An Approach for Describing Heterogeneous Web Resources in a Unified Schema

Ran Tang and Ying Zou
Department of Electrical and Computer Engineering
Queen's University
Kingston, Canada
{ran.tang, ying.zou}@queensu.ca

Abstract— **Web resources can be located and accessed through the Internet to provide information and functionalities to users. However, different types of web resources (e.g., Web Services, RESTful services and information based web pages) are represented in heterogeneous ways. It prevents the ability to discover and locate web resources with similar functionality in the service composition. In this paper, we present an approach to represent various web resources in a unified schema. Such a schema can improve the opportunity to find web resources with similar functionality, but not limiting to the same type of web resources. We develop techniques to automatically extract information from web resources. The results of the case study show that our approach can effectively describe web resources to facilitate the discovery of similar web resources among different types.**

Keywords- web resource; Web Service; HTTP API; HTML tag; descriptive tag

I. INTRODUCTION

Web resources can be located and accessed through the Internet to provide various services, such as information access and on-line banking. Web resources are described in heterogeneous formats. For example, Web Service Description Language (WSDL) is used to describe SOAP (Simple Object Access Protocol)-based Web Services to make remote procedure calls. HTTP-based APIs are simpler web resources, implemented using HTTP with universal resource identifier (URI). Examples of HTTP-based APIs are twitter, Flickr [10] and Yahoo APIs. The HTTP-based APIs can be described using web pages or WSDL. Informational websites are implemented by various technologies, such as Ajax, HTML and XML.

Similar to SOAP-based Web Services, various web resources can be discovered in the paradigm of Service-Oriented Architecture (SOA). SOA professionals select web resources of varied types [17] to compose SOA applications according to the functional and non-functional requirements. A particular functional requirement can be fulfilled by different types of web resources. For example, some service providers offer online billing service as SOAP-based Web Services and others offer similar functionality using HTTP-based APIs. The SOA professionals choose the best matched web resource to compose an SOA application. However, the chosen web resource may fail or the Quality of Service (QoS) may degrade. Such web resource needs to be replaced. It is critical for service composition and replacement to discover and locate web resources with similar functionality. To identify web resources with similar functionality, an SOA professional often manually examine the functionality by reviewing the descriptions of web resources in various formats. The lack of automation is partially due to the absence of a standard format for describing various web resources. Although a number of existing approaches (e.g., [12]) handle the automatic discovery of similar SOAP-based Web Services, such approaches are limited to discover a single type of web resources, i.e., the SOAP-based Web Services.

To assist the automatic discovery of various web resources with similar functionality, we propose a schema to uniformly describe different types of web resources. The unified representation provides a better chance to discover web resources with similar functionality than limiting the service discovery within the web resources of a single type. Moreover, we develop a technique to automatically extract information from web resources and construct the new description of each web resource using the unified schema.

The remainder of the paper is as follows. Section II gives an overview of web resources. Section III proposes a unified description schema to represent web resources. Section IV discusses the approach of extracting information from web resources and describing them using the unified description schema. Section V presents the case study. Section VI gives a survey of related work. Finally, Section VII concludes the paper and explores the future work.

II. OVERVIEW OF WEB RESOURCES

In this section, we introduce various types of web resources: web pages, SOAP-based Web Services, and HTTP-based APIs.

Web pages. A web page includes various kinds of information, such as text, images, audio and video. The information in a web page is presented in HTML format and provides navigation to other web pages via hyperlinks. Visually, a web page can be divided into three parts: header, content part and footer. The header contains common information to all web pages within one website, such as the logo of the website and menus. The content part delivers unique information. The footer includes contact and copyright information. It can be identical throughout the website. Moreover, a HTML file of a web page can carry information invisible to end-users, such as comments, meta tags, scripts

and style information. The meta tags contain the title, textual description and keywords of the web pages.

A web page is rendered by a web browser into a vivid visual format. When composing an SOA application, the web pages are used as a user interface to interact with the end-users. We categorize web pages into two groups according to their interaction with end-users: (1) Informational web pages which give factual information on a specific topic or event; and (2) Web forms which are used to collect information from users by allowing end-users to enter data and send the data to the server for processing.

