Ontology Modeling for REST Open APIs and Web Service Mash-up Method

Wan Jung
Electronics and Communications Engineering
KwangWoon University, Seoul, Korea
jung_wan@kw.ac.kr

Sang Il Kim
Electronics and Communications Engineering
KwangWoon University, Seoul, Korea
rlatkd234@kw.ac.kr

Hwa Sung Kim
Electronics and Communications Engineering
KwangWoon University, Seoul, Korea
hwkim@kw.ac.kr

***Abstract*- According as the web technology rapidly advances, web services are taking a step further up to the era of Web 2.0 which goes beyond the existing closed web services and enables ones to open and share information freely. Web services are usually provided in the form of Open APIs (Application Programming Interface). As the number of open APIs is increasing and its variety of the web services have wide spectrum, mash-up web services that combine several open APIs are fairly proliferated on Web 2.0 environment and even the service users are also interested in producing their own mash-up web services.**

When generating mash-up web services, however, the users have difficulties in that they should find Open APIs they need by looking up the web service information by their own and also they are forced to determine whether it is necessary. Also, when looking up the web service information, numerous unnecessary Open APIs are retrieved along with the desired Open API, which makes it difficult for users to find desired Open APIs readily and swiftly, because the current Open API search is performed only by key words. To overcome these obstacles, it is necessary to provide a mash-up platform for the service users to be able to generate their own new mash-up web service on their own comfort and purpose they want.

In this paper, we present the key technologies for automatic generation of new mash-up service using open APIs. As the key technologies, ontology modeling method specifying the semantic open API information which is necessary for mash-up, is presented. And, the automatic service mash-up method using the ontology will be presented.

I. INTRODUCTION

The unique characteristics of web 2.0 are participation, sharing and openness.[1] Among them, open API which means protocol or rule for using the web service is propagandized as openness and its use is also skyrocketed as growth pace of mash-up service.[2] Due to the rapid distribution of mobile devices such as smart phone, tablet pc, service users who is not the service developers, also want to participate in the process of service production personally. Users may readily create and share a new web service by means of Open API from web service providers. This kind of trend has been an important turning point for increasing uses of Open APIs.

Although, the pattern of mash-up services using open APIs, has uncountable forms due to wide variety of open APIs, the service users, however, has some inconveniences in that they should look up the information about web service they need by their own and also they are forced to determine whether it is necessary. So, it is difficult for the service users to produce the web services that meet the preference or intention of individual service users. To overcome these obstacles, the automatic mash-up service platform, which is able to generate the new mash-up services automatically based on the purpose of service users, is needed.

This research presents the ontology modeling method, which defines open API information semantically to make it possible to implement automatic mash-up. With the appropriate modeling of open API into ontology, it is possible to generate mash-up service automatically for the general web service users. The followings are the contents of the research: In section II, succinct explanation of the related technology that is needed in this paper is described. In section III, illustration of effective ontology modeling method of open API for the automatic mash-up is presented. In section IV, the algorithm which decides if mash-up is possible or not between API methods for automatic mash-up is described. And finally, we describe the process of generating service URI for service mash-up, and in section V, conclusion is presented.

II. BACKGROUND

A. REST(Representational State Transfer)

REST is a type of design for access to web pages. REST structurally represents URL with Request transfer method inquiring the result from the server and it exchanges the data based on the HTTP protocol. Unlike current web services based on SOAP, REST is most frequently used method for open API because of ease of operation with HTTP that has no additional protocol.

B. Open API and Mash-up Service

Mash-up service means a comprehensive service that is generated by combining information, contents or other web

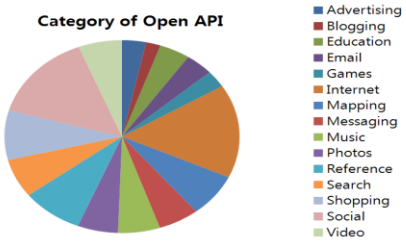

Figure 1. Ratio according to web service categories

services provided from the web in the form of Open APIs. The key feature of the Mash-up is open API. Open API provides the interface that makes it possible to generate new web service. Because Open API is available for all users, it is considered user friendly interface comparing past version which opens internally only.

