Composing Data-Driven Service Mashups with Tag-based Semantic Annotations

Xuanzhe Liu[1], Qi Zhao[1], Gang Huang[1], Hong Mei[1], Teng Teng[2]
[1]*Key Laboratory of High Confidence Software Technologies (Peking University)Ministry of Education*
[2] *Kingdee Middleware Company Ltd*
Email: {liuxzh,zhaoqi06,huanggang}@sei.pku.edu.cn, meih@pku.edu.cn, teng_teng@kingdee.com

***Abstract*—Spurred by Web 2.0 paradigm, there emerge large numbers of *service mashups* by composing readily accessible data and services. Mashups usually address solving situational problems and require quick and iterative development lifecyle. In this paper, we propose an approach to composing data-driven mashups, based on tag-based semantics. The core principle is deriving semantic annotations from popular tags, and associating them with programmatic inputs and outputs data. Tag-based semantics promise a quick and simple comprehension of data capabilities. Mashup developers including end-users can intuitively search desired services with tags, and combine several services by means of data flows. Our approach takes a planning technique to retrieving the potentially relevant composition opportunities. With our graphical composition user interfaces, developers can iteratively modify, adjust and refine their mashups to be more satisfying.**

***Keywords*-Service composition, Service mashup**

I. Introduction

Spurred by Services Computing and Web 2.0 paradigm, the Web is currently moving towards a participatory and "*programmable*" platform. One noteworthy trend over the Web is the rapid growing "*service mashups*"[1] that combine existing web services and data sources[2]. For example, as of February 2011, Programmableweb.com has published more than 5,500 web mashups. Several mashup tools have been developed, such as Yahoo! Pipes[1], IBM DAMIA[2] and Google Mashup Editor[3], support mashup development with several facilities[3]. However, we argue that there are still some key challenging issues remained. Firstly, since mashups are usually developed for meeting situational requirements in short development lifecycle, and many mashup developers are *end-users*, they exactly wish fast discovery of desired services and easy integration with less programming efforts. Secondly, due to the ever-increasing number of Web services, it requires simple and native semantic approaches to deal with the interoperability and integration of services. Thirdly, most mashup developers want to figure out all the intermediate steps needed to generate the desired mashups automatically[4], so it might take the evolving collections of services that can possibly incorporate with existing ones.

Addressing the problem above, we propose an approach for composing service mashups. We take the fact that most of current service mashups are exactly "*data-driven*" ones[3]. Inspired by the success of popular social tagging for the Web resources, we derive the tag-based semantics, and associate them with the programmatic inputs and outputs data of Web services. Tag-based service model is relatively simple and close to natural representation. Developers including end-users can simply take tag-based search for service discovery, and combine a set of tags as tag-based data flows, for describing their desired outputs. We develop a two-phase forward-backward planning technique, to retrieve all possible solutions constituting the outputs, based on the tags in repository. These solutions not only contain the desired outputs, but also some additional interesting or relevant ones as potential composition opportunities. Our composition runtime interprets the tag-based solutions and generates the mashups. Such makes the mashup development perform in "*data-centric*" fashion, and lowers the complexity of underlying developer programming tasks. Additionally, with our graphical composition environment, developers can obtain all immediate composition results visually, and iteratively refine their goals until the final outputs satisfying. The main contributions of this paper are as follows:

- A tag-based service semantic model for providing simple and intuitive understanding and discovery of services capabilities;
- A data-driven service composition approach for lowering the complexity of underlying programming efforts;
- A planning technique for retrieving both desired outputs and some possible interesting or relevant composition opportunities;
- A graphical composition tool implemented for aiding the rapid and high-quality mashup development.

The rest of this paper is organized as follows. Section II describes our approach with motivating example. Section III describes the approach overview, with the tag-based service model(in Section IV), the two-phase search planning techniques (in Section V) and the implementation of iMashup UI (in Section VI). Section VII evaluates the approach with experimental results. Section VIII presents related works and Section IX ends the paper with conclusion and future work.

[1]Yahoo Pipes. http://pipes. yahoo.com
[2]IBM DAMIA. http://services.alphaworks.ibm.com/graduated/damia.html
[3]Google Mashup Editor. http://code.google.com/intl/zh-CN/gme/

978-0-7695-4463-2/11 $26.00 © 2011 IEEE
DOI 10.1109/ICWS.2011.31

II. Motivating Example

To begin with, we describe a motivating example, and illustrate how the tag-based composition approach reduces complexity for creating new mashups. Suppose John is planning a trip and building an application that quickly finds hotel and nearby restaurants in the tourist city. Firstly, John selects a *HotelSearch* web service, which accepts inputs as *("hotel","city","state")* and generates outputs as *("address", "telephone","fax", "zipcode")*. John next selects another Web service *BooRah RestaurantSearch*[4], which can produce outputs *("rest-name","address", "telephone")* with the inputs *("food-pref", "zip")*. Then John extracts the *("zipcode")* of *HotelSearch* using XPath query and connects it to *BooRah RestaurantSearch* service as input parameter *("zip")*. Suppose John submits the requests like *("Hilton", "Washington DC", "Chinese food")* to the mashup application, the hotel and its nearby *Chinese* restaurants will be retrieved and displayed on map.