SOAP-based Web Services. A SOAP-based Web Service is a self-contained software component. It communicates with other applications using SOAP over HTTP as a transport protocol. The functionality offered by a SOAP-based Web Service is described using WSDL. Specifically, as shown in Figure 1, a WSDL file contains two sections, i.e., the abstract section and the concrete section. In the abstract section, a collection of operations is defined in the "interface" part. The operation name, input parameters and output are wrapped by predefined WSDL tags. In the concrete section, the underlying transport protocol and the address of the service are defined. A client program connecting to a web service can read the WSDL to determine the available operations. The client can then use SOAP to call the operations listed in the interface specified in WSDL.

SOAP-based Web Services can handle asynchronous processing and invocation. SOAP-based Web Services are especially useful to integrate legacy systems into SOA applications. In particular, the Web Service infrastructure provides industrial standards (e.g., Web Services Reliable Messaging (WSRM)) and APIs (e.g., Java APIs for XML Web Services with the support of client-side asynchronous invocation) to ease the integration. The transaction management and security are well supported. Hence SOAP-based Web Services can be used to satisfy the complex non-functional requirements.

HTTP-based APIs. A HTTP-based API communicates with other applications using a defined set of HTTP requests and the corresponding responses. An HTTP request uses the HTTP verb and URI to indicate the functionality (i.e., operation) to invoke. The response is encoded in XML or JavaScript Object Notation (JSON) format. The response can be described in proprietary formats specified by HTTP-based APIs providers. Many leading Internet companies expose their data and functionalities as HTTP-based APIs and make them a valuable source for composing SOA applications.

Industrial efforts attempt to standardize the descriptions for HTTP-based APIs using Web Application Description Language (WADL) and WSDL2.0. However, no specification languages are widely accepted. Typically, HTTP-based APIs can be described in two ways: single-page description and multiple page description. Single-page description is used to describe an HTTP-based API with a moderate number of operations. If an API contains a large number of operations listed in a single web page, it leads to an oversized web page which hinders the transmission of the description over the Internet and makes it difficult to read. Hence, multiple-page description is used by complex HTTP-based APIs with comprehensive operations. Each web page describes one operation. A portal web page lists hyperlinks to the web pages of individual operations. The description of an operation includes the functionality, the input and the output description.

Figure 1 Example of a WSDL file for SOAP-based Web Service

RESTful services are special HTTP-based APIs. They are resource-centric as opposed to other operation-centric HTTP-based APIs. In RESTful services, no operations are explicitly defined. Each URI represents a resource associated with HTTP verbs, i.e., POST, GET, PUT, and DELETE. The HTTP verbs allow create, retrieve, modify and delete the resource. A RESTful service is suitable for representing and manipulating information. If we need to provide functionality to perform actions, such as shipping an order, it is difficult to use RESTful services. In practice, few HTTP-based APIs follow the resource-centric style, although many HTTP-based APIs are referred as RESTful services [13].

III. A UNIFIED SCHEMA FOR DESCRIBING WEB RESOURCES

To facilitate the discovery of various web resources with similar functionality, we propose a unified description schema to describe resources shown in Figure 2. The general description part represents the bibliographic information of the web resources. The operation descriptions represent the functionality of the web resources.

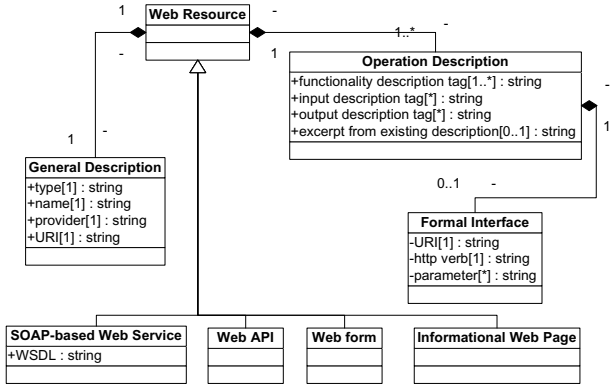

Figure 2 The unified description schema

The general description of a web resource provides a bibliographic description about the web resource, i.e., the type, the name, the provider and the URI of the web resource. Such descriptions are common to all types of web resources. There are standards suitable for representing the general description. For example, using 15 base text fields (e.g., title, type and publisher), a Dublin Core metadata schema can describe various resources, e.g., books and web pages [11]. We adopt a similar text field approach as the Dublin Core format.

