve the automatically generated WADL description.

More specifically, we used the client interface of Google's REST-Describe to create the WADL specification of a given REST API. To do so, we first manually extracted a list of the HTTP requests that the API supports. Then we automatically converted this list of URIs to its corresponding WADL,

[1]A demo can be found here: http://tomayac.de/rest-describe/latest/RestDescribe.html# and the code can be downloaded from here: http://code.google.com/p/rest-api-code-gen/



```
<resources base="http://zootool.com/">
    <resource path="api">
        <resource path="users"> [83 lines]
        <resource path="items"> [30 lines]
        <resource path="add">
            <method name="GET">
                <request>
                    <param name="url" type="xsd:anyURI" r
                    <param name="title" type="xsd:string"
                    <param href="#apikeyParam"/>
                    <param name="tags" type="xsd:string"
                    <param name="description" type="xsd:s
                    <param name="referer" type="xsd:anyUR
                    <param name="public" type="yesNo" sty
                </request>
                <response> [2 lines]
            </method>
        </resource>
    </resource>
</resources>
```

Figure 2. An example for the generated WADL files. This WADL is obtained from Zootool's API. All the elements are closed for more clarity, and only the 'add' method is expanded. This figure is generated using the Author XML editor toolbox.

using the URI batch-processing option in the REST-Describe client interface. We further manually edited the resulting WADLs to add the responses (response and representations elements that explain the response media type), documents (doc elements that describe methods in free text) and some information about parameters (*i.e.* those that could not be inferred from the URIs such as their default value, type and if they are required or not).

Figure 2 presents one of the generated WADL files using the converter module, which corresponds to the Zootool API. This WADL specification contains several resources that one could modify through the Zootool's API. For example the resource with the path value 'add' (which is expanded in the figure) has a 'GET' method which calls the API with the following sample URI: http://zootool.com/api/add/?url=http://www.google.com&title=Google&tags=search, google&description=searchEngine&public=y. This method adds a new item to the user's bookmarks, which also has some parameters to set information about this bookmark (such as 'tags' and 'description').

B. The WADL-Mapper Module

The Mapper module is responsible for the core interface-matching method, namely to recognize similar functionalities of two given REST web services based on their WADL descriptions. Before providing a detailed description of the method, let us argue for its rationale. The goal of our method is to find a partial matching between groups of structured data, where the underlying data structure is a shallow tree of three levels – resources, methods and parameters. The actual objective is to match the elements at the second layers of the trees, *i.e.* the methods. One should notice that although the usual WADL files have more levels of structure for defining global and nested elements, this structure does not actually convey any information about the functionalities of the underlying web service and only is an ease for flexible API documentation and mostly preventing repeated descriptions. Furthermore, the size of each web-service tree is small, as every web service provided a small set of functionalities. The matching is uncertain, as two methods may not have the exact same number of parameters and parameter names do not match. In some cases they are not even synonyms *e.g.* tag and bookmark.

A first approach we considered was to use some off-the-shelf Machine Learning (ML) solution. To make a problem accessible to ML methods one has to develop, out of the primary data, feature vectors or similarity measures (to be used in conjunction with a Kernel method); if there exists a sufficient quantity of labeled data, one may thus learn a good representation. However in our case, because of lack of labeled data (*i.e.* already matched web-services), most informative features should be extracted manually. Here beside structural information, linguistic relations (similar names of two parameters) should be considered in such feature design. While automated design of such feature set using statistical learning techniques is an standard approach in NLP, manual design is deemed very difficult.

Due to lack of labeled data, one may consider unsupervised techniques, *i.e* clustering for the task at hand. Here the goal would be to have each cluster contain similar methods from different services. However usual clustering methods are not suitable here because of two reasons. In many cases, a correct solution would imply that many clusters should only have a single member (for methods in one service with no corresponding methods in the other service), which is , in principle, undesirable in clustering. Furthermore, since methods in a single service tend to share many similarities, such as common naming schemes for their identifiers and common parameters for example, usual clustering methods would tend to place them within the same cluster, which violates the problem specification that requires methods from different services to be matched against each other.

Recognizing that the most straightforward techniques were inadequate for our problem, we propose a heuristic method called Semantic Flow Matching (SFM). This is a graph-theoretic matching approach that incorporates linguistic knowledge to find the best matching. Our method works without labeled data (unsupervised) and leverages available toolsets in optimization (*i.e.* max-flow calculation).