In the scenario above, John has to investigate each Web service specification (WSDL files or API documentations) when creating the mashups. It requires complex programming skills to process XML (e.g., using XPath). And, as one zipcode usually covers several districts, the *BooRah RestaurantSearch* service might not return the closest restaurant to the hotel.

Later, we make John re-create such a mashup with our approach. John begins by selecting some keywords *"Hilton","Washington"* as the initial input, for these keywords natively describe his requirements. Our approach suggests a list of services that can accept these keywords, e.g., the *HotelSearch*, *HotelRanking* (which returns the reputation rank with a given hotel and city) and *HotelReservation* (which requires the hotel name, city, state and date). John next selects *HotelSearch*, and gets the returned results *("1919 Connecticut Ave., NW, Washington DC", "Zipcode, 20009","Tel, 1-202-483-3000","Fax, 1-202-232-0438")*. John then selects the address *"1919 Connecticut Ave"* and adds a new tag *"Chinese food"* as the desired outputs. Our approach then plans a list of three composition solutions. As *BooRah RestaurantSearch* requires the *zip* as input (which is note included in current state), we suggest adding the tag *"20009"*, or assembling *Google GeoCode* service to convert *"1919 Connecticut Ave"* to the zipcode, and connecting it to the *BooRah RestaurantSearch* service. We also suggest the *Yahoo! Travel Guide* service, which can generate the rating and destination address of a restaurant, given the source address and food preference. The *Yahoo! Travel Guide* service might be more satisfying, as it returns the closest restaurant opposed to the hotel address instead of zipcode. Now John can attain outputs of the restaurant *"Raku Asian Diner"* with address *"1900 Queen Street"*. Then iMashup iteratively suggests the *Yahoo! Traffic Report* service planning the best traffic routine from *"Hilton"* hotel to *"Raku Asian Diner"*, the *Yahoo! Travel Guide* service listing the spots along the routine, and the *flickr* service displaying the related photos for the spots.

The example above illustrates some key features of our approach:

- ***Simple service model***: Our approach explores the semantics from tags, and associates them to service inputs and outputs. Taking the lesson learned from social annotations on *flickr*, *del.icio.us* and *twitter*, we believe that tag-based semantics simply represent service capabilities and facilitate service discovery.
- ***Data-driven composition***: Our approach takes the data-centric composition view which emphasizes the data over the services. Developers can simply use tag-based data flows to describe the composition, while not paying attention to underlying programming details.
- ***Higher quality assistance***: Our approach suggests ranked solutions that meet the desired goals, and might provide more potential relevant composition opportunities. It assists developers iteratively refine their mashups for higher quality.

III. Overview of Approach

Following the features above, Figure 1 shows the architectural overview of our approach: a service repository named *Service Community*, the *Service Advisor*, and the *User Interface*. Our approach consists of four main phases, each of which can be iterated more than once.

The *Service Community*provides *Service Publication Interfaces* for publishing individual services and mashups applications, and also exploits a *Service Crawler* to download available web services from existing registries (including Xmethods [5],Strikeiron[6],WebServiceX [7] and Esynaps[8] and the published mashups on Programmableweb. With a series of data mining techniques[5], the *Tag Manager* derives the tag hierarchies from a set of tags, which might be from service providers(e.g., in service documentations), service consumers (e.g., annotations, comments and feedbacks), or results of service invocations. The *catalog manager* keeps track of all services and mashups to facilitate the search of resources.
. With a data mining approach[5], we derive these tags into a set of semantically meaningful clusters. Then we associates tags with Web services inputs/outputs.

The *Service Advisor* is responsible for planning the composition solutions relevant to desired goals. It captures the current composition states and dynamically plans a set of candidate potentially relevant composition solutions that can be added to the mashups. It means that, according to

[4]http://api.boorah.com/restaurants/WebService

[5]http://www.xmethods.com
[6]http://www.stikeiron.com
[7]http://www.webservicex.net
[8]http://www.esynaps.com

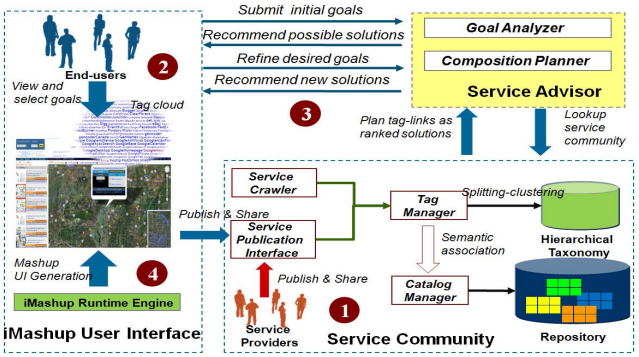

Figure 1. Overview of iMashup Approach

the semantics from the hierarchical tag taxonomy, once a tag contained in service operation *A*'s outputs can be semantically associated with another tag in service operation *B*'s inputs, a tag-based data flow (named ***Tag Link*** in this paper) will be created. All tag links are manipulated in a Directed Acyclic Graph (DAG). The *Goal Analyzer* interprets developers queries, *Composition Planner* will perform a graph-based planning process to generate all the possible relevant compositions that can constitute the tag-based goals at least once.

