Socially-Enriched Semantic Mashup of Web APIs

Jooik Jung and Kyong-Ho Lee

Department of Computer Science
Yonsei University
Seoul, Republic of Korea
jijung@icl.yonsei.ac.kr, khlee@cs.yonsei.ac.kr

Abstract. As Web mashups are becoming one of the salient tools for providing composite services that satisfy users' requests, there have been many endeavors to enhance the process of recommending the most adequate mashup to users. However, previous approaches show numerous pitfalls such as the problem of cold-start, and the lack of utilization of social information as well as functional properties of Web APIs and mashups. All these factors undoubtedly hinder the proliferation of mashup users as locating the most appropriate mashup becomes a cumbersome task. In order to resolve the issues, we propose an efficient method of recommending mashups based on the functional and social features of Web APIs. Specifically, the proposed method utilizes the social and functional relationships among Web APIs to produce and recommend the chains of candidate mashups. Experimental results with a real world data set show a precision of 86.9% and a recall of 75.2% on average, which validates that the proposed method performs more efficiently for various kinds of user requests as compared to a previous work.

Keywords: Web api, mashup recommendation, functional semantics, social relationship.

1 Introduction

In the past few years, Web mashups have attracted tremendous interest from both service developers and end-users. These applications exhibit the ability to combine existing service functionalities with a minimal development effort and thus making them a powerful tool for providing composite services that satisfy users' requests [5]. However, the explosive growth of Web APIs (hereafter, when we use the term "API", it refers to a "Web API") raises challenging problems of how to enforce the adequacy of the mashups and the ways to accelerate the discovery of the component APIs. Moreover, current mashup composition methods manually search and select the component services and thus aggravating the overall mashup generation process [1]. Hence, many of the researches have tried to exploit various social or functional features of APIs as a solution to the aforementioned issues. Despite the effort, most of the contemporary approaches utilize these social and semantic features separately or exhibit the problem of cold-start where APIs that have no history of being selected for mashup composition are never selected in future compositions.

C. Liu et al. (Eds.): ICSOC 2012, LNCS 7636, pp. 389–403, 2012.

In this paper, we propose a novel technique for recommending mashups from natural language requests as well as exploiting both functional and social features of Web APIs and their corresponding ontologies in the process. To elaborate, we first present a systematic approach for extracting functional semantic descriptors from a user request, which are required to facilitate the discovery and composition processes of APIs. We then represent the functional and social features of candidate APIs with graph-based network models. Finally, the assessments of the candidate mashup chains are computed for the purpose of recommendation. As for the social features, we exploit the popularity, collaboration and ratings of APIs to augment the social richness of the proposed method. Furthermore, the executability of a candidate mashup is computed by exploiting the connectivity between the input/output parameters of the participating APIs.

To evaluate the performance of the proposed approach, 20 different natural language requests, each with varying complexity, were used. The experimental results showed a precision of 86.9% and a recall of 75.2% on average. Particularly, it is worth mentioning that as the complexity of a natural language query increased, the precision of the proposed algorithm on that specific request also depicted an increase.

The remaining sections are organized as follows. In Section 2, we present a brief survey of related work. Section 3 describes, in detail, the proposed hybrid method for recommending Web API mashups. The results and analysis of our experimentation are presented in Section 4. We conclude this paper and discuss our plans for future work in Section 5.

2 Related Work

Given the proliferation of Web-based services like Web APIs, there have been many researches on how to compose them efficiently and accurately. The following papers discuss various approaches for discovering and composing APIs, and recommending the resulting mashups.

In [2, 3] a keyword-based search approach which integrates social information is proposed for the purpose of selecting mashup components. First, the authors build an API functional taxonomy, which is used to locate the APIs that match the desired functionalities, using the descriptions of APIs. The description-based technique is enhanced by combining social ranking measures to rank each API. However, the method neglects the functional features of APIs such as their input/output parameters and thus the executability of the resulting mashup is not guaranteed.

The authors in [4] propose a mechanism to specify the functional semantics of Web services based on action and data ontologies. Composite Web services are represented by a graph which describes the relations among the component services in terms of input and output parameters and their functional semantics. We concur with this approach of assigning each Web service, or API in our case, with its corresponding functional semantics to accelerate the service discovery process. However, this particular work lacks the utilization of social information which has the potential to enhance the mashup formation process.

In [21] a method which combines semantics and collective knowledge to assign component descriptors to each Web API is introduced. The author states that this hybrid technique ultimately accelerates the speed of API selection process by manipulating these component descriptors. Here, the technique does not exploit any of the past historical information of APIs.

The majority of works in the area of mashup composition have utilized the tags of mashups and APIs for the purpose of recommendation. In [6] the authors propose a social technique to mine the tags of mashups and APIs for recommendation purposes. However, there is a pitfall to this approach since API developers do not necessarily reuse the same tags to describe APIs. In [7, 8] tag-based clustering approaches are proposed for computing the similarity between tag clouds, where the services corresponding to a specific tag are grouped together. In these researches, the usefulness of mining tags in discovering candidate APIs cannot be judged due to insufficient experiments.