C. Ontology

Ontology is the tool for explaining concept and the correlation. In other words, it is the meaning of system elements in the information system field as terms or concepts. For semantic web, ontology describes the common language making it available for the computer to process the portion that a person handles intuitively and literally. The examples of ontology expression languages are RDF, RDFs and OWL (Ontology Web Language), which have triple structures including subject, property and object.

III. ONTOLOGY MODELING FOR OPEN API

In this section, for the automatic mash-up using the Open APIs, the ontology modeling method through the analysis on web service categories, URI paths, and input & output parameters of Open API method is described.

A. Open API Ontology Modeling based on Service Category

Web services, which are provided in the form of Open APIs, have very wide spectrum of services. Fig. 1 shows the representative web services for each category [3]. As a result of this wide spectrum, it is necessary that Open APIs should be classified and sorted according to the service categories based on their frequency of uses. In other words, Open API ontology modeling needs to be designed according to the service categories. And, the categories assorted by the type of services are located at upper class in the hierarchy and the Base URI for Open API representing each web service are located at lower class of hierarchy, as shown Fig. 2

B. REST Open API Ontology modeling based on URI Paths

As shown in Fig. 3, assessing the protocol ratio used for Open APIs from programmable web site [4], the portion of Open APIs based on REST is much bigger than the other protocols. This is due to the convenience in handling the service resources with URI and HTTP [5], so its utility for mash-up is very high among others.

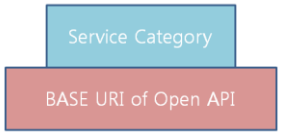

Figure 2. Hierarchical relationship between service category and base URI

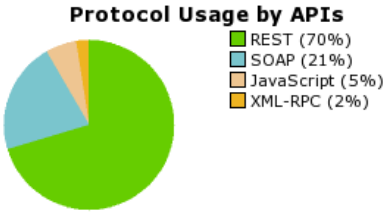

Figure 3. Ratio of protocols used by Open APIs in programmable web site

In this paper, we modeled ontology on Open APIs based on REST protocol, which is most frequently used for mash-up service. Since REST protocol based open API can handle Open API resources easily by URI, URI-based ontology modeling is necessary. And, hierarchical ontology modeling is desirable for the REST protocol based open API information, because URI paths have hierarchical structure like UNIX file system. Ontology modeling is performed according to the following rules: URI scheme of URI structure is handled only when HTTP is used and the paths of ontology node should follows URI of each open API. And the paths of ontology node are classified into absolute paths and relative paths for simplification of ontology and URI particularity. For the additional paths that are added to the current URI, it is defined as relative one. In other words, same as the file system in which child directory is defined as '/' symbol followed by child directory name, if additional URI is added to current URI, it should be defined as relative one. And Base URI which is a turning point is defined as absolute one. [6]

Table I illustrates category, Open API name, Base URI and Query Parameter taking an example of 'flickr' [7] that is the photo-sharing service. In order to access flickr API method, URI would be generated by adding method name, api_key for authentication to the already existing base URI on Table I, and then the access to the service is possible using newly generated URI. So the ontology need to be designed hierarchically as shown in Fig. 4.

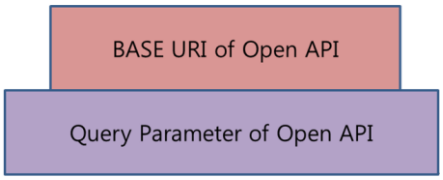

Figure 4. hierarchical relationship between base URI and Query parameter

Table I. Details of 'flicker' API

Category	Photo
API Name	Flickr
Base URI	http://api.flickr.com/services/rest
Query Parameter	Method, api_key ..