Among the four fields of general description, the type, the name, the provider fields are self-descriptive. The URI field of the general description refers to the URI identifying the web resource on the Internet. The URI of a SOAP-based Web Service points to its WSDL file. For a HTTP-based API, the URI field is filled by the URI of its description web page. For web pages, the URI field refers to the web page itself. To invoke a particular functionality of the resource, another URI may be required, which is described in operation description.

The operation description describes the functionalities offered by a web resource. A web resource can deliver one or more functionalities. An operation represents a primitive unit of functionality to compose a SOA application. Different types of web resources contain varied number of operations. Most SOAP-based Web Services and HTTP-based APIs provide complex functionalities and contain multiple operations. The sole functionality of an informational web page provides information to the end-users. We treat such an informational web page as one operation. A web page may contain one or more web forms, each of which provides a particular functionality (e.g., searching or submitting information). Hence, we consider each web form as an operation. For a RESTful service, the HTTP verbs associated with the resources can be converted as operations. For example, if the resource "book" is associated with two HTTP verbs, i.e., GET and DELETE, we obtain two operations, i.e., getBook and deleteBook.

To describe each operation, we use the tag-based description, the formal interface and the excerpt of existing description. The tag-based description can concisely convey the functionality of the operation to end-users. It facilitates the machine to automatically compare the functionalities of operations in order to discover similar web resources. The tag-based description uses a set of descriptive tags (i.e., keywords) to informally represent an operation, including the functionality description, the input description and the output description. The input/output descriptions help describe the different operations with the same name. For example, two operations are named as "displayOrders". One operation with the parameter "productID" is different from the other one with the parameter "custermerID".

The formal interface is intended for machine consumption in order to facilitate automatic invocation. The formal interface provides information to support the invocation of an operation. The SOAP-based Web Service is originally described with a formal interface. Hence, a client program can be automatically generated to invoke operations in a SOAP-based Web Service. In contrast, other web resources (e.g., the HTTP-based APIs) do not have a formal interface and the SOA professional need to write the request to invoke the operation manually. In our proposed schema, the fields contained in the formal interface depend on the type of web resources. To invoke an HTTP-based API operation, the URI, the HTTP verb, and the input parameters of the operation are required. These three fields are also required for invoking a web form. Only URI is required to access an informational web page. The formal interface is described in an XML format which can be interpreted by a machine to automatically invoke the operation.

An excerpt is taken from the existing description of an operation. It provides more readable and detailed information, such as examples and demonstrations. The excerpt is available only for SOAP-based Web Services and HTTP-based APIs. It also offers a shortcut for a SOA professional to understand the operation without having to search for the operation in the entire web page.

IV. Techniques for Describing Web Resources Using the Proposed Schema

To represent various web resources in a unified format, we analyze the original description of web resources to extract the information required by the proposed schema. In the following subsections, we discuss our technique to (1) analyze the description file of the web resources to identify the operations offered by the web resources; (2) extract descriptive tags from its existing description for each operation; (3) construct a formal interface for each operation to facilitate automatic invocation; and (4) assemble the aforementioned parts of information to construct a complete description for the web resources.

A. Parse Description Files

To analyze the original description of web resources, we need to parse the description file in order to build a Document Object Model (DOM) tree structure. Web resources can be described in HTML or WSDL format. The description in WSDL file is easy to parse due to the well conformance to the WSDL specification. Unlike the WSDL file, malformed HTML file are quite common around the web. A HTML file may contain mismatched HTML tags although it can be correctly displayed by web browsers due to the fault-tolerance capability of web browsers. To generate DOM tree structure from a HTML file, we use HTML syntax checker [8] to correct the malformed HTML tags. Then we parse the HTML into DOM tree structure. For example, the HTML fragment shown in Figure 3 is converted to a DOM tree depicted in Figure 4. Each node in the DOM tree corresponds to an HTML tag.

B. Identify Operations of Web Resources

To describe web resources using the proposed schema, we need to identify their operations. The operations of the SOAP-based Web Services and informational web pages are relatively easy to identify. For SOAP-based Web Services, the <operation> tags in the WSDL file explicitly indicate the definition of the operations of a SOAP-based Web Service. Therefore, we can directly identify them using the <operation> tags. An informational web page is interpreted

as providing one operation (i.e., the "retrieve information" operation). No processing is required to analyze a web page. For the web pages containing web forms, the form tag <form> defines a web form. Therefore, each form tag, <form>, is interpreted as an operation.