The basic idea of SFM approach is illustrated in Figure 3. To match two web services w_1 and w_2, the mapper process first builds a heterogeneous network of their WADL elements and semantically related terms. In this network every method element is connected to its corresponding elements (currently parameters and resources), each of which is, in turn, connected to their corresponding terms in the semantic network of synonym terms. To find the proper match for

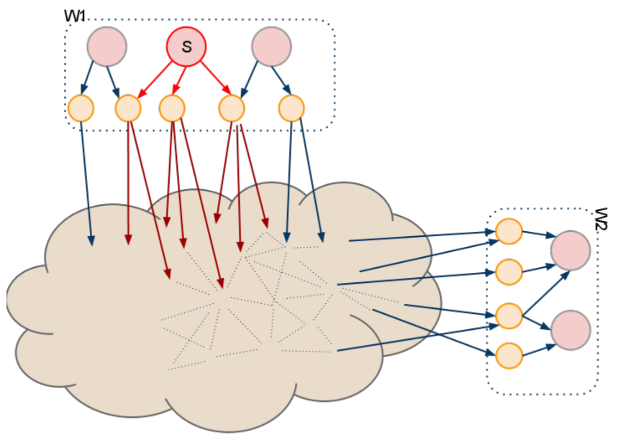


Figure 3. Semantic Flow Matching Network: an abstraction overall view. Nodes corresponding to the WADL elements of web services w_1 and $w2$ are illustrated in the dotted boxes. The middle cloud stands for the synonymity network extracted from WordNet, in which the generic English terms are connected to each other if they are synonym.

method s in w_1, the mapping process sets that method as a sink, directs all the edges from elements of w_1 outward and all edges to w_2 inward, and then searches for the method in w_2 that receives the maximum semantic flow form s. The corresponding method is a possible match for s. In the rest of this section we elaborate the process of building this heterogeneous network and finding the matchings.

1) Semantic Flow Matching Network: The SFM network has two types of nodes, WADL element nodes and term nodes. WADL element nodes consist of method, parameter and resource elements. It could be easily extended to include other elements as well; most interesting elements to add to the SFM network are the optional doc elements that document the functionality of methods in narrative text, as well as the responses elements that include the media-type of the output of these methods.

Each of the WADL element nodes are connected to one or more term nodes which are the English terms in the synonymity cloud, where term nodes are connected to each other if they are synonyms. More specifically, every parameter element is connected to the terms related to its 'name' attribute and every resource element is connected to the terms extracted from its 'path' attribute. For example the WADL resource element with path attribute value 'add' (expanded in Figure 2) is connected to the 'add' term node and the parameter element node with name attribute value 'tags' is connected to 'tag' term node in the synonymity cloud where it is connected to its synonym terms such as 'label', 'mark'.

We have incorporated two kinds of knowledge in building the SFM network – common natural-language processing knowledge, for node and edge discovery, and domain knowledge, for edge weighting. To connect the element nodes to term nodes we have used Porter stemmer [23] in order to extract the related terms to each WADL element. We also used WordNet [24] for adding edges between synonym terms. The SFM network edges are weighted using the following general heuristic rules, based on the domain knowledge [2]:

- Edges from element nodes are weighted more than edges between term nodes.
- Edges from method elements to resource elements are weighted more than edges to parameter elements.
- Edges to required parameters are weighted more than edges to regular parameters.
- Edges to resources of a method are weighted based on their path depth
(*e.g.* /api/tags/add $\rightarrow w(add) > w(tags)$).

2) Flow-Based Matching: The SFM algorithm is shown in 1. We excluded edge weighting details to limit the complexity of the presentation. The algorithm first builds the SFM network as described in Section III-B1, then uses it to find the possible matches between the methods of the compared services. It calculates the semantic flow from each method node, m_1, in the first given WADL file to every method elements, m_2, in the second WADL file and consider them as a match if this flow is greater than a predefined threshold (δ). These matches are inserted in M – a set of possible matches, sorted based on the maximum flow. More specifically, we set m_1 as a sink and compute the maximum flow from m_1 to m_2 in the SFM network where all the edges from elements of the first WADL is outward and all the edges to elements of the second WADL are inward (illustrated in Figure 3). Then we store this matching in the sorted set of possible matchings, M, if its maximum flow is greater that δ.