Mahsup developers view, search and select tags as desired outputs on the *User Interface*, as well as attain recommendations from *Service Advisor*. Developers are allowed to choose one of the generated composition solution. We have developed a prototype *iMashup Runtime Engine*[6] interprets the composition of corresponding services, and display the immediate results. Also, the developers can refine generated composition solutions, by adding new tags or importing new developed services.

IV. Tag-based Service Semantic Model

A. Tag-based Semantic Model

We firstly propose the tag-based service semantic model. One observed fact is that most services mashups development mainly follows a *"data plus code"* fashion[3][7][8]. It means that, most mashup developers may pay more attention on what data a Web service can consume and produce, and integrate them in a fast and simple manner. Based on such assumption, we associate a set of tags with input/output messages of Web services, as the semantic annotations describing the properties of the data that the Web service can consume and produce.

We begin with a basic model that associates tags with Web services operations. Usually, a Web service is published through WSDL with an accessible URL. Thus a Web service can be described as a set of 5-tuple space: *(ws, tag, url, user, time)*. It means that the Web service *ws* of the *url* is annotated by the *user* with *tag* at the specific *time*. In this model, we mainly focus on the tag and the service, and define the following set: $T=\{t_1,t_2, ..., t_N\}$,$\varpi=\{ws_1,ws_2, ..., ws_M\}$, where N and M respectively represents the number of tags and Web services in *Service Community* repository.

B. Deriving Hierarchical Tag Taxonomy

Although tags provide an easy and instinct view for annotating web service semantics, some limitations must be considered. As tags are freely and arbitrarily chosen instead of relying on a controlled vocabulary, tag ambiguity will significantly take side-effect on composition efficiency. Different people might use the same tag to express different meanings (homonyms), or different tags might denote the same meanings (synonyms), or the same tag might represent different meanings (polysemys). Such ambiguity will make incorrect matching between service data. Another problem is that tags only represent a flat but not hierarchical annotation structure, thus the large number of tags might bring difficulties to developers in browsing the system capability and application domain.

To respond to the difficulties for tag-based semantics, we have designed an unsupervised technique to derive several semantically meaningful concepts from tags[5]. Each cluster consists of several tags with similar specific concepts. For example, the tags {*"wind", "temperature","temperatureF", "humidity", "weather","NOAA", "37°C"*} all reflect the concept *"weather"*, therefore they are categorized into one cluster. To facilitate mashup developers for fast browsing, we assign the notion of ***"Feature Tag"*** for each cluster. It indicates that tag t_i covers lots of other tags in cluster C_j, so t_i is the *Feature Tag* that is more capable of summarizing the cluster C_j's semantics than any other tags in cluster C_j. For example, "*weather*" is the *Feature Tag* above.

After deriving the clusters as well as the *Feature Tags*, we now have attained a structured taxonomy T representing the hierarchical relationships between tags. The hierarchical tag taxonomy is denoted as T, which contains a set of semantic concepts $\{C_1, C_2, ..., \}$ and each C_i has several tags $\{t_{i1}, t_{i2}, ...\}$. For tag $t_1 \in T$ and $t_2 \in T$, if t_2 represents higher-level semantic than t_1, we formally define the t_1 is a sub-tag of t_2 as $t_1 \prec t_2$, indicating that Web services tagged by t_1 can be also semantically tagged by t_2. For example, the tag *"Google Map"* is the sub-tag of *"Map"*. Obviously, all tags in C_i are sub-tags of the *"Feature Tag"* F in its cluster. The sub-tag relationship can be transitive,if $t_1 \prec t_2$ and $t_2 \prec t_3$, we can conclude $t_1 \prec t_3$. Also, for $\forall t \in T$ is a sub-tag of itself, $t \prec t$. We define a tag transforming function f:

$$f(t') = t, \qquad \text{if} \qquad t' \prec t \tag{1}$$

C. Attaching Tag-based Semantics to Services

We then will formalize the service semantic model, by attaching tag-based semantic annotations to inputs and outputs messages. Assume that a Web service has two sets of parameters, ws^i=$\{I_1,I_2,...\}$ for request messages (as inputs) and ws^o=$\{O_1,O_2,...\}$ for response messages (as outputs). Let set ***P*** be the union of input and output parameters,$ws^i \cup ws^o$. For every parameter $p \in$ ***P***, with the derived hierarchical tag-based taxonomy *T*, we define a mapping function $\Gamma : P \rightarrow T$, between service parameters and tags:

$$\Gamma(p) = \begin{cases} t & \text{if exists } t \in T \\ \varnothing & \text{otherwise} \end{cases} \quad (2)$$