It is more challenging to identify operations from the existing description of an HTTP-based API due to the lack of a machine-readable standard format to describe HTTP-based APIs. A web page describing an HTTP-based API is essentially plain text annotated by decorative HTML tags, such as paragraph tag, <p> and division tag, . HTML tags do not indicate the meaning of the enclosed plain text. For the example shown in Figure 3, the operation names are wrapped by anchor tag, <a>. However, the operation names in another HTTP-based API might be wrapped by paragraph tag <p> or other HTML tags. Due to the diversified use of HTML tags to describe the operations, a generic rule cannot be derived to identify operations using the type of HTML tags. The existing approach [1] requires human to manually identify each operation.

To automate the identification of operations, we compare the similarity among the HTML tag structures. The API provider describes all operations of an API in an identical style of appearance using server side scripting (e.g., PHP [6]). The HTML fragments for different operations have similar HTML tag structures. Figure 3 shows two similar HTML fragments corresponding to two operations: "Shop.activity.ShipOrder" and "Shop.activity.VerifyCreditCard". Figure 4 shows the tree structure of the HTML tag structures corresponding to the two operations shown in Figure 3. The left part of tree represents the operation named "Shop.activity.ShipOrder". The right part of the tree represents the operation named "Shop.activity.VerifyCreditCard". From the tree, we can observe that the HTML tag structures of the two operations are similar.

Using the similarity of HTML fragments, we design and develop an iterative algorithm to identify operations in two steps: (1) locate a set of HTML fragments with similar HTML tag structures; and (2) verify whether the located HTML fragments correspond to operations to filter out the false positives when other irrelevant content may contain similar HTML tag structures. If the identified HTML fragments are false positive, we iteratively locate another set of HTML fragments with similar HTML tag structures. The algorithm ends when we verify all the similar HTML fragments.

Locate similar HTML tag structures. To locate HTML fragments with similar HTML tag structures, we develop two algorithms to handle the single-page description and multiple-page description for HTTP-APIs due to the differences in organizing the description of operations.

A portal web page and all web pages linked by the portal web page are the input to the algorithm which is used to locate similar HTML tag structure for multiple-page description. We have no prior knowledge about whether such web pages correspond to an HTTP-based API or not. If these web pages correspond to an HTTP-based API, most of the web pages linked by the portal page should be the descriptions of individual operations. Then such web pages should have similar HTML tag structures. However, the portal web page may link to web pages in other websites. For example, a web page of an API (e.g., http://www.example.com/api) may link to the website of JSON (i.e., http://www.json.org/) as a reference. The description of individual operations should be within the same website. Hence, we filter out web pages in other domains. It is common that the header and footer of all web pages in the same web site are identical. This would mislead the algorithm to identify the header and footer as operations. We remove the headers and the footers in all web pages to avoid such a case. There are no HTML tags available in the web pages explicitly indicating the boundary of header and footer. We remove the header by comparing the beginning part of the linked page with the portal page. The identical content at the beginning is the header to remove. The same technique applied to the footer. The content part of each linked page remains. After filtering, we compare the HTML tag structure of the linked web pages. If a set of linked pages have similar HTML tag structures, they become the output of the algorithm and then are verified to determine whether such identified HTML fragments correspond to operations.