The maximum flow problem is a long-standing problem in optimization theory. It calculates the maximum flow from a source to a sink through the given flow network where edges have weight/capacity. There exists several methods for solving the maximum flow problem. Here we have used the Edmonds-Karp implementation of the maximum flow algorithm [25] implemented in JUNG, Java Universal Network/Graph framework [26]. The Edmonds-Karp algorithm is based on the Ford-Fulkerson method [**?**], which computes the maximum flow in order of $\mathcal{O}(|V| \cdot |E|^2)$. The basic idea of the Ford-Fulkerson method is as follows: while there exists a path from the source to the sink with available capacity on all the edges of the path, add this flow to the maximum flow, update the edge capacities and then proceed to the next path [25].

IV. Experiments and Evaluation

To evaluate our method, we have generated WADL specifications for several different available social bookmarking

[2]Implementation details including the exact weighting of edges based on these rules could be accessed in http://webdocs.cs.ualberta.ca/~rabbanyk/research/WS/src/logic/aligner/

Algorithm 1 SFM matcher

```
Require: $wadl_1$ and $wadl_2$

  {Building the SFM network}
  for all method element $m$ in $wadl_1 \cup wadl_2$ do
    add node $m$ to SFMnet
    for all WADL element $w$ related to $m$ do
      add node $w$ to SFMnet
      if $m \in wadl_1$ then
        add edge $< m, w >$ to SFMnet
      else
        add edge $< w, m >$ to SFMnet
      end if
      {connecting w to the synonymity cloud}
      if $w$ is a resource then
        terms ← extractTerms($w_{path}$)
      else if $w$ is a parameter then
        terms ← extractTerms($w_{name}$)
      end if
      for all term $t$ in $terms$ do
        add node $t$ to SFMnet
        if $m \in wadl_1$ then
          add edge $< w, t >$ to SFMnet
        else
          add edge $< t, w >$ to SFMnet
        end if
        {expanding the synonymity cloud}
        for all term $t_s$ synonym to $t$ do
          add node $t_s$ to SFMnet
          add edge $< t, t_s >$ and $< t_s, t >$ to SFMnet
        end for
      end for
    end for
  end for

  {Matching the method elements}
  for all method element $m_1$ in $wadl_1$ do
    for all method element $m_2$ in $wadl_2$ do
      $mf \leftarrow$ MAX-FLOW form $m_1$ to $m_2$
      if $mf > \delta$ then
        $ind \leftarrow 0$
        while $M(ind)_{mf} > mf$ do
          $ind \leftarrow ind + 1$
        end while
        insert $<< m_1, m_2 >$ , $mf>$ in $M(ind)$
      end if
    end for
  end for
```

APIs – Del.icio.us [27], Diigo [28], Hbookmark [29] and Zootool [30] [3]. Figure 4 visualizes the SFM network constructed for the mapping between Zootool and Del.icio.us. Figure 5 presents a closer look of the SFM network which illustrates connection between 'add tags' method in Del.icio.us and 'add' method in Zootool.

The SFM approach generates a set of possible matches of pairs of methods, each associated with a confidence value – the actual value of flow between that pair of methods. next, it chooses the non-overlapping matches from this set greedily – starting with the match with the highest confidence value, and proceeding down the confidence values, keeping those that do not overlap (common methods) with any of the previous matches. Here we report the obtained results for all the 6 combination of a pair of these APIs.

Table I shows the example URIs and our matching result for Diigo and Zootool, where SFM correctly maps the only two methods in the Diigo to their corresponding methods in Zootool. We could see that the SFM method correctly matched the 'POST bookmark' method from Diigo with the 'GET add' method from Zootool with a high confidence. It also correctly matched the other Diigo method, 'GET bookmarks' with its similar method in the Zootool.

In the other five combinations of APIs, our SFM approach found only a methods in common between these APIs since these APIs are very loosely coupled, where more than substitutions, they are complements for each other. For example the Zootool's API is more focused on the users and followers aspect of social bookmarking while the Del.icio.us API is has more methods for managing tags. Consequently the result of matching these two APIs would give us just one possible match with high confidence, 170 and the next possible match has a confidence of only 17, which should not even be considered as a possible match. Generally there is a big gap between the flow/confidence of highly matched methods and poorly matched methods and this δ parameter could be set automatically as the mean of all flows/confidences. Table II shows the matches for all the remaining combinations. We could see that the SFM approach finds only one match between Del.icio.us and Zootool, Del.icio.us and Hbookmark, and Diigo and Hbookmark. While it finds two matches between Del.icio.us and Diigo and three between Zootool and Hbookmark.