Equation (2) indicates that, if there exists a tag $t \in T$ semantically equivalent to parameter *p*, we can replace *p* by *t* in terms of composition. From our investigation, the Web service parameters are usually defined in form of *verb* plus *"noun"*, such as *"PostZipcodeRequest", "GetRestaurantInfoResponse"*, etc. So we extract the nouns in the parameters and bind them to a tag. In our approach, we bind a parameter with a *Feature Tag* F in *T*. From the inheritance relationship defined in last section, it means that all sub-tags $\{t_1,t_2,...\} \prec F$ can also be bound to *p*.

Associated with semantics, we then describe a Web service *ws* as a two-tuple,$<T^i,T^o>$, where T^i and T^o respectively represents the set of tag-based descriptions for inputs that can be consumed and outputs that can be produced. Here, we constrain that $T^o \subseteq T^i$, which ensures all inputs can be consumed by the service. Obviously, to guarantee the constraints above, for $\forall t \in T^o$, $\exists t' \in T^i$ such that $t \prec t'$.

V. Data-Driven Service Composition

A. Tag-based Composition Semantics

In terms of service composition, if a Web service ws_1 can produce t_1 as its output and the service ws_2 can consume t_1 or its father tag t_2 as its input, we can conclude that ws_1 and ws_2 are composable. Then the tag-based service composition problem is defined simply as the result of creating a data flow by tags, namely ***Tag Link (TL)*** in the following.

Definition V.1. *We denote* **TL** *as* $<T_1^i, T_1^o>$, $<T_2^i,T_2^o>$, ..., $<T_n^i,T_n^o>$ *for each* $<T_k^i,T_k^o>$ *corresponds to a Web service* ws_k. *If* $\forall k$, *the composition is valid when the following two preconditions are satisfied:*

1) $\forall t \in T_{k+1}^i, \exists t' \in T_k^o$, *such that t=f(t');*
2) $\mid T_k^o \mid \geq \mid T_{k+1}^i \mid$.

The above definition indicates the data dependencies between Web services: subsequent Web services may use the outputs produced by preceding Web services as inputs. Such data dependencies are described in terms of tags, and determined by tag-based semantics. The first precondition ensures the semantic mapping and propagation between Web services, and the second precondition ensures that no extra parameters are left.

B. Tag-based Composition Goal

From the data-centric perspective, the composition problem is just that achieving their desired goals from their initial requests, while not making them know the underlying composition details. As we have associated tags with services, the mashup developers can now simply describe their goals in form of tags, and submit the requirements to our system. In our approach, given the developer goals $g \subseteq T$, select a set of Web services $\varpi = \{ws_1, ws_2, ...\}$ such that each Web service $ws_k \in \varpi$, for $\forall t \in g, \exists t' \in \Gamma(P)$ and $t' \prec t$ where P is the parameters set of ϖ. It indicates that ϖ can finally produce all the tags in g or sub-tags thereof. For example, the goal *Map, Temperature, Routine* can be matched to a the *Google Map Service* with the output *gmap* due to the sub-tag relationship $gmap \prec Map$.

The developer goal can be directly matched to tag-based outputs of a single Web service. Moreover, it can be accomplished by searching a sequence of Web service that can produce the desired outputs. Such sequence of Web services composition can be viewed as an analogy of searching the "*hyperlinks*" between web pages[5]. In other words, goal satisfaction is the search for all possible ***Tag Links*** that can be constructed from an initially given set of tags.

C. Data-Driven Composition with Tag-based Semantics

The tag links are used to create a DAG. Then our approach performs a graph-based planning approach for tag-based goal-driven composition.

1) Data-Driven Composition Planning: The problem of tag-based, data-driven composition can be described as that of generating several *Tag Links* that can produce the output satisfying the desired goal. A data-driven composition problem for service composition can be transformed to the graph-planning model with AI planning techniques. We formally define such a problem as a planning problem as a seven-tuple:
Ψ=$< T, \varpi, S, r_0^i, g, \Theta(s), \tau >$, where:

1) T is a set of hierarchical tags;
2) ϖ is a set of Web services, $\forall ws \in \varpi$ is a two-tuple defined in last Section, in form of $< T^i, T^o >$;

3) S is a set of composition states,$\forall s \in S$ is a collection of tags in T;
4) $r_0^i \subseteq T$ is the initial input tags, and the initial state of the composition s_0 = r_0^i;
5) $g \subseteq T$ is the desired goal described by tags, and the final state of the composition $s_G \supseteq g$;
6) $\Theta(s)$ is a set of Web services $\{ws_1, ws_2, ...\}$, $\Theta(s) \subseteq \varpi$, such that $\forall ws_k, T_k^i \subseteq s$. It means that ws_k can be invoked in the state s by requesting tag $t_s \subseteq T_k^i$;
7) τ is the state transition function, $\tau(ws_k, s')$=s that transit the state s' to s, in condition that s=$s' \cup T_k^o$ and $ws_k \in \Theta(s')$. It means that the output tags of ws_k in $\Theta(s')$ are the subset of input tags of one Web service $ws_l \notin \Theta(s')$;