Figure 3 Example of HTML fragments of HTTP-based API operations

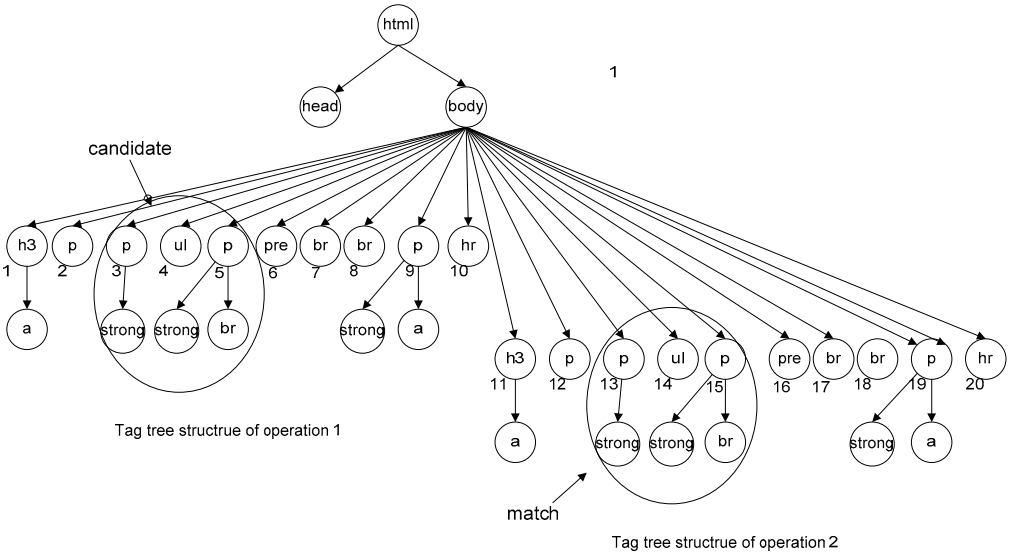

Figure 4 Tree structure of the HTML fragments

A single web page is served as the input to the algorithm that locates similar HTML tag structure for single-page description. Similarly, we have no prior knowledge about whether it is a description of a HTTP-based API. The algorithm is more complex as opposed to the algorithm for analyzing multiple-page description. In multiple-page description, only each linked page may map to an operation. The potential similar HTML tag structures are separated by different web pages. Therefore, we do not need to consider HTML fragments of varying sizes. In contrast, in single-page description, we need to enumerate HTML fragments of varied sizes as candidate to search for similar HTML tag structures.

Assume that there are n children HTML tags and these children are numbered as 1…n. The candidate structure (a→b) is an ordered collection of tags starting from the a-th child to the b-th child where 1<=a<=b<=n. For example, when we search for similar HTML tag structures among the children of the <body> tag shown in Figure 4, the candidate (3→5) is the HTML tag structure "p ul p". We can find one match for this candidate as labeled in Figure 4. However, the matching one is only a fraction of the similar HTML tag structure under the <body> tag. To avoid only detecting a fraction of similar structure, we start from the largest possible candidate containing most tags at the current level. The total number of children tags at the current level is n. Hence, we start from candidate with the size = n/2, which is the largest size for being possible to obtaining any matches of the candidate.

If similar structures are discovered among the children of an HTML tag, the identified similar structures may not map to operations because a structure can be two or more operations concatenated together. We further detect whether it can be broken down into smaller similar structures.

If there is no similar structures among the children of an HTML tag, we recursively process each child of the HTML tag using a depth-first traversal. For example shown in Figure 4, we start from the root tag in the tree structure, i.e., the <html> tag. We cannot find any similar structures among its children, i.e., the tag <head> and the tag <body>. Then we search for similar structures among the children of the tag <head> with no successes. Finally, we find the similar structures among the children of the tag <body>.

Verify HTML fragments with similar tag structure. The identified HTML fragments with similar HTML tag structures may not correspond to operations since any web page that contains a list of items (e.g., a web page introducing all albums of a singer) can contain similar HTML tag structures. We identify the following two heuristics to examine if the identified HTML fragments are operations:

(1) If the words contained in an HTML fragment are too few, it is unlikely that the HTML fragment is used to describe an operation. The shortest operation description among the HTTP-based APIs used in our case study contains 30 words. Therefore, we use 30 as the threshold.

(2) If the HTML fragment does not contain keywords related to a programming language API, such as "parameter", "argument", "response" or "output", it is unlikely that the HTML fragment is mapped to the description of an operation.

C. Generate Descriptive Tags for Operations

As described in the schema for describing operations, the descriptive tags for an operation are divided into three parts, i.e., functionality description, input description and output description. To generate descriptive tags, we extract tags from the existing description of the operation and combine with the descriptive tags retrieved from external sources, such as the social bookmarking repositories.

Extract descriptive tags from existing description. To extract descriptive tags for functionality description, input description and output description of a web resource operation, we locate the three parts in the existing description. Then we extract and process noun/verb for each part. The three parts for an SOAP-based Web Service operation can be located using WSDL tags. For the operation representing an informational web page, the functionality description is always "retrieve information" since the functionality of all informational web pages provides information to end-users without input parameter. The meta tags indicate the content or the informational web page. We treat the meta tags as the output of the "retrieve information" operation. Therefore, we use the meta tags as the output description for the operation identified from an informational web page.