In [9] a faceted classification of Web APIs and an algorithm which ranks those APIs are proposed. By using this approach, the authors argue that the API retrieval process can be improved. Although the technique provides a coarse-grain mechanism for API discovery, the semantic descriptions of APIs are not taken into consideration.

Some of the works [10, 11, 12, 13] are launched to exploit the social networks of mashup developers for constructing mashups. To improve the composition process, the authors in [10] suggest that developers should consider the social networks or collaborative environments of users. Some of the information extractable from social networks are users' past experiences [11, 13] related to the services that they have used. By exploiting the social networks and this "extra" information, the authors in [12] propose that the recommendation of component services is possible from the perspectives of mashup developers.

We strongly believe that the exploitation of social networks has the potential to impact the discovery and composition process of Web APIs and thus we aim to integrate this feature with our approach. ProgrammableWeb[1] is a popular online repository of APIs and mashups [14, 15]. In our work, we utilize this repository for building our data set, which are to be exploited for discovering APIs and constructing their corresponding social and connectivity graphs.

3 The Proposed Mashup Recommendation Algorithm

In this section, we present the proposed mechanism to combine the Web API discovery via functional semantics and the corresponding mashup chain composition based on the social elements and input/output connectivity of candidate APIs. The mechanism consists of the following four major phases: (3.1) extraction phase, (3.2) discovery phase, (3.3) chaining phase, and (3.4) selection phase. Furthermore, the chaining phase is divided into two sub-phases: chaining based on input/output connectivity graph and chaining based on social graph, and the selection phase is also composed of two sub-phases: connectivity analysis and social analysis. The general overview of our approach is illustrated in Figure 1.

[1] http://www.programmableweb.com/

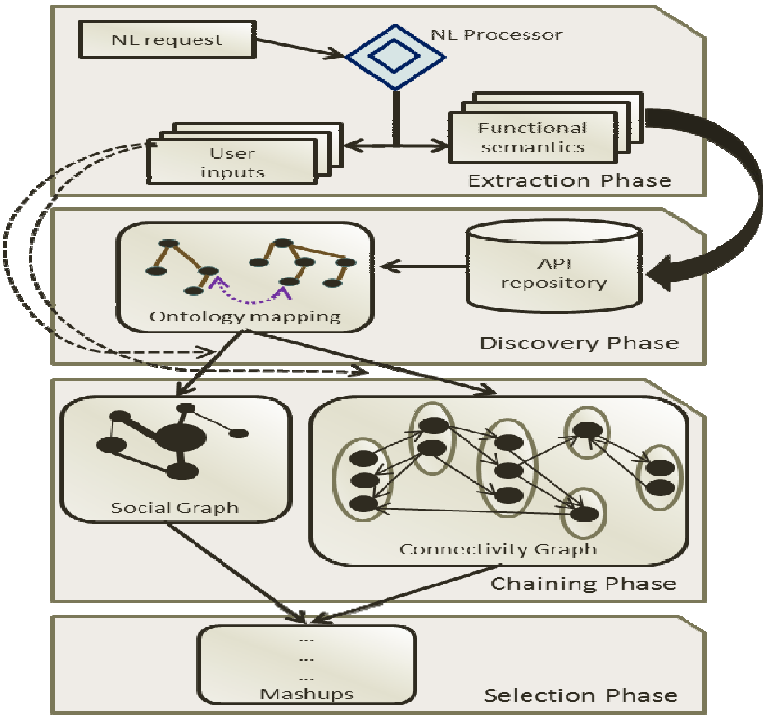

Fig. 1. Overview of the proposed approach

3.1 Extraction Phase

In the extraction phase, we gather the necessary information, namely the functional semantics and user inputs of the requested mashup operations from a user request, which are to be utilized in the next phase. In order to demonstrate this process, we begin by extracting the functional semantics and user inputs from a natural language request as shown in Figure 2. We have particularly selected our input language to be in the form of a natural language request from all other possible choices, as it would be sufficient for ontology mapping.

Before digging into the details of the proposed method, we first define the terms: functional semantics and user inputs. Functional semantics consist of two components, action and object. These two elements combined describe the kinds of services that a particular mashup service offers. As an example, for a functional semantic pair {*rent, movie*}, *rent* and *movie* correspond to the action and object components respectively. User inputs represent various input parameters exploitable by the operations of the mashup service. Unlike the Web services discussed in [4], the current Web APIs do not provide their functional semantics explicitly, and the descriptions of their operations and input/output parameters do not follow any form of a rigid guideline. Thus, it is worth noting that our dataset of Web APIs and proposed ontologies are constructed manually by our team through analysis of APIs available from the ProgrammableWeb directory.