Table III demonstrates our evaluation result for the SFM method. Precision shows the number of pairs of correctly matched methods divided by the total number of matched methods, while the recall shows the the number of pairs of correctly matched methods divided by the total number of pairs that should be matched, the correct matches, which are counted manually. We could see that based on our case study data set that consists of six social bookmarking APIs,

[3]These generated WADLs could be accessed in main directory of the project code.

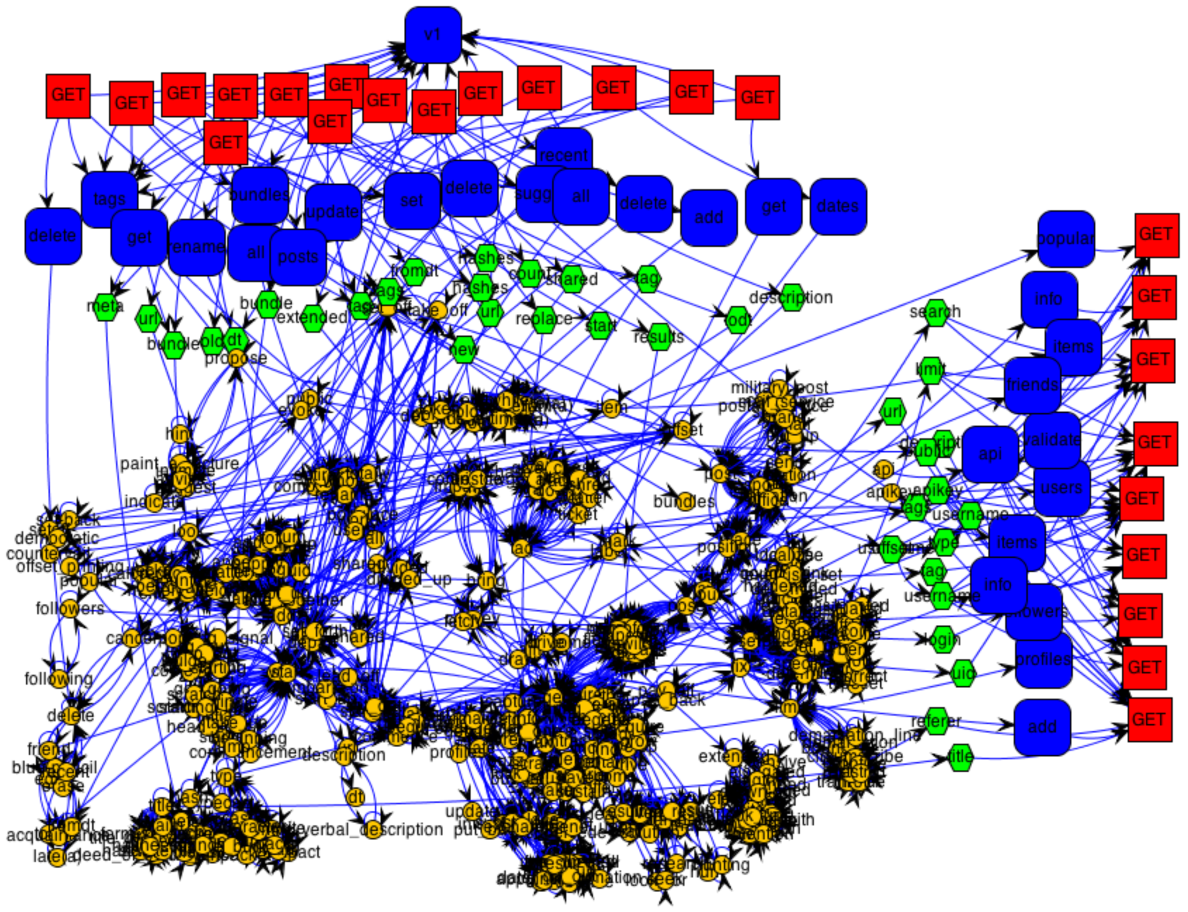


Figure 4. SFM network generated from Zootool and Del.icio.us WADL descriptions. Red rectangles represent the methods elements, blue rounded rectangles represent resource elements, green circles stands for param elements and orange circles show the term nodes.

the SFM method achieves the total precision of 88 percent with 25 percent standard deviation and the total recall of 80 percent with 28 percent standard deviation.