The boundaries between the three parts are not explicitly defined in a HTTP-based API operation description. However, the three parts are delimited by keywords, e.g., "parameter", "argument", "response" and "output". We use these delimiter words to locate the three parts. For example shown in Figure 3, the text appearing before the delimiter word "argument" belongs to the functionality description. The text appearing after the delimiter word "response" belongs to the output description. The text in between is the description for the input parameters.

It is challenging to locate the functionality description and output description for web pages containing web forms due to the varied style and layout the web pages containing the web forms. The description for the input parameters is extracted from the HTML tag <input>.

The number of words in the textual descriptions of each part of the description may be considerably large and many of them may not be relevant. We consider that nouns and verbs are more important and keep only the verbs and nouns. Some words can be used both as noun and adjective, such as "deliverable". To find out the actual usage of a word in the sentence, we apply a Part-Of-Speech (POS) Tagger [7]. POS tagger reads text in a natural language and assigns parts of speech to each word, such as noun, verb and adjective. In this way, we correctly filter out other types of words. We treat multiple consecutive nouns as an atomic phrase, such as "music festival" which represents a unique meaning. Furthermore, we remove all stop words. Finally, we detect synonyms using WordNet [18] and keep one of them.

Collect descriptive tags from external sources. The descriptive tags that are automatically extracted may not be as accurate as the descriptive tags that are manually added by human. The social bookmarking repository contains descriptive tags for web resources created by large number of end-users. Therefore, it provides a good source to obtain high-quality descriptive tags. We use these descriptive tags to augment the extracted descriptive tags. Many social booking websites, such as delicious [5] allow the public to access their repository via published APIs. The descriptive tags for web resources can be retrieved by submitting the URI of the web resources. If a descriptive tag is both extracted from the existing description of a web resource and collected from social bookmarking repository, we mark it as a more reliable descriptive tag.

D. Construct Formal Interface for Operations

To generate the formal interface, we locate the URI, HTTP verbs and input parameters depending on the resource type. More specifically, a URI is required for invoking an operation of HTTP-based API, a web form and an informational web page. For an HTTP-based API operation, we locate the URI using string pattern matching with regular expression specified in [14]. We do not only match URI beginning with "http". In a number of API descriptions, the URIs appearing in the operation descriptions are incomplete due to the omission of the base URI in the description of each individual operation. The regular expression is also used to detect part of URI. The URI of a web form is defined in the "action" attribute of the <form> tag.

It is necessary to use the HTTP verbs to construct a HTTP request for invoking an operation of a HTTP-based API, a web form or an information web page. For an HTTP-based API operation, we locate the HTTP verbs by matching the keywords GET, PUT, POST, and DELETE. The HTTP verb of a web form can be located in the "method" attribute of the HTML tag <form>. The HTTP verb used to access an informational web page is GET.

In addition, the input parameters are needed to invoke an operation of an HTTP-based API or a web form. For an HTTP-based API operation, the description of each parameter follows the same HTML tag structures. The parameter name is usually located at the beginning of the description of the parameter. Therefore we detect parameter descriptions and extract the first word of each parameter description as the parameter names. The parameters of a web form are indicated by the HTML tag <input>.

V. CASE STUDY

We conduct a case study to evaluate the effectiveness of our approach. Specifically, objectives of the case study are: (1) verify if our approach can correctly describe web resources. The techniques for extracting information from SOAP-based Web Service, web form and informational web page have been discussed and evaluated in the existing literature (e.g., [4][16][15]). Hence, we focus on evaluating our technique that extracts information and describe HTTP-based API; (2) evaluate whether the unified description schema can help discover similar web resources of different types.

TABLE I SUMMARY OF THE EXPERIMENT SET OF OPERATIONS

	Two HTTP-based API operations	**Two SOAP-based Web Service operations**	**One HTTP-based API operation and one SOAP-based Web Service operation**	**Total**
#pairs of similar operations	6	4	15	25
# pairs of dissimilar operations	49	41	95	185

TABLE II RESULT OF IDENTIFYING OPERATIONS OF HTTP-BASED APIS

Category	#Total web pages	#Web pages containing operation descriptions	#Web pages from which operations are identified	#Web pages from which operations are correctly identified	Precision	Recall
Enterprise	30	9	9	9	100%	100%
Communication	30	8	7	7	100%	87.5%
Internet	30	11	9	9	100%	81.8%
Others	30	9	8	8	100%	88.9%
Total	120	37	33	33	100%	89.2%