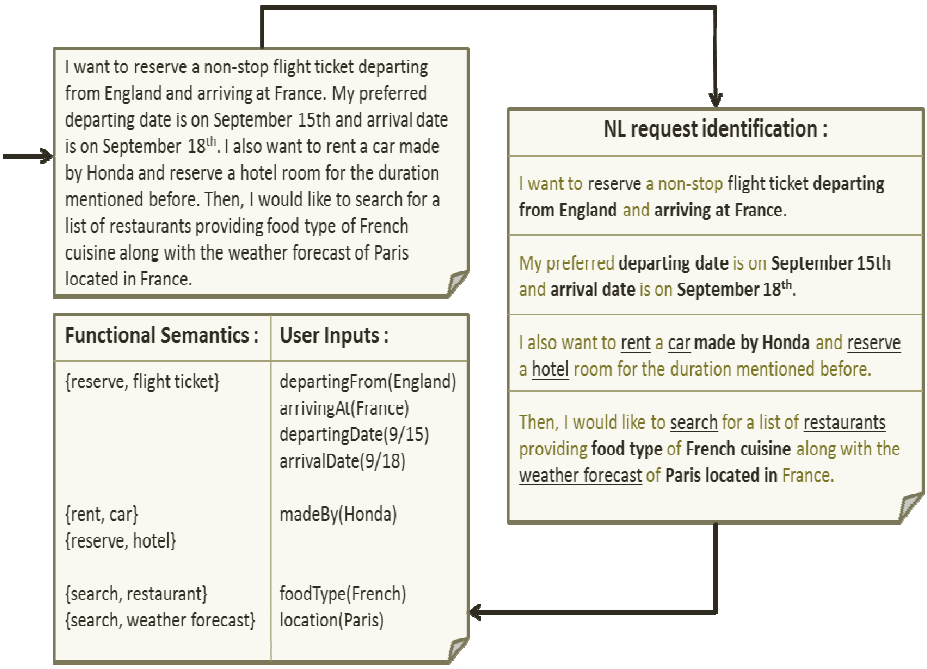

Fig. 2. An example of a natural language request and the proposed extraction method

To initiate the extraction phase, the proposed method divides a natural language request into sentence blocks similar to the work of Lim and Lee [16]. Each of these sentence blocks are then processed by a natural language processor known as the RASP system [17]. Specifically, a natural language query is checked for main verbs, which may represent the action component of the functional semantics to be extracted, and nouns, which may describe the object component of the functional semantic and possibly user inputs. The diagram on the right hand side of Figure 2 shows the result from applying a natural language processor to the natural language description illustrated on the top left hand side of the figure. The action and object components of the functional semantics are indicated by underlined texts, and the user inputs are depicted by bolded texts.

In order to finalize the extraction process, the identified verbs and nouns are extracted and categorized into functional semantics and user inputs. Main verbs and nouns are classified as functional semantic pairs whereas auxiliary verbs and nouns are categorized as user inputs. The final outcome of the extraction phase is illustrated in the bottom left hand side of Figure 2.

3.2 Discovery Phase

Once functional semantic pairs have been extracted and user inputs have been obtained, the discovery phase initiates. The purpose of this phase is to locate all APIs whose functional semantics conform to the functional semantics of a user's natural

language request. It is worth mentioning that previous researches have assigned a single pair of functional semantics to each Web API. This may yield inaccurate data due to the imperfection of the object and action ontologies and the functional semantic pair selection algorithm. Thus, we have assigned a functional semantic pair to each service operation of a Web API and allocated the union of those pairs to the corresponding API to discover additional APIs that may have been ignored otherwise. Once this preparation ends, we commence the API discovery process by searching the repository of APIs where the functional semantics of APIs are advertised along with the ontologies that our team has constructed and reorganized. By doing so, we accelerate the API discovery process tremendously as we mitigate the need for an entire API database search for selecting candidate APIs.

To elaborate, we first scan the API repository using the functional semantic pairs that we have acquired from the extraction phase. If there is a match between APIs' advertised functional semantics and the functional semantic pairs from a user's natural language query, the corresponding API gets flagged as a candidate API. Here, we propose two types of matches: exact match and partial match. An exact match is where any two functional semantic pairs have the same action and object components. For this particular case, we can save much of the database access time as the inference via object and action ontologies are not necessary for selecting candidate APIs. On the other hand, if two concepts are not an exact match, their relationship must be inferred. In this case, therefore, the action or object ontologies must be accessed for further verification.