V. Summary and Conclusions

In this paper we proposed SFM, a novel graph-theoretic approach for matching similar methods of two given RESTful APIs based on their WADL descriptions. Our approach proposes a solution to a more general problem, web-service matching; which has many prospects in service discovery, composition and integration as well as many foreseeable applications for clients – sharing, migration, adaptation, *etc.* Reviewing earlier solutions to similar matching problems, schema/document/signature matching, we found that they could not naively be applied in our context. The SFM approach is designed specifically for REST API alignment, however the base semantic flow idea could be easily adopted to other semantic element matching problems. Yet there are several important aspects related to the proposed matching that require further discussion.

For the sake of simplicity, the current implementation of SFM approach does not include several sources of information – mainly the optional doc elements and response media-types. These should be included in the SFM network to obtain a more precise matching. However, even including all those related information is unlikely to produce a perfect result, as the textual descriptions of web services do not completely convey their underlying semantics [1]. For an effective and perfect matching we should also incorporate the behavioural knowledge (*i.e.* Business Process) in the protocol descriptions of the web services.

Another issue is that the term nodes are currently connected to each other based on their synonymy relations in WordNet. However the naming scheme of the service parameters depends on the developers [1] and is not necessarily consistent across, and possibly not even within, web services. It includes many different naming rules, hypernyms, abbreviations, *etc.* and general purpose lexical references such as WordNet are not suitable for our problem. For example consider our social-bookmarking example, in that context ‘bookmark’ and ‘tag’ are synonym, however they are