A. Setup

We collect 37 web pages that contain the existing descriptions of HTTP-based APIs from different domains, such as enterprise application domain, communication domain, and Internet application domain. The providers of the HTTP-based APIs are from different sectors, such as government agencies, IT companies, and individual developers. We avoid selecting many HTTP-based APIs from the same providers to ensure the case study result is not skewed due to the particularity of a provider. We group the web pages into categories as specified in [13]. To test whether our approach can distinguish web pages that contain the description of HTTP-based APIs from other web pages, we also collect 83 web pages containing no description of HTTP-based APIs. In each category, we mix the web pages containing descriptions of HTTP-based APIs with the web pages that does not contain description of HTTP-based APIs.

To evaluate the effectiveness of the unified description schema for discovering similar operations from different types of web resources, we extract and prepare an experimental set of operations. Table I gives a summary of the set of operations. They are extracted from different types of web resources and are represented using the unified schema. The experimental set contains similar and dissimilar operations from different types of web resources.

B. Evaluation Criteria

We measure the effectiveness of our approach on identifying operations using precision and recall. Precision is defined in equation (1). It measures if any irrelevant web page content is misidentified as the description of operations. Recall is defined in equation (2). It evaluates whether our approach can correctly identify all web pages that contain HTTP-based API operations without omissions.

$$precision = \frac{\#\text{correctly identified API descriptions}}{\#\text{identified API descriptions}} \quad (1)$$

$$recall = \frac{\#\text{correctly identified API descriptions}}{\text{total \# of API descriptions}} \quad (2)$$

We also define the criteria to evaluate if our approach can identify the similar operations. If matches are found in functionality description tags, input description tags and output description tags of two operations, the two operations are considered similar. Otherwise the two operations are considered not similar. We measure the effectiveness of discovering similar operations using accuracy and coverage. Accuracy is defined in equation (3), which measures the percentage of the correctly identified similar operations. Coverage is defined in equation (4). It measures whether our approach can find all similar operations.

$$accuracy = \frac{\#\text{correctly identified pairs of similar operations}}{\#\text{identified pairs of similar operations}} \quad (3)$$

$$\text{coverage} = \frac{\#\text{correctly identified pairs of similar operations}}{\text{total \# of pairs of similar operations}} \quad (4)$$

C. Analysis and Discussion of Results

The results for identifying HTTP-based API operations are shown in Table II. The high precisions show that our approach can correctly identify operations for HTTP-based APIs descriptions. Our approach does not misidentify any web pages that contain no HTTP-based API operations. The recalls show that our approach may fail to recognize web pages that contain HTTP-based API operations. An investigation shows that the HTML tag structures of operations are different in some HTTP-based APIs. Such HTTP-based API descriptions are hand-crafted web pages. Therefore, there is no similar HTML tag structure in such HTTP-based APIs. However, most HTTP-based API descriptions are automatically generated using server side scripting. This should not affect the usefulness of our approach.

Table III lists the result to show the ability of our approach to discover similar operations. The accuracy is very high, i.e., 100%. To qualify the similarity between operations, we need to match the descriptive tags of functionality, the input and the output. This strict requirement helps avoid false positive. The recall shows that our approach misses some pairs of similar operations, especially in the cases involving HTTP-based APIs due to the difficulty in accurately extracting descriptive tags from the existing description written in natural language.

TABLE III RESULT ON DISCOVERING SIMILAR OPERATIONS

	Two HTTP-based API operations	Two SOAP-based Web Service operations	One HTTP-based API operation and one SOAP-based Web Service operation	Total
Accuracy	100%	100%	100%	100%
Coverage	66.6%	100%	86.7%	84.0%

VI. Related Work

Metadata. Much research has been conducted to derive a metadata schema for describing certain types of resources on the Web. The working group of the Internet Engineering Task Force (IETF) proposes Internet Anonymous Ftp Archive (IAFA) templates [2] to allow effective access to FTP archives. This template describes the contents available in the archive. Another metadata schema Dublin Core [3] describes network electronic information. Specifically, a Dublin Core metadata record can describe books, video, sound, image, text files, and web pages. These schemas have the potential of describing different types of web resources. However, they focus on representing the bibliographic information of resources and have limited capacity to describe functionality accurately. Our proposed schema is different from the aforementioned schema. We use the tag-based description and the formal interface to facilitate the representation of operations, which more accurately represents the functionality of web resources.