For the partial match cases, (1) is revised from the work of Li et al. [18] to compute a similarity value between two ontology concepts. The equation involves two key elements, the height of the matching parent, denoted as *parHeight* and distance to the parent, denoted as *d*, which determine the similarity between two different ontology concepts. The rationale behind this equation is that the similarity between two ontology concepts must increase as the height of their intersecting parent concept increases and as the distance to that parent from two concepts decreases.

$$ConceptSim(C_1, C_2) = \frac{parHeight(C_1, C_2)}{(d(C_1) + d(C_2)) / 2} \quad (1)$$

where $parHeight(C_1, C_2)$ = *height of matching parent concept of concept* C_1 *and* C_2 ,
$d(C_n)$ = *concept* C_n*'s distance to the matching parent concept*

Here, we describe the process of mapping the concepts from object and action ontologies to the object and action components of the functional semantic pairs extracted from the previous phase. This process is a revised version of the work of Klusch et al. [19]. As mentioned earlier, this mapping process is targeted for the candidates which fall into the partial match category as these APIs may still be selected if their values from (1) are above a threshold. Continuing with our example of the natural language request described previously, we have picked {*reserve, hotel*} for a demonstration. Figure 3 illustrates the action and object ontologies for a Web API named TourCMS[2]

[2] http://www.tourcms.com/

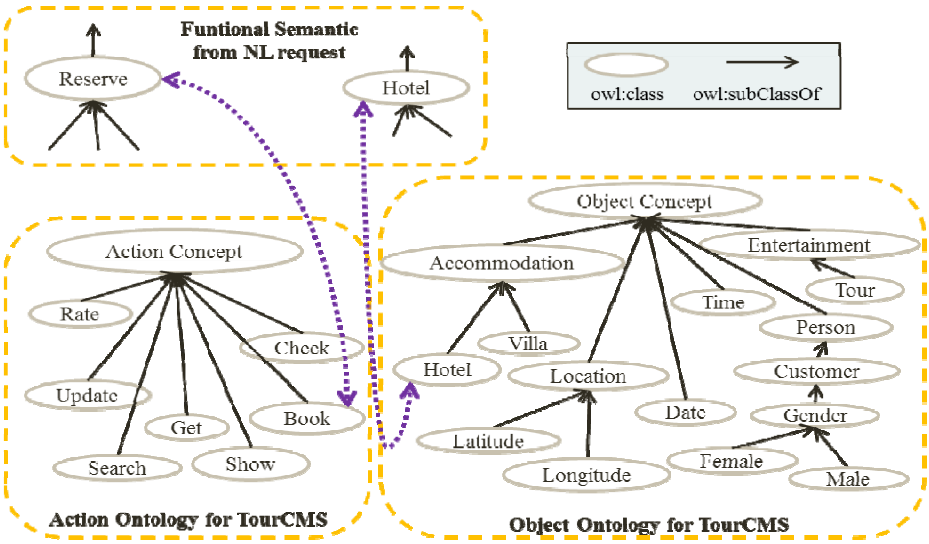

Fig. 3. An example of mapping concepts using action/object ontologies

and how the functional semantic pair, {*reserve, hotel*}, gets mapped to the corresponding ontology concepts in the case of a partial match. It is also worth noting that the threshold is adjustable by the users of this system.

3.3 Chaining Phase

The next step involves the composition of the candidate APIs obtained from the previous phase to yield mashup chains [20]. This mashup chain construction process depends on the proposed I/O connectivity graphs and social graphs. In addition, the user inputs and the functional semantic pairs from the extraction phase are utilized for further pruning.

Chaining Based on Connectivity Graph

Here, we exploit the user inputs and the functional semantic pairs obtained from the extraction phase to construct an I/O connectivity graph. An I/O connectivity graph is a graph depicting the collaboration relationships between various APIs at the operation level. In other words, this graph takes into account that APIs are in fact bridged by the connectivity of their operations. The construction of the graph is based on the mappings of the candidate APIs' input/output parameters within the I/O ontology. Moreover, Figure 4 shows a simplified version of the I/O connectivity graph containing arbitrarily generated APIs for illustration purposes. In this figure, the enclosing nodes and the enclosed nodes represent the APIs and their operations respectively. Also, the direction of the edges between two enclosed nodes dictates the dataflow. That is, the operation where the arrow points to can at least take one output of the