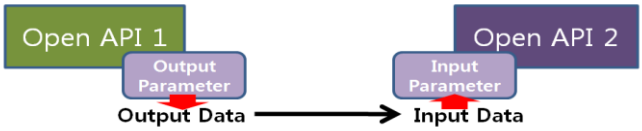

Figure 5. Input and output relationship between two Open APIs

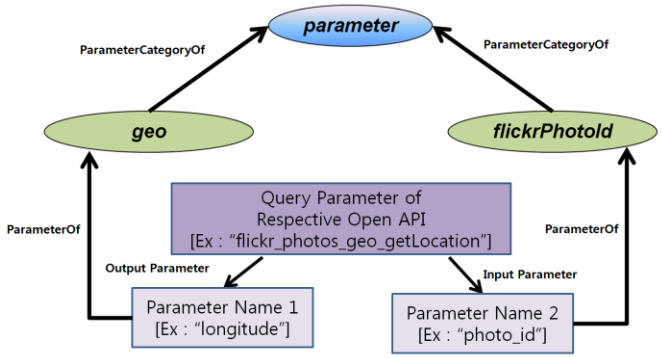

Figure 6. Property of Open API input and output data

C. Open API Method Parameter Ontology Modeling

Mash-up service means a comprehensive service that is generated by combining information, contents or several other web services provided in the form of Open APIs from the web. [8] The mash-up service could contain more than two types of open APIs. As shown in Fig. 5, if more than two open APIs are conglomerated, output data of an open API is provided to other Open API as input data. Therefore, in order for mash-up to be possible between Open APIs, transferred data should have the same semantics.

In order to determine if mash-up is possible or not, it is necessary to classify input and output parameters by semantically, and then parameter category including both input and output parameters should be modeled as input-output parameter ontology. In other words, if input parameter of a first Open API and output parameter of a second Open API are semantically identical, parameter ontology should be designed so that parameters of two Open APIs belong to same category. If parameters of two Open APIs belong to same parameter category, these two APIs are considered mash-up-able. So, the parameter is a barometer for the compatibility between Open APIs. Fig. 6 shows that input and out parameters belong to different parameter categories.

IV. AUTOMATIC MASH-UP METHOD BASED ON OPEN API ONTOLOGY

In this paper, Open API ontology for automatic mash-up is modeled based on the classification according to service categories, URI paths, and input & output parameters of Open API method. And the query engine is also implemented for ontology search using Java and Jena 2.6.4 library. [9]

Fig. 7 illustrates the hierarchical structure of designed 'flickr' open API ontology. Query parameter (Open API Method Parameter*)* ontology has been designed hierarchically, dividing into method dependent one and non-method dependent one. Non-method dependent parameter is subordinated by the method dependent parameter, because query parameters are different according to methods. Therefore, the method dependent ontology node should be located as upper class than the non-method dependent one. And also because all of the resources could not be shown, the picture is presented as abbreviated form. The followings are the omitted one

- Fig. 7 shows the 'flickr' open API resources in 'photo' service category, but it presents only representative APIs per each service category in the case of other categories.
- Api-key query parameter for authentication is necessary for all of the methods in 'flickr' API but it is omitted for legibility.
- Input and output values are also partially presented for legibility.