Definition V.2. *Suppose that the initial input tags r_0^i and desired goal g, where $g \subseteq T$ and $r_0^i \in T$. The tag-based service composition problem is to find a finite sequence of services, $ws_1, ws_2, ..., ws_n$ such that:*

1) $\forall ws_k (k = 1, ..., n)$ *is a two-tuple* $< T_k^i, T_k^o >$;
2) ws_k *is invoked sequentially from 1 to n;*
3) $\forall ws_k$, $T_k^o \supseteq T_{k+1}^i$;
4) $g \subseteq (r_0^i \cup T_1^o \cup ... \cup T_n^o)$;
5) *the total cost* $\sum_{k=1}^{n} C(ws_k)$ *is minimized.*

2) Composition Planning Algorithms : As the size of the search space will be exponential to the size of tags, to address the intractable tag-based service composition, we propose a heuristic planning algorithm within polynomial-time complexity. When the developer selects a tag from the tag cloud or input a keyword as the initial request r_0^i, the planning algorithm first computes the cost of achieving individual tags starting from r_0^i by conducting a *forward search*. Such a Depth-First step constructs all the possible *Tag Links* that can perform the final goal. Based on the results above, the planning algorithm then approximates the sequence of *Tag Links* that connects r_0^i to the final goal by a *regression search* step.

In the forward search stage, the planning algorithm employs a cost value $g_{r_0^i}(t)$, which indicates the cost of achieving $t \in T$ from r_0^i. We can recursively compute the value of $g_{r_0^i}(t)$ according to the following equation:

$$g_{r_0^i}(t) = \min C(ws) + \max_{t' \in s'} g_{r_0^i}(t') \tag{3}$$

where *C(ws)* is an invocation cost of a Web service. In our model, we simply assume that the default value of *C(ws)* is 1. Initially,$g_{r_0^i}(t)$ is assigned to 0, if $t \in r_0^i$, or to ∞ otherwise. Then the initial state s is r_0^i. For each $\forall ws_k \in \Theta s$, we add all tags in T_k^o to the state s and the function $g_{r_0^i}(t')$ is updated until $\forall t \in T_k^o$. In case that $\Theta(s)$ does not increase any longer, it means that no additional tags can be added from the search space and no solution can be found. We maintain a Web service ws as a predecessor Web service of $t \in T$ if ws is the first service that can produce t, and denote Ω as the set of predecessor Web services for t.

Now we can generate all the possible composition solutions to g from r_0^i. Next step of the planning algorithm is to retrieve the optimal composition plans by a backward search step, as shown in algorithm V.2. For each state s in all generated solutions, we associate a temporary goal g' to denote the goal starting from s. Initially g' is assigned to r_0^i, and we specify a set of Web services ω that belongs to $\Omega(t)$, where $t \in g'$.The planning algorithm selects a Web service from ω (lines 4-7 in algorithm) at each backward step. We use the heuristic that choosing a Web services matching g' earlier in the regressive search to make the arrival at the initial state faster. We identify $\delta(ws_k)$ as the value of $[T_k^o] \cap g'$, which indicates that a Web service can match g' better than others in current solutions. A Web service with higher δ holds a higher probability to match g' so that the search space will be reduced. According to the hierarchical tag structures, our planning approach can improve the composition quality by avoiding partial composition as much as possible.

The cost of our planning algorithm consists of basic forward search planning and regressive backward planning. The procedure of forward search can be transformed to a depth-first graph traversal procedure. The maximum length of *Tag Links* is limited by $|\varpi|$, so the iteration's upper limit is $|\varpi|$. At each iteration step there are at most $|\varpi|$ Web services and associated $|T|$ tags examined, so the forward search is accomplished in $O(|\varpi|^2|T|)$. On the other hand, the regressive search procedure has at most $|\varpi|$ iterations. At each iteration step, a binary-sort algorithm can be employed to retrieve the a Web service ws with the largest δ, and such a step costs O($|\varpi| \log |\varpi|$). So the total cost of backward search is $O(|\varpi|^2 \log |\varpi|)$. Due to the fact a Web service has a set of parameters (described as tags), obviously $|T| \gg |\varpi|$, and the cost of forward search will be much bigger than that of backward search. To draw a conclusion, the cost of our