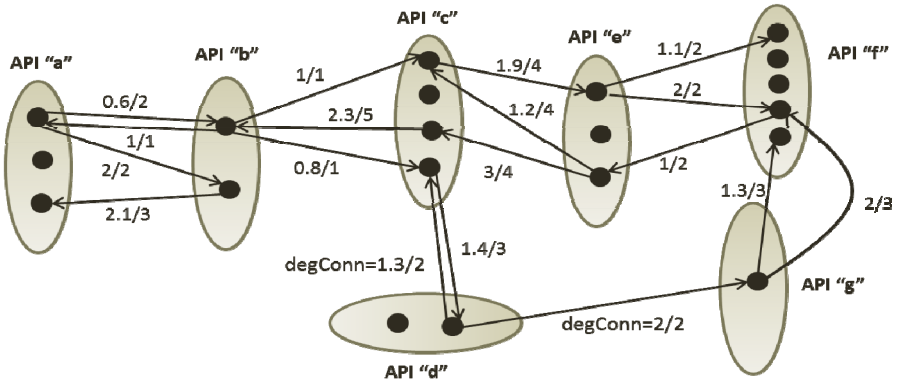

Fig. 4. A simplified version of the proposed I/O connectivity graph

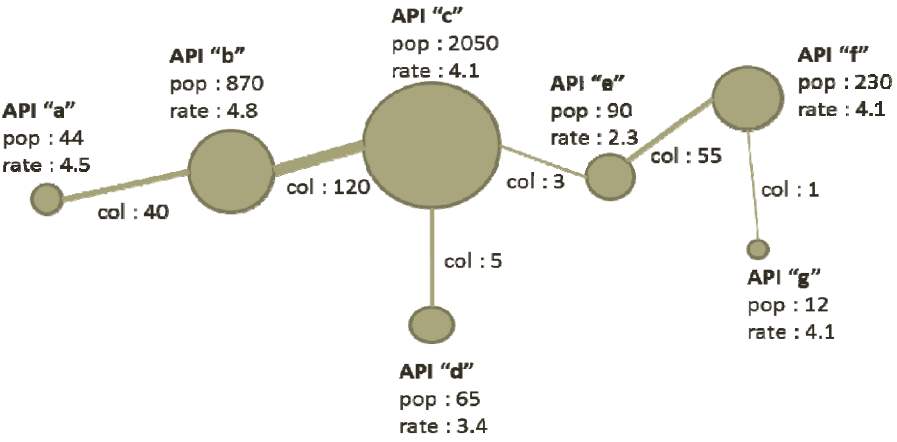

Fig. 5. A simplified social graph version of Figure 4

preceding operation as its input. The edge weight representing the degree of connectivity, denoted as *degConn*, then becomes the following:

$$degConn(u \rightarrow v) = \frac{|ExactMatch| \; + \; \alpha \cdot |PartialMatch|}{total\ number\ of\ v's\ inputs} \tag{2}$$

where $degConn(u \rightarrow v)$ = *degree of connectivity from operation u to v,*
$|ExactMatch|$ = *number of concepts matching exactly,*
$|PartialMatch|$ = *number of concepts matching partially,*
α = *weight for partial match,*
$0 \leq degConn(u \rightarrow v) \leq 1$

This equation is essentially the ratio of the total number of inputs of the target operation that are provided by the preceding operation to the total number of inputs of the target operation. Furthermore, it is worth mentioning that the value of the weight for

partial match can be selected by the users to alter the degree of emphasis on partial match cases.

Once the I/O connectivity graph illustrating the candidate mashup chains is obtained, we then try to reduce the search space even further. For this purpose, we deploy two pruning methods which facilitate the successive reduction of the search space: pruning via functional semantics and pruning via user inputs. First, we reduce the search space by pruning the mashup chains whose combined functional semantics do not contain all of the functional semantics extracted from a natural language description. This is due to the fact that any of the candidate chains may be connected simply by their input/output parameters. If so, the resulting candidate mashup chains may be inadequate for satisfying the user request and are removed as a consequence. After that, we execute another pruning method which exploits the user inputs. That is, we reduce the search space further by pruning the mashup chains whose APIs' operations do not utilize all of the user inputs as their operation parameters. In other words, the operations of the APIs composing the mashup chains must take all of the user inputs as their input parameters.

Chaining Based on Social Graph

Once an I/O connectivity graph has been revised to yield various mashup chains satisfying a user request, a social graph for these finalized candidate mashup chains is constructed. A social graph is a graph portraying the relationships between APIs by capturing various social elements such as the rating and popularity of an API, and the collaboration of two APIs. Figure 5 illustrates a simplified version of the social graph for the APIs contained in the I/O graph generated from the previous section. This graph is similar to the API collaboration network proposed in the work of Tapia et al. [2], with the addition of API ratings to enhance the social richness of our approach. In this figure, nodes and edges represent APIs and their collaborations in existing mashups respectively. In addition, the popularity, denoted as *pop*, is the number of times that a particular API is used in the formation of mashups. It is important to note that the size of a node is dependent on its popularity value. The collaboration, denoted as *col*, describes the number of times that two adjacent APIs are used concurrently in the composition of mashups. This factor also indicates how thick the edges should be in a social graph. Lastly, the rating, denoted as *rate*, is the averaged user rating value of a particular API.

3.4 Selection Phase

In this section, we describe how an I/O connectivity graph and a social graph are exploited in recommending the finalized candidate mashup chains satisfying a user request.