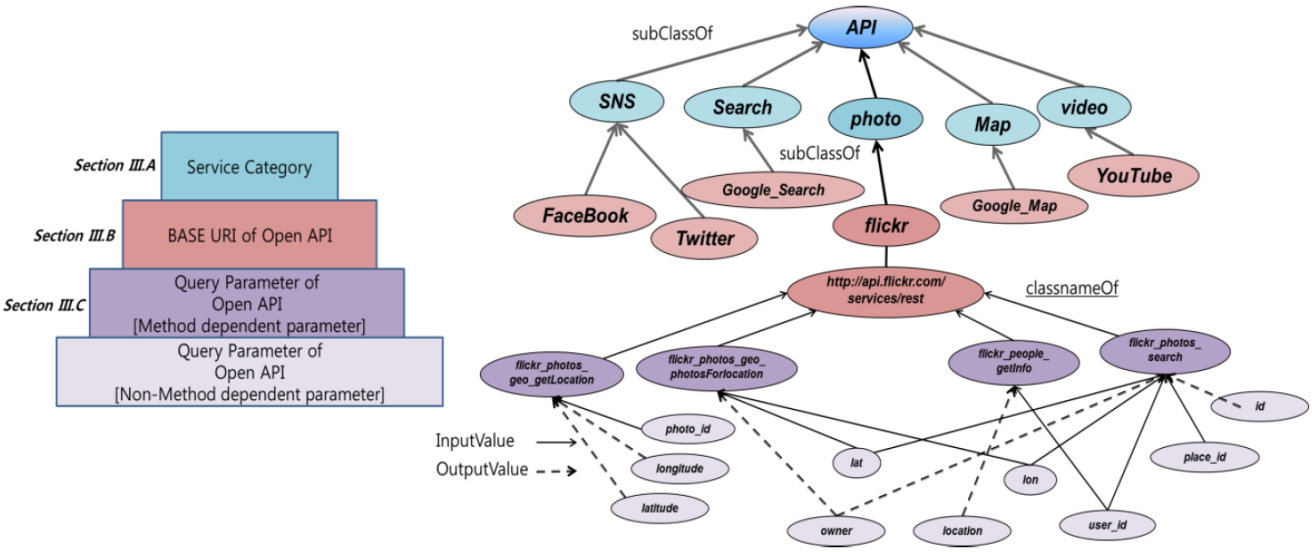

Figure 7. Hierarchical structure of ontology and example of 'flickr' API ontology

```
The Service Category
1.Photo
2.SNS
3.Map
4.Video
5.Search
 Choose the Number of the Service Category :
1
```

Figure 8. Examples for service category input

```
<photos page="2" pages="89" perpage="10" total="881">
        <photo id="2636" owner="47058503995@N01"
               secret="a123456" server="2" title="test_04"
               ispublic="1" isfriend="0" isfamily="0" />
```

Figure 9. Example XML output as response of method access

As mentioned in subsection C of Section III, because parameters make it possible to find mash-up-able element, ontology should be designed by expressing the relation between output parameter of a method and input parameter of the other method.

A. Selection of Service Category and API Search

When providing the automatic mash-up service, automatic composition algorithm needs the service category given by user as initial input in order to know what kind of service user wants. Implemented system in this research for demo, five most frequently used items (photo, SNS, Mapping, Video and Search) were tested. As shown in Fig. 8, the lists presenting service items, show service categories by searching on pre-designed ontology. After receiving service category as input from user, the system searches API in such category. If the system is given 'photo' category from the user, flickr API would be popped up.

B. API Method Search and XML Attribute of API Method

After API is determined based on the service category given by user, all of the methods related to the selected API would be searched. For example, as shown in Fig. 6, 'flickr' API has total 4 methods, 4 items are spotted. Thereafter, all of the output parameters of retrieved 4 methods would be looked up. Fig. 9 exemplified output data as response for method access. Output data is defined attribute of the returned result in XML format when requesting method using service URI of Open API. Fig. 9 shows returned result data in XML format from 'flickr.photos.search' method of flickr API. XML results has attributes such as 'id', 'owner', 'secret' in 'photo' element. So each attribute could be mapped one-to-one to output value. All the searched output parameter would be extracted by parsing after determining if there is mash-up-able method with parameters. The reason why all of the output parameter should be looked up is to find parameter category to which output parameter belongs, and to determine if mash-up is possible. In other words, parameters are the key components to determine whether the mash-up process is possible.