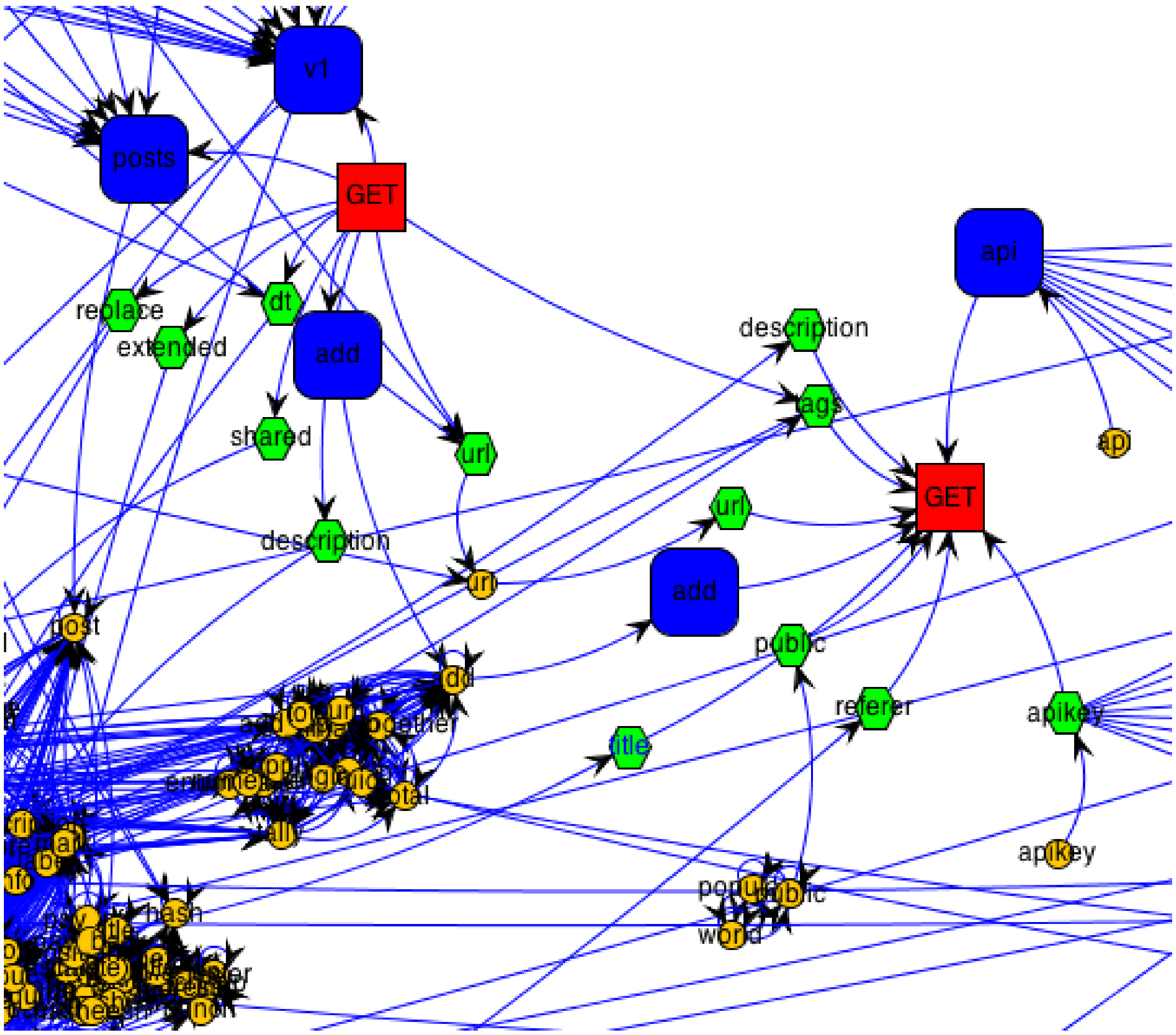


Figure 5. SFM network generated from Zootool and Del.icio.us WADL descriptions - closer look. We could see that how the upper ‘GET’ methods – which applies on ‘posts’ and ‘add’ resources and has ‘url’, ‘description’, ‘shared’, *etc.* named parameters – is semantically connected to the ‘GET’ method in the right – which corresponds to an ‘add’ resource and has similar parameters named ‘description’, ‘url’, ‘title’, ‘tags’, *etc.*. This semantic connection is mainly through the ‘add’, ‘url’ and ‘description’ term nodes which also includes other flows through synonym relations.

not synonymous in the general English sense, and are not listed as such in WordNet. One remedy to this problem could be to consider connections between term nodes that are close in other senses – composite elements or edit distance (‘tag’ $\rightarrow$ ‘mark’ $\rightarrow$ bookmark). Generally, for a precise matching we should have domain-specific semantic knowledge where terms are connected based on their meaning in the context of the problem.

The last point is that we need a more extensive evaluation for the SFM approach. The current data set is small, due to the difficulty of generating WADL files, required time for finding similar APIs and also the required manual modifications. Apart from the small scale of our data set, the selected APIs are very loosely coupled which makes the experiments on exact matching impossible. In other words, because there is a small overlap between functionalities of the selected APIs, we could not experiment with different types of matching algorithms, while there is at most one or two methods matched. Therefore we only considered the most highly matched methods as our results.

References

[1] X. Dong, A. Halevy, J. Madhavan, E. Nemes, and J. Zhang, “Similarity search for web services,” in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, ser. VLDB ’04. VLDB Endowment, 2004, pp. 372–383.

[2] R. Nayak and B. Lee, “Web service discovery with additional semantics and clustering,” in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelli-*

Table I
RESULT OF MATCHING ZOOTOOL AND DIIGO. THE UPPER PART SHOWS THE INPUT URIS TO THE SYSTEM AND THE BOTTOM PART IS THE SFM MATCHING RESULTS. THE SFM METHOD CORRECTLY MATCHED THE 'POST BOOKMARK' METHOD FROM DIIGO WITH THE 'GET ADD' METHOD FROM ZOOTOOL WITH HIGH CONFIDENCE OF 174. IT ALSO CORRECTLY MATCHED 'GET BOOKMARKS' FROM DIIGO WITH ITS ANALOG METHOD IN THE ZOOTOOL, 'GET ITEMS'.

https://secure.diigo.com/api/v2/bookmarks?user=joel&count=10&start=0&sort=1&tags=Spot,Film&filter=public&list=goodee
https://secure.diigo.com/api/v2/bookmarks?url=www.diigo.com&title=Diigo+-+Web+Highlighter+and+Sticky+Notes&tags=diigo,bookmark,highlight&shared=yes&desc=sample&readLater=no

http://zootool.com/api/users/items/?username=bastian&apikey=123&login=true&tag=po&offset=2&limit=3
http://zootool.com/api/users/info/?username=bastian&apikey=123
http://zootool.com/api/users/validate/?username=bastian&apikey=123
http://zootool.com/api/users/friends/?username=bastian&apikey=123&offset=2&limit=3&search=true
http://zootool.com/api/users/followers/?username=bastian&apikey=123&offset=2&limit=3&search=true
http://zootool.com/api/users/profiles/?username=bastian&apikey=123
http://zootool.com/api/items/info/?uid=iw6og3&apikey=123
http://zootool.com/api/items/popular/?type=week&apikey=123
http://zootool.com/api/add/?url=http://www.google.com&title=Google&apikey=123&tags=search,google&description=searchEngine&referer=http://www.bing.com&public=y