Connectivity Analysis

For every I/O connectivity sub-graph corresponding to each of the finalized candidate mashups, we calculate the connectivity rank using the degree of connectivity values. Recall that the degree of connectivity, denoted as *degConn*, represents the ratio of how many of the required inputs of a particular operation are satisfied by its preceding

operation in their collaboration. To elaborate, the total degree of connectivity for a particular mashup signifies the executability of that mashup. By taking advantage of I/O connectivity of APIs, the newly created APIs are given an opportunity to be utilized and hence alleviating the cold-start problem. The following formula describes the connectivity rank calculation for a candidate mashup chain:

$$ConRank(I) = \frac{\sum_{u,v}^{q} degConn(u \rightarrow v)}{q} \tag{3}$$

where ConRank(I) = connectivity rank for mashup chain I,
degConn(u→v) = degree of connectivity from operation u to v,
q = total number of connectivity relationships in the mashup chain,
$0 \leq ConRank(I) \leq 1$

Social Analysis

Once all of the candidate mashup chains have been assigned with their corresponding connectivity rank values, we then evaluate the social ranks of those chains. As mentioned above, we have three social factors in our social graph namely the popularity, collaboration, and user rating, denoted as *pop*, *col* and *rate* respectively. To calculate the social rank values of a candidate mashup, we use (4), which is based on the fact that the popularity of a single Web API is greater than or equal to the total collaboration of that API. Thus, the first segment of the equation computes how many of the existing collaboration relationships are remaining in the newly constructed social graph. As this number increases, the candidate mashup chain is assigned a higher social rank value. Moreover, the second segment of (4) is simply normalizing the ratings of the participating APIs.

$$SocRank(I) = \frac{\sum col(i)}{\sum pop(i)} \cdot \frac{\sum rate(i)}{MaxRating \cdot |i|} \tag{4}$$

where SocRank(I) = social rank for mashup chain I,
col(i) = collaboration for API i,
pop(i) = popularity for API i,
rate(i) = rating for API i,
MaxRating = 5,
$0 \leq SocRank(I) \leq 1.$

Once the connectivity and social ranks for every candidate mashup chain have been obtained, we then determine the final assessment values of those chains. Here, we refer to the final assessment as the recommendation assessment, denoted by *recAssess*. The computation of *recAssess* value utilizes both the social rank and connectivity rank as described in (5).

$$recAssess(I) = \beta \cdot SocRank(I) + (1 - \beta) \cdot ConRank(I) \tag{5}$$

where recAssess(I) = recommendation assessment of mashup chain I,
β *= weight,*
$0 \leq recAssess(I) \leq 1$

Table 1. A Portion of the Natural Language Queries along with their Extracted Functional Semantics and User Inputs

	Request17	***Request10***	***Request8***	***Request3***
Extracted functional semantics	{reserve,flight ticket} {rent,car} {reserve,hotel} {search,restaurant} {search,weather}	{show,house} {search,person} {inform,time} {compute,radius}	{search,menu} {search,price} {inform,friend} {view,photo}	{search,car} {rent,car} {show,map}
Extracted user inputs	departFrom(England) arriveAt(France) departDate(9/15) arriveDate(9/18) madeBy(Honda) foodType(French) location(Paris)	city(Vancouver) firstName(Sam) lastName(Lee) longitude(123,06) latitude(49,13)	foodType(Japanese) restaurantName(Guu) city(London)	minPrice(0) maxPrice(8500) carType(SUV)

4 Experimental Results

In order to evaluate the performance of our approach, we have manually built a Web API repository based on a well-known real world API database, ProgrammableWeb.com. To elaborate, we have extracted and parsed 614 APIs along with their data such as name, popularity, rating, description, and category. Then, we have analyzed APIs' description manuals available on the Web to obtain all of their operations and input/output parameters. As a result, we ended up with 777 operations and 7128 input/output parameters. Thereafter, we have extracted the functional semantics from each operation and also constructed action/object/input/output ontologies for those APIs using Protege[3], an open-source ontology editor.

Once our API database was established, 20 different natural language requests with varying complexity were manually created for test purposes. Table 1 illustrates a small portion of the natural language queries and the outcomes obtained from the extraction phase. Moreover, a connectivity graph and a social graph were constructed for each request, and various factors required for the computation of recommendation assessment values were also calculated. Finally, the candidate mashup chains with higher recommendation assessment values than the optimal threshold were selected for recommendation.

To verify whether a candidate mashup chain is a valid one or not, we had our teams of mashup experts compose mashup services for a comparison with the automatically generated candidate mashup chains. If a particular mashup service executes successfully and produces a sound outcome as expected by a given request, then that mashup chain is considered to be valid.