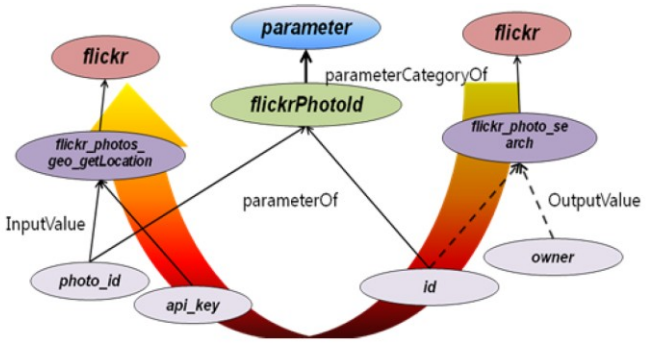

Figure 10. Example mash-up procedure between two methods by 'FlickrphotoId'

C. Decision of mash-up-ability with Method parameter and Mash-up by Data parsing

It is possible to decide whether mash-up is possible using API method parameters. After finishing the searching job for XML attribute of API method parameters, the correlation between input and output parameters of two API methods should be checked if it is appropriately related with the parameters.

As shown in Fig. 10, each Open API method parameter (XML attribute), that is also called as query parameter, has output or input value property. Decision process of *mash-up-ability* is performed as follows: first, parameter category corresponding to OutputValue property of the Open API method searched in previous subsection B is searched. Then checking process to find that there exists any InputValue property of other Open API method parameter in searched parameter category is performed. If such InputValue property of other Open API method is found, then Open API method searched in previous subsection B or upper method in hierarchy is mash-up-able with the newly found Open API method or upper method in hierarchy. In other words, if a parameter category has any API method that is correlated with input and output value, one or more methods on the upper class over such method are possible for mash-up process.

Fig. 10 demonstrates the decision process for mash-up-ability. XML attribute of 'flickr_photo_search' method that is searched in previous subsection B, are 'id' and 'owner'. However, parameters for 'owner' are omitted for clarity of the demonstration. And, the attribute 'id' is shown to belong to 'flickrphotoid' parameter category. Namely, subject 'id' has parameterOf property with 'flickrphotoid' parameter category. And Fig. 10 also shows that 'flickrphotoid' has 'photo_id' attribute in addition to 'id', who is the subject having 'parameterOf' property. By checking that 'photo_id' has inputValue property, 'flicker_photo_geo_getLocation' is found that it is the method having inputValue property. This means that attribute 'id' of 'flickrphotoid' and attribute

```
Created URI : http://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=5ac8b2580a38113a5f310a2723c6723a&place_id=Y1pL8yxTUb7rOocm
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
<photos page="1" pages="1957" perpage="250" total="489200">
        <photo id="8053198813" owner="87547839@N03" secret="a558441ef0" server="8177" farm="9" title="Tierfotografie" ispublic="1" isfriend="0" isfamily="0" />
        <photo id="8053210835" owner="87547839@N03" secret="20837b184d" server="8039" farm="9" title="Tierfotografie" ispublic="1" isfriend="0" isfamily="0" />
        <photo id="8053213761" owner="87547839@N03" secret="4d2f7468b6" server="8462" farm="9" title="Tierfotografie" ispublic="1" isfriend="0" isfamily="0" />
        <photo id="8053207072" owner="87547839@N03" secret="6afa0eacf8" server="8176" farm="9" title="Tierfotografie" ispublic="1" isfriend="0" isfamily="0" />
        <photo id="8053214466" owner="87547839@N03" secret="2a325c43d5" server="8310" farm="9" title="Tierfotografie" ispublic="1" isfriend="0" isfamily="0" />
```

Figure 11. Example of URI generation and its result

```
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
<photo id="8053198813">
        <location latitude="49.872688" longitude="8.650159" accuracy="11" context="0" place_id="y0Hg9w1XVrmm8AY" woeid="643787">
                <locality place_id="y0Hg9w1XVrmm8AY" woeid="643787">Darmstadt</locality>
                <county place_id="IXSAjAdQUL9tHXviCw" woeid="12597057">Stadtkreis Darmstadt</county>
                <region place_id="Y1pL8yxTUb7rOocm" woeid="2345485">Hesse</region>
                <country place_id="h7eZVDlTUb50Btij9Q" woeid="23424829">Germany</country>
        </location>
</photo>
</rsp>
```

Figure 12. Example results of the 'flickr_photo_geo_getLocation' method using the results from 'flickr.photos.search'

'photo_id' of 'flicker_photo_geo_getLocation' have same semantics, and two methods are mash-up-able.