Extraction of information from web resources. Little research has been conducted on extracting information from HTTP-based APIs. Maleshkova et al. propose SWEET [1] to enable developers to identify different parts (e.g., operations, input and output) of the API. However, current approach of identifying operations is mostly manual, with limited tool support. Our approach increases the level of automation for the identification of operations. Raghavan and Garcia-Molina [16] discuss the techniques to extract information from HTML form (i.e., web form). Craven [15] proposes techniques for processing information in the meta tags of a web page. Xiao et al. [4] describe an approach to extract descriptive tags from WSDL file for SOAP-based Web Services. These approaches are complementary to our technique for extracting information and describing HTTP-based APIs. We use the techniques collaboratively to describe all types of web resources.

VII. Conclusion

Web resources are described in heterogeneous formats. Web resources are discovered and selected to compose applications in the paradigm of the SOA. In the life cycle of an SOA application, component web resources may need to be replaced. The heterogeneous formats of describing web resources hinder the ability to discover similar web resources. To identify web resources with similar functionality, a SOA professional needs to manually examine the functionality by reviewing the descriptions of different web resources in various formats. To assist the automatic discovery of various web resources with similar functionality, we propose a schema to uniformly describe different types of web resources. The uniformed representation provides a better chance to discover web resources with similar functionality rather than limiting the service discovery to only one type of web resources. Moreover, we develop techniques to automatically extract information from web resources and construct new description of each web resource using the proposed unified schema. The cases study show that our approach can effectively extract information from HTTP-based API descriptions and discover similar operations from web resources of different types.

There are several limitations of our work. (1) We rely on the availability of execution logs to mine service composition patterns. It can be challenging to collect a large amount of execution logs produced by multiple service-oriented applications in a production environment. Our approach assumes the users have the access to the execution logs of multiple applications. However, the access to execution logs may be restricted due to confidentiality consideration and other concerns. (2) The unified schema does not include all type of web resources. In the future, we plan to study more types of web resources (e.g., RSS). We also want to conduct a larger case study with more web sources collected from the Internet.

References

[1] Maleshkova, M.; Pedrinaci, C.; and Domingue, J. 2009.Supporting the creation of semantic restful service descriptions. In Workshop: Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2) at 8th International Semantic Web Conference

[2] Deutsch, P., A.E. Bunyip, M. Koster, and Stumpf (1995), "Publishing Information on the Internet with Anonymous FTP." http://info.Webcrawler.com/mak/projects/iafa/iafa.txt.

[3] Weibel, S., J. Grodby, and E. Miller (1995), "OCLC/NCSA Metadata Workshop Report," Dublin, EUA. http://www.oclc.org:5046/oclc/research/conferences/metadata/dublin core report.html.

[4] Xiao, H, Zou, Y., Tang, R., Ng, J., Nigul, L., An Automatic Approach for Ontology-Driven Service Composition Proc. IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2009) , Taipei, Taiwan, 14-15 December 2009

[5] Delicious, http://delicious.com/

[6] PHP, http://php.net/

[7] Stanford Log-linear Part-Of-Speech Tagger, http://nlp.stanford.edu/software/tagger.shtml

[8] JTidy, http://jtidy.sourceforge.net/

[9] Zhao, H. and Doshi, P., Towards Automated RESTful Web Service Composition, 2009 IEEE International Conference on Web Services

[10] Flickr API, http://www.flickr.com/services/api/

[11] Dublin Core Metadata Initiative, http://dublincore.org/

[12] Sycara, K., Paolucci, M., Ankolekar, A., et al.: Automated discovery, interaction and composition of semantic web services. Journal of Web Semantics 1, (1), 27–46 (2003), December

[13] Programmable web, http://www.programmableweb.com/

[14] Uri - Regular Expression Library, regexlib.com/

[15] Craven, T. C, HTML Tags as Extraction Cues for Web Page Description Construction, Informing Science: International Journal of an Emerging Transdiscipline. Vol. 6, pp. 1-12. 2003

[16] Raghavan, S., Garcia-Molina, H., Crawling the Hidden Web, Proceedings of the 27th International Conference on Very Large Data Bases, p.129-138, September 11-14, 2001

[17] JOpera, http://www.jopera.org/

[18] WordNet, http://wordnet.princeton.edu/