possible match with MAX-FLOW of: **174**
POST bookmarks: [title, shared, bookmarks, desc, v2, readLater, url, tags, api]
&
GET add:[public, apikey, description, title, api, referer, add, url, tags]

possible match with MAX-FLOW of: **76**
GET bookmarks: [user, sort, bookmarks, v2, count, list, filter, tags, api, start]
&
GET items:[apikey, tag, username, limit, api, offset, users, items, login]

gence, ser. WI '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 555–558.

[3] H. R. Motahari Nezhad, G. Y. Xu, and B. Benatallah, "Protocol-aware matching of web service interfaces for adapter development," in *Proceedings of the 19th international conference on World wide web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 731–740.

[4] R. Mikhaiel and E. Stroulia, "Examining usage protocols for service discovery," in *Service-Oriented Computing - ICSOC*, ser. Lecture Notes in Computer Science, A. Dan and W. Lamersdorf, Eds. Springer Berlin / Heidelberg, 2006, vol. 4294, pp. 496–502.

[5] Y. Hao and Y. Zhang, "Web services discovery based on schema matching," in *Proceedings of the thirtieth Australasian conference on Computer science - Volume 62*, ser. ACSC '07. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2007, pp. 107–113.

[6] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," in *Journal on Data Semantics IV*, ser. Lecture Notes in Computer Science, 2005, ch. 5, pp. 146–171.

[7] G. SALTON and C. BUCKLEY, "Global text matching for

Table II
MATCHING RESULTS OF THE SFM METHOD FOR ALL THE POSSIBLE COMBINATION OF COMPARING APIS. FOR EXAMPLE WE COULD SEE THAT THE SFM METHOD CORRECTLY MATCHED THE 'GET ADD' METHOD FROM DEL.ICIO.US WITH THE 'POST BOOKMARK' FORM DIIGO. SIMILARLY IT CORRECTLY MATCHED THE TWO ANALOG 'GET BOOKMARKS' METHODS IN DIIGO AND HBOOKMARK.

2) Del.icio.us and Zootool
possible match with MAX-FLOW of: 170.0
GET add: [shared, url, extended, replace, posts, v1, description, dt, tags, add]
&
GET add:[public, apikey, description, title, add, referer, url, tags, api]

3) Del.icio.us and Hbookmark
possible match with MAX-FLOW of: 100.0
GET set: [bundles, tags, bundle, v1, tags, set]
&
GET tags:[page, user, rpp, tags, api, cb]

4) Del.icio.us and Diigo
possible match with MAX-FLOW of: 70.0
GET add: [shared, v1, url, extended, replace, posts, description, dt, tags, add]
&
POST bookmarks:[bookmarks, title, shared, desc, readLater, v2, api, url, tags]

possible match with MAX-FLOW of: 28.0
GET recent: [v1, count, posts, recent, tag]
&
GET bookmarks:[user, bookmarks, sort, count, v2, list, api, filter, tags, start]

5) Diigo and Hbookmark
possible match with MAX-FLOW of: 214.0
GET bookmarks: [user, v2, sort, bookmarks, api, count, list, filter, tags, start]
&
GET bookmarks:[page, user, rpp, bookmarks, cb, api]

6) Zootool and Hbookmark
possible match with MAX-FLOW of: 90.0
GET add: [public, apikey, description, title, referer, api, url, tags, add]
&
GET tagged:[tag, page, user, rpp, tagged, api, cb]

possible match with MAX-FLOW of: 74.0
GET followers: [apikey, search, limit, followers, username, offset, api, users]
&
GET search:[q, page, user, rpp, search, api, cb]

possible match with MAX-FLOW of: 74.0
GET items: [apikey, tag, username, limit, offset, items, api, users, login]
&
GET tags:[page, user, tags, rpp, api, cb]

Table III
EVALUATION OF THE SFM METHOD. THE PRECISION AND RECALL OF THE SFM METHOD IS REPORTED FOR EVERY PAIRS OF APIS CONCLUDING WITH AN AVERAGE/TOTAL PRECISION AND RECALL AT THE LAST ROW.