If the user chooses 'flickr_photo_search' as first API method for the mash-up process, the input for query parameters is demanded. In the case of 'flickr_photo_search' method, api_key to authenticate the URI and 'place_id' are used as the query parameter input. Since each method is called using service URI, service URI need to be generated before calling the method. Also, same service URI is used to receive the return value from method. In order to generate service URI, query parameter should be added to base URI 'flickr' or 'http://api.flickr.com'. In other words, additional URI component '/?method=flickr.photos.search' is added to 'http://api.flickr.com'. Also, Query parameter would be added in order of '&', 'query name', '=', 'value'.

Fig. 11 shows the service URI that is generated based on ontology and the response result from URI. URI response result is obtained in XML format and it is used as input value to other method. Therefore, as shown in Fig. 10, 'id' value of 'flickr_photo_search' method is parsed then transferred to 'flickr_photo_geo_getLocation' method as input.

Fig. 12 shows the substitution result to 'photo_id' of 'flicker_photo_geo_getLocation' method after parsing the 'id' value of 'flickr_photo_search' method.

The 'flickr_photo_geo_getLocation' method generates the service URI using the input value from 'flickr_photo_search' method then finally obtains the mash-up result. Therefore, it is possible to mash-up two methods: 'flickr_photo_search' method that provides a simple picture searching function and 'flickr_photo_geo_getLocation' method that provides location information of a picture

V. CONCLUSION

In this paper, we implemented an automatic web service composition method, which makes it possible for general web service users to generate their own new mash-up web service automatically on their own comfort and purpose they want, using open APIs on the basis of REST. Compared to current mash-up service developed by web service developers, the proposed method can generate new mash-up service considering the intention and functionality that service user wants.

Since the detailed open API information is necessary for automatic mash-up service generation, we first designed the Open API ontology considering service category, URI, and method parameters for determining the possibility of mash-up. As far as the automatic composition method is concerned, it is designed by generating unique URI and parsing XML data returned from URI, because REST based Open APIs can be accessed only by HTTP.

As the further research, the study on ontology design and automatic mash-up method for other open APIs, which is not

able to provide services only by Java script such as Google Map API, are further required. And also, as the increasing number of API, study about automatic modeling tools for current ontology is expected to be activated in the near future.

ACKNOWLEDGMENT

This 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(2011-0025226)

REFERENCES

[1] John Musser, Tim O'Reilly "Web 2.0 Principles and Best Practices" O;Reilly Media Inc. November 2006
[2] ChangHoon Oh, " The Mash-Up Guide using Open API" , acorn publishing co. , Seoul ,2009
[3] http://www.programmableweb.com/apis/directory
[4] programmableweb : http://www.programmableweb.com/
[5] Fielding, Roy Thomas. Architectural Styles and the Design of Network based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.
[6] Y. Yohei, "The technology to support Web for web developer", Gihyo Digital Publishing . Tokyo , Japan April 2010
[7] http://www.flickr.com/services/api/
[8] Merrill, D. Mashups: The new breed of Web app. http://www-128.ibm.com/developerworks/library/x-mashups.html?ca=dgr-lnxw16MashupChallenges
[9] http://jena.apache.org/
[10] http://www.flickr.com/services/api/flickr.photos.search.html
[11] http://www.flickr.com/services/api/flickr.photos.geo.getLocation.html