Input: r_0^i and g
Output: Ω is the predecessor set of Web services for t
1 initialization;
2 s=r_0^i:C=$\varnothing$;d=0;;
3 **while** $\neg(s \supseteq g)$ **do**
4 λ=$\{ws_k | ws_k \in \Theta(s), s \notin C\}$;
5 **for** t *in* $T_k^o(ws_k \in \lambda)$ **do**
6 **if** $g_{r_0^i}(t)$=$\emptyset$ **then**
7 $g_{r_0^i}$(t)=d;$\Omega(t)$=$\Omega(t) \cup ws_k$;s=s$\cup t$;
8 **end**
9 **end**
10 C=C$\cup\lambda$;d++;
11 **end**

Algorithm V.1: Basic Forward Search Planning

Input: r_0^i, g and Ω
Output: A tag link TL

initialization;
g'=r_0^i;
while $\neg(g' = \varnothing)$ **do**
  ω=$\cup_{t \in g'} \Omega(t)$;
  $\chi = \max \delta(\omega)$;
  TL=$TL \cup \chi$;
  g'=$[g' \setminus (T_\chi^o \cup r_0^i)] \cup T_\chi^i$;
end
s=r_0^i;
while $\neg(TL = \varnothing)$ **do**
  if $(ws_k \in \Theta(s)$ *and* $ws_k \in TL)$ **then**
    $s = s \cup ws_k$;
    $TL = TL \setminus T_k^o$;
  end
end

Algorithm V.2: Regressive Backward Search Planning

planning algorithm is determined by the number of services and the size of tags in the *Service Community*.

VI. User Interface Design

One of the key features of our approach is to develop a user-friendly composition interface. We have implemented the browser-based iMashup composition tool[6][9] to support the data-driven mashup composition. As shown in Figure 2, a tag cloud will be first displayed to the mashup developers, indicating all the relevant functionalities in current *Service Community*. Tags with larger fonts than others, are the *Feature Tags*. Developers can select one or more tags from the tag cloud, or directly search for several tags as composition goals. The *Service Advisor* recommends a set of optional composition solutions that can constitute the goal, and display them on the screen by sorting their ranking values. Each recommended solution not only has to contain all the tags of the goal, but also may bring new tags as the potential relevant composition goals. Developers are allowed to iteratively refine their goals by removing tags, or adding tags from the recommended solutions. Each time the goal is modified, it will result in new recommendations generated from the planning technique. Meanwhile, it will immediately retrieve all the possible tags in *Service Community* that can reach such goal, and synchronize the tag cloud view to display only relevant tags.

At anytime, developers can click each alternative solution, and the user interface will display a graphical view of the its composition topological structure. The graphical representation consists of a set of vertices and edges, where each vertex stands for the Web service

[9]iMashup is now partially open-sourced on http://code.google.com/p/imashup/

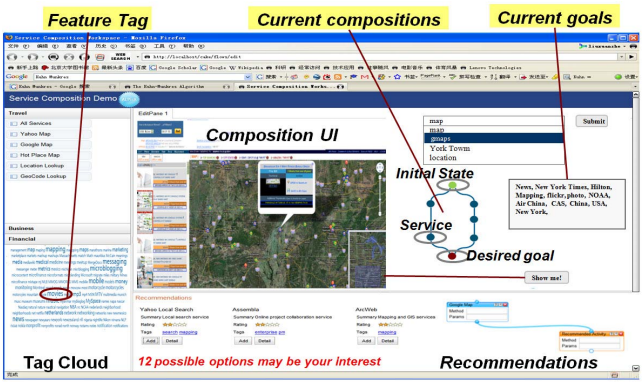

Figure 2. iMashup UI Interface

associated with tags and each edge means the data flow (*tag links*) between the Web services. It also allows the developers to have a deep insight, by double-clicking the vertex, and they will be navigated to the *"Service Component View"* mode and view the detailed descriptions of the services. For space limited, the design details can be referred from our previous work[6]. Once the developers regard that a (partial) composition solution is currently satisfying, they can immediately run it by clicking *"Show me!"*. We have designed an event-driven service composition runtime to delegate the requests to the remote services. Our composition tool interprets data exchange in JSON[10] format. The composition results will be displayed to developers, to help them interactively revise whether their goal has been satisfied, and iteratively refine their goals based on current solutions.

VII. Experimental Evaluation

This section evaluates our approach from two aspects: the composition quality and planning performance. Since tag-based semantics play the crucial role in our approach, we have taken more than 200 Web services, 3,100 mashup applications and 23,971 tags as data sets[11].Using our data mining techniques[5], we have attained 594 distinguished clusters.

A. Composition Quality Evaluation

To evaluate the composition quality, we conduct experiment by testing the outputs produced by our planning techniques. We set up a series of experiments to evaluate the concept coverage of tags. Twenty developers arbitrarily search their goals on our system, and we check how many of

[10]JavaScript Object Notation. http://www.json.org

[11]These are from *http://www.programmable.com* and existing Web service resources on *Service Community* platform described in [5]

their desired goals or relevant interested outputs are captured by the composition algorithm. *Top-k* in Figure 3 means the *k* ranked composition solutions including their desired outputs. It can be observed that in the *Top-5* curve, about 80% of all developers have all their desired or relevant goals in the *Top-3* recommendations, and about 10% users have more than 90% desired or relevant goals. The same observation holds for the *Top-5* plot where 78% of the users can discover more than 90% of their desired or relevant goals covered by the our composition techniques. Although only 68% of all users are satisfied with 75% results from *Top-10* returned plans, we argue that the results are acceptable, for the developers are more likely to prefer less than five options. The results have shown that our planning algorithm can capture most user desired goals or potential interests.