As for the precisions of the resulting mashups from all 20 requests, the outcome ranged from minimum of 77.8% precision to maximum of 93.6% precision. In addition, the values of recall ranged from 72.5% to 87.5%. Figure 6 illustrates the precision and recall values of final mashups recommended from all of the natural language requests. Particularly, the natural language query described earlier in the paper,

[3] `http://protege.stanford.edu/`

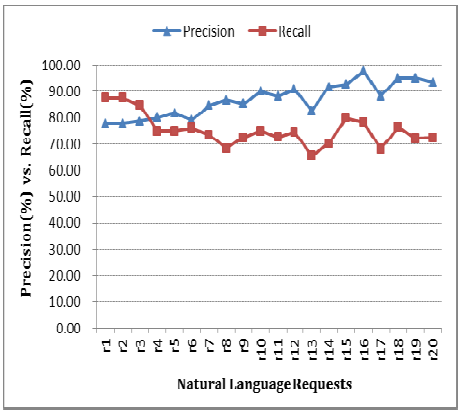

Fig. 6. A precision vs. recall graph of the proposed approach

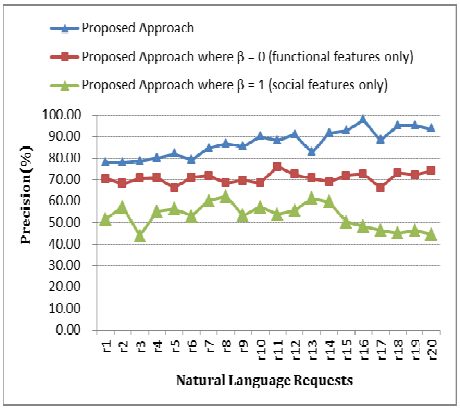

Fig. 7. A precision graph for three different versions of the proposed method

request17 from Table 1, yielded a precision of 86.7% and a recall of 68.4%. Overall, the experimental results depicted an average precision of 86.9% and an average recall of 75.2%.

In addition, Figure 7 depicts three different precision graphs of the resulting mashups obtained using different versions of our proposed method. The first line from the bottom represents the precision of the recommended mashups where β, from Equation (5), was set to be 1. By assigning so, the proposed method only considered the social features of APIs and ignored the functional features such as the I/O connectivity of API operations. This particular version of the proposed method exhibited an average precision of 53.1% which was significantly lower than the proposed method which exploited both the social and functional features of APIs.

On the other hand, the middle line in Figure 7 illustrates the precision of the recommended mashups, where β, from Equation (5), was set to be 0. This particular graph represents the version of our proposed method which only utilizes the functional features of APIs. For this approach, the precision was observed to be 70.7% on average. This value was still lower than the precision of the proposed approach which integrates both the functional and social features of Web APIs. Also, we have included, at the top, the precision graph of the original version of our method from Figure 6 in Figure 7 to facilitate easy comparison. It is worth mentioning that for this particular graph line, we have used 0.41, which was found to exhibit relatively high precision after several experiments, for the value of β.

In comparison with the experimental results from the work of Gomadam et al. [9], our algorithm exhibited a significant improvement in performance. As for the experimental setup, Gomadam et al. have also exploited the Web APIs from ProgrammableWeb.com and tested with 5 different user queries. In their work, the average precision was found to be around 77% and the average recall was approximately 70%. These numbers indicate that our approach has shown 9.9% and 5.2% improvements in the precision and recall respectively.

Nevertheless, the proposed algorithm yielded 13.1% of erroneous results on average for those 20 natural language queries. From all of the recommended mashups,

13.1% of them were either not executable or did not satisfy the user requests due to the following reasons. First, due to the fact that ontologies are human-constructed, they may contain insufficient information required for concept mapping and inference engines. For example, a functional semantic pair, {*rent, Mustang*}, extracted from a user request may not be able to infer that Mustang is a car after ontology concept mapping. Consequently, an incorrect service was discovered and regarded as a candidate API service.

Second, an error in the extraction of functional semantic pairs and user inputs from the test queries was unavoidable. To elaborate, the proposed algorithm accepts a user query that is composed in natural language. By forming the queries in natural language, mashup users are able to diminish the formality of query generation compared to other querying techniques. However, as these users gain more expressive power when constructing queries, it becomes harder for a natural language processor to capture all of their intentions. As a consequence, the extracted functional semantic pairs and user inputs may vary from what the users expect. In this case, the proposed approach selected an incorrect service which is not consistent with the intention of our test queries.

5 Conclusions and Future Work

In this paper, we have presented a hybrid approach which combines both the social and functional features of APIs to enhance the API discovery and composition processes. Moreover, we have introduced new algorithms for computing the rankings of the resulting candidate mashup chains to assist mashup users. We have first extracted the functional semantic pairs and user inputs from a natural language request. Then, we have selected the corresponding Web APIs that match the extracted functional semantic pairs. After that, a connectivity graph and a social graph between the candidate APIs have been constructed based on ontology mapping. Finally, the candidate mashup chains have been examined in order to recommend the ones that are adequate for satisfying a user's request.

Overall, the proposed algorithm performed efficiently for a number of different natural language queries, each with varying complexity. The experimental results showed an average precision of 86.9% and an average recall of 75.2%, which implies a significant improvement from a previous work. In addition, our experimental results demonstrated that the combination of social and functional features exhibited a significantly better precision than these exploited separately.