APIs	precision	recall
Diigo and Zootool	1	1
Del.icio.us and Zootool	1	.33
Del.icio.us and Hbookmark	1	.5
Del.icio.us and Diigo	1	1
Diigo and Hbookmark	1	1
Zootool and Hbookmark	.33	1
TOTAL	.88±.25	.80±.28

information retrieval," *Science*, vol. 253, no. 5023, pp. 1012–1015, 1991.

[8] C. Pahl, "An ontology for software component matching," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, pp. 169–178, March 2007. [Online]. Available: http://portal.acm.org/citation.cfm?id=1230649.1230650

[9] E. Stroulia and Y. Wang, "Structural and semantic matching for assessing web-service similarity," *International Journal of Cooperative Information Systems*, vol. 14, pp. 407–437, 2005.

[10] M. Fokaefs, R. Mikhaiel, N. Tsantalis, and E. Stroulia, "An empirical study on web service evolution," in *Proceedings of the IEEE International Conference on Web Services 2011 (ICWS 2011)*, Washington, DC, USA, October 2011.

[11] D. Kehagias, A. G. Castro, D. Tzovaras, and D. Giakoumis, "A semantic web service alignment tool." in *International Semantic Web Conference*, 2008.

[12] A. He and N. Kushmerick, "Learning to attach semantic metadata to web services," in *The Semantic Web - ISWC 2003*, ser. Lecture Notes in Computer Science, D. Fensel, K. Sycara, and J. Mylopoulos, Eds. Springer Berlin / Heidelberg, 2003, vol. 2870, pp. 258–273.

[13] W. Abramowicz, K. Haniewicz, M. Kaczmarek, and D. Zyskowski, "Architecture for web services filtering and clustering," in *Internet and Web Applications and Services (ICIW)*, May 2007, p. 18.

[14] Y. Wang and E. Stroulia, "Flexible interface matching for web-service discovery," in *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, ser. WISE '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 147–.

[15] J. Kopecký, K. Gomadam, and T. Vitvar, "hrests: An html microformat for describing restful web services," in *Web Intelligence*. IEEE, 2008, pp. 619–625.

[16] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. "big"' web services: making the right architectural decision," in *WWW*, J. Huai, R. Chen, H.-W. Hon, Y. Liu, W.-Y. Ma, A. Tomkins, and X. Zhang, Eds. ACM, 2008, pp. 805–814.

[17] J. Lathem, K. Gomadam, and A. P. Sheth, "Sa-rest and (s)mashups : Adding semantics to restful services," in *ICSC*. IEEE Computer Society, 2007, pp. 469–476.

[18] Wikipedia, "Representational state transfer — Wikipedia, the free encyclopedia," 2010. [Online]. Available: \url{http://en.wikipedia.org/wiki/Representational_State_Transfer}

[19] ——, "Web service — Wikipedia, the free encyclopedia," 2010. [Online]. Available: \url{http://en.wikipedia.org/wiki/Web_service}

[20] ——, "Web application description language — Wikipedia, the free encyclopedia," 2010. [Online]. Available: \url{http://en.wikipedia.org/wiki/Web_Application_Description_Language}

[21] M. Hadley, "Introducing wadl," http://weblogs.java.net/blog/mhadley/archive/2005/05/introducing_wad.html, May 2005.

[22] T. Steiner, "Automatic multi language program library generation for rest apis," Master's thesis, University of Karlsruhe / cole Nationale Suprieure d'Informatique et de Mathmatiques Appliques de Grenoble, July 2007.

[23] M. F. Porter, "An Algorithm for Suffix Stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[24] C. Fellbaum, Ed., *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*, illustrated edition ed. The MIT Press, May 1998.

[25] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, pp. 248–264, April 1972.

[26] J. O'Madadhain, D. Fisher, S. White, and Y. Boey, "Automatic multi language program library generation for rest apis," UCI-ICS, Tech. Rep., Oct. 2003. [Online]. Available: http://www.datalab.uci.edu/papers/JUNG_tech_report.html

[27] "Del.icio.us api," http://www.delicious.com/help/api.

[28] "Diigo api," http://www.diigo.com/tools/api.

[29] "Hbookmark api," http://groups.google.com/group/hbookmark/.

[30] "Zootool api," http://zootool.com/api/docs/general.