Another results of our planning approach is the discovery

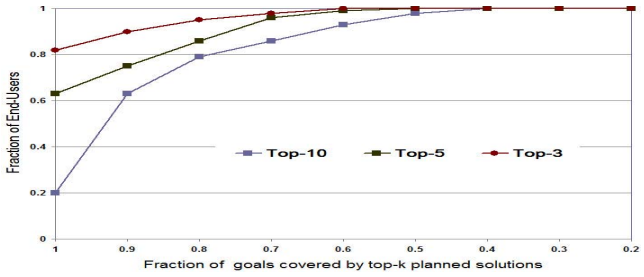

Figure 3. User Requirements Satisfaction

of potential composition opportunities. We choose 25 sample mashup applications from *Programmableweb*, each of which at least contains 10 services. For each sample mashup, we extract the outputs from its services and the final output. For each output and the input tags, we run the planning composition algorithm to produce the plans that can retrieve the same output given the user inputs and form a mashup incrementally. As shown in Figure 4, we can observe that the planned mashups almost have the same size as the original mashups. For further investigation, we check that the coverage of common concepts associated with the outputs between the planned solutions and original mashups can reach a very high rate. It means that our algorithm can generate desired outputs as the original mashups. Moreover, another fact is that our planning techniques can retrieve more concepts than the original mashups in over 90% cases. It means that, based on the hierarchical tags, our approach can generate more potential relevant composition solutions to enlarge the developers' composition opportunities.

B. Composition Planning Performance Evaluation

This experiment aims at verifying the scalability of the data-driven planning composition with the number of services. We set up a series of simulations by running multiple queries against different sizes, from (1,000 to 10,000

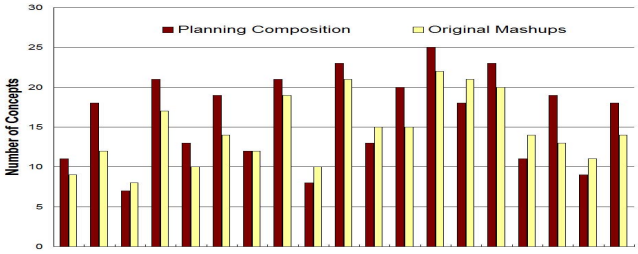

Figure 4. Output Concepts Coverage

services). As the hierarchical structure can be periodically generated, we assume that the maximum size of tags is 5000, which indicates upper limit of states of the search space. Figure 5 shows the execution time for a series of queries. For each repository size, all planning durations keep increasing with the number of request inputs involved. We can note that, the planning algorithm can answer most of the requests in less than 60 seconds. Once the size of services exceeds 8000, the algorithm has to cost more than 2 minutes to recommend the composition solutions. Obviously, such overhead cannot be accepted by the developers in real scenarios, for it is too long for them to wait for retrieving the immediate responses.

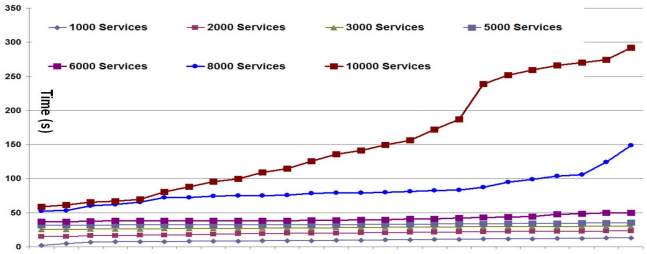

Figure 5. Planning Performance

VIII. Related Works

Mashups have also encompassed a wide research interests in the last few years. A number of mashup approaches, techniques and tools have emerged, both for enterprise and personal purpose[3]. Following the data-centric fashion, several works have been done for the rapid prototyping of mashups. *MashupAdvisor*[4] suggests recommendations of mashups based on current inputs; *Marmite*[9] supports processing data services in a flow manner and integrate with other data services by operations like join and filter; *SpreadMash*[10] takes the spreadsheet model to combine data from several Web services; intelligent matching[11] and automatic completion[12] also make the creation of data mashups more efficiently.