Although our current database of Web APIs contains a sufficient amount of data to conduct valid experiments, we believe that there is a necessity for an exhaustive experiment with a more large volume of APIs. In addition, we are looking to adjust or possibly move away from formatting our input in natural language form as this particular technique exhibits a fair amount of noise compared to other query techniques. So, we will be researching on other various query techniques to enhance the quality of our work. Furthermore, we strongly believe that the exploitation of wisdom of crowds through crowdsourcing techniques is an area that has the potential to enhance the API composition process as stated in Section 2. Therefore, we will also be investigating different ways to leverage on this aspect for its integration with our approach.

Acknowledgment. The research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2011-0026423).

References

1. Elmeleegy, H., Ivan, A., Akkiraju, R., Goodwin, R.: Mashup advisor: a recommendation tool for Mashup development. In: IEEE International Conference on Web Services, ICWS, pp. 337–344 (2008)
2. Tapia, B., Torres, R., Astudillo, H.: Simplifying mahsup component selection with a combined similarity- and social-based technique. In: 5th International Workshop on Web APIs and Service Mashups, MASHUPS (2011)
3. Torres, R., Tapia, B., Astudillo, H.: Improving Web API Discovery by leveraging social information. In: IEEE International Conference on Web Services, ICWS, pp. 744–745 (2011)
4. Shin, D.H., Lee, K.-H., Suda, T.: Automated generation of composite web services based on functional semantics. Journal of Web Semantics 7(4), 332–343 (2009)
5. Maximilien, E.M., Wilkinson, H., Desai, N., Tai, S.: A Domain-Specific Language for Web APIs and Services Mashups. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 13–26. Springer, Heidelberg (2007)
6. Goarany, K., Kulczycki, G., Blake, M.B.: Mining social tags to predict mashup patterns. In: 2nd International Workshop on Search and Mining User-Generated Contents, SMUC, pp. 71–78 (2010)
7. Bouillet, E., Feblowitz, M., Feng, H., Liu, Z., Ranganathan, A., Riabov, A.: A folksonomy-based model of web services for discovery and automatic composition. In: IEEE International Conference on Services Computing, SCC, pp. 389–396 (2008)
8. Fernandez, A., Hayes, C., Loutas, N., Peristeras, V., Polleres, A., Tarabanis, K.: Closing the Service Discovery Gap by Collaborative Tagging and Clustering Techniques. In: 7th International Semantic Web Conference, ISWC, pp. 115–128 (2008)
9. Gomadam, K., Ranabahu, A., Nagarajan, M., Sheth, A.P., Verma, K.: A faceted classification based approach to search and rank web apis. In: IEEE International Conference on Web Services, ICWS, pp. 177–184 (2008)
10. Schall, D., Truong, H.L., Dustdar, S.: Unifying Human and Software Serivces in Web-Scale Collaborations. IEEE Internet Computing 12(3), 62–68 (2008)
11. Maaradji, A., Hacid, H., Daigremont, J.: Towards a Social Network Based Approach for Services Composition. In: IEEE International Conference on Communications, ICC, pp. 1–5 (2010)
12. Maaradji, A., Hacid, H., Skraba, R.: Social Web Mashups Full Completion via Frequent Sequence Mining. In: IEEE World Congress on Services, SERVICES, pp. 9–16 (2011)
13. Maaradji, A., Hacid, H., Skraba, R., Lateef, A., Daigremont, J., Crespi, N.: Social-based Web services discovery and composition for step-by-step mashup completion. In: IEEE International Conference on Web Services, ICWS, pp. 700–701 (2011)
14. Yu, S., Woodard, C.J.: Innovation in the Programmable Web: Characterizing the Mashup Ecosystem. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 136–147. Springer, Heidelberg (2009)
15. Wang, J., Chen, H., Zhang, Y.: Mining user behavior pattern in mashup community. In: 10th IEEE International Conference on Information Reuse & Integration, IRI, pp. 126–131 (2009)

16. Lim, J.H., Lee, K.-H.: Constructing composite web services from natural language requests. Journal of Web Semantics 8(1), 1–13 (2010)
17. Briscoe, T., Carroll, J., Watson, R.: The second release of the RASP system. In: Proc. of the COLING/ACL Conference, pp. 77–80 (2006)
18. Li, Y., Bandar, Z.A., McLean, D.: An Approach for Measuring Semantic Similarity between Words Using Multiple Information Sources. IEEE Transactions on Knowledge and Data Engineering 15(4), 871–882 (2003)
19. Klusch, M., Fries, B., Sycara, K.: OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. Journal of Web Semantics 7(2), 121–133 (2009)
20. Roy Chowdhury, S., Daniel, F., Casati, F.: Efficient, Interactive Recommendation of Mashup Composition Knowledge. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) ICSOC 2011. LNCS, vol. 7084, pp. 374–388. Springer, Heidelberg (2011)
21. Melchiori, M.: Hybrid Techniques for Web APIs Recommendation. In: 1st International Workshop on Linked Web Data Management, LWDM, pp. 17–23 (2011)