Rather than processing data heterogeneity in ad-hoc manners, it is argued that mashup development might leverage some semantic approaches to coordinate the

interoperability of diverse Web services[1]. We argue that, the exploration of tag-based mashup development might be unique, compared with most existing works. Tag-based semantics require much less skills and are much more instinctive and simpler. The successful adoption of tags on social networking platforms, such as *Del.li.cous*, *flickr* and *Twitter*, has revealed that tags not only benefit the organizing contents, but also facilitate the navigation for users to discover interesting resources. We also note that, the unified use of tags can be accepted by service providers, business analysts, application developers and even end-users. As such, the tag-based service semantics make service capabilities more natively and simple described. It will significantly bridge the gap of requirement and development, and accelerate the data-driven mashup development without so many underlying programming efforts. It should be particularly noted that, the primitive of tag-based service composition was also proposed in the MARIO system[13], which provides us useful inspiration. However, our approach works by checking if a set of *tag links* available in the current state can be used to construct an input to a Web service; and if so, it generates a new *tag links* corresponding to the desired outputs. Therefore, compared to MARIO, our approach can enlarge the space of composition opportunities, find potential relevant services for their emerging situational requirements, and promote the mashup for higher quality with useful assistances.

IX. Conclusions and Future Work

This paper describes how to make use of pervasive tags for the data-driven mashup development tasks. We have described tag-based service semantic model, data-driven composition planning techniques, and the graphical composition user interfaces.

The current work mainly aims at personalized mashups, and cannot well support complex composition logics in enterprise context. One ongoing interesting issue is to derive some *service oriented collective intelligence*[14] for better problem solving.

Acknowledgment

This work was supported by the National Basic Research Program (973) of China under Grant No. 2009CB320703, the National Natural Science Foundation of China under Grant No. 60821003, 60873060, 60933003, 61003010; the EU FP7 Project under Grant No. 231167; and the National S&T Major Project under Grant No. 2009ZX01043-002-002.

References

[1] D. Benslimane, S. Dustdar, and A. P. Sheth, "Services mashups: The new generation of web applications," *IEEE Internet Computing*, vol. 12, no. 5, pp. 13–15, 2008.

[2] J. Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding mashup development," *IEEE Internet Computing*, vol. 12, no. 5, pp. 44–52, 2008.

[3] V. Hoyer and M. Fischer, "Market overview of enterprise mashup tools," in *Proceedings of 6th International Conference on Services Oriented Computing*. Berilin-Heidelberg: Springer-Verlag, 2008, pp. 708–721.

[4] H. Elmeleegy, A. Ivan, R. Akkiraju, and R. Goodwin, "Mashupadvisor: A recommendation tool for mashup development," in *International Conference on Web Services*, September 2008, pp. 337–344.

[5] X. Liu, G. Huang, P. Wen, and H. Mei, "Discovering homogeneous web services community in the user-centric web environment," *IEEE Transactions on Services Computing*, vol. 2, no. 2, pp. 167–181, April-June 2009.

[6] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards service composition based on mashup," in *IEEE SERVICE CONGRESS*, 2007, pp. 332–339.

[7] A. Bozzon, M. Brambilla, F. M. Facca, and G. T. Carughi, "A conceptual modeling approach to business service mashup development," in *Proceedings of IEEE International Conference on Web Services, ICWS 2009*. Los Angeles, CA, USA: IEEE Computer Society, 2009, pp. 751–758.

[8] A. Leff and J. T. Rayfield, "Eds: An elastic data-service for situational applications," in *Proceedings of IEEE International Conference on Web Services, ICWS 2010*. IEEE Computer Society, 2010, pp. 187–194.

[9] J. Wong and J. I. Hong, "Making mashups with marmite: Towards end-user programming for the web," in *Proceeding of ACM 2007 SIG Conference on Computer/Human Interaction (CHI'07)*, May 16-24 2007, pp. 1435–1444.

[10] W. Kongdenfha, B. Benatallah, J. Vayssière, R. Saint-Paul, and F. Casati, "Rapid development of spreadsheet-based web mashups," in *Proceedings of 18th World Wide Web Conference*, J. Quemada, G. León, Y. S. Maarek, and W. Nejdl, Eds., 2009, pp. 851–860.

[11] C. Wu, T. S. Dillon, and E. Chang, "Intelligent matching for public internet web services towards semi-automatic internet services mashup," in *Proceedings of IEEE International Conference on Web Services, ICWS 2009*. Los Angeles, CA, USA: IEEE Computer Society, 2009, pp. 759–766.

[12] T. M. O. Greenshpan and N. Polyzotis, "Autocompletion for mashups," in *Proceedings of 35th International Conference on Very Large Data Base (VLDB'09)*, Lyon, France, 2009, pp. 538–549.

[13] E. Bouillet, M. Feblowitz, Z. Liu, A. Ranganathan, and A. Riabov, "A faceted requirements-driven approach to service design and composition," in *Proceedings of 2008 IEEE International Conference on Web Services (ICWS 2008)*. IEEE Computer Society, 2008, pp. 369–376.

[14] M. Tanaka, Y. Murakami, D. Lin, and T. Ishida, "Service supervision for service-oriented collective intelligence," in *Proceedings of 2010 IEEE International Conference on Services Computing, SCC 2010*. IEEE Computer Society, 2010, pp. 154–161.