Migration of SOAP-based Services to RESTful Services

Bipin Upadhyaya, Ying Zou
Dept. of Electrical and Computer Engineering
Queen's University
Kingston, Ontario, Canada
{9bu, ying.zou}@queensu.ca

Hua Xiao
School of Computing
Queen's University
Kingston, Ontario, Canada
huaxiao@cs.queensu.ca

Joanna Ng, Alex Lau
IBM Canada Laboratory
Markham, Ontario, Canada
{jwng, alexlau}@ca.ibm.com

ABSTRACT
Web services are designed to provide rich functionality for organizations and support interoperable interactions over a network. Web services are mainly realized in two ways: 1) SOAP-based services and 2) RESTful services. For the service providers, RESTful services can improve system flexibility, scalability, and performance as compared to the SOAP-based Web services. It is equally attractive to end users as it is consume less resources (i.e., battery, processor speed, and memory). Additionally, REST-based services do not include complex standards and heterogeneous operations; and hence are easier to consume and compose as compared to SOAP-based Web services. We provide an approach to migrate SOAP-based services to RESTful services. We identify resources from a SOAP-based Web service by analyzing its service description and mapping the contained operations to resources and HTTP methods. To demonstrate the effectiveness of our approach, we conduct a case study on a set of publicly available SOAP-based Web services. The results of our case study show that our approach can achieve high accuracy of identifying RESTful services from the interfaces of SOAP-based services. Our approach can improve the performance for invoking Web services after SOAP-based services are migrated to RESTful services.

Keywords: Service Migration, RESTful Services, SOAP Services, Thin Clients

1. INTRODUCTION
Mobile Internet is driving mobile devices growth exponentially faster than previous computing technologies. Mobile Internet devices exceeded ten billion units in 2010 [29]. With the increasing use of mobile devices, mobile applications will generate a larger percentage of Web service requests. Despite the fact that the condition of mobile computing has largely improved in recent years [26, 27], applying traditional Web services (i.e., SOAP-based services) models to mobile computing may result in unacceptable performance overheads. There are several challenges in the process of consuming traditional Web services from mobile clients. Cell networks have limited bandwidth and are often billed based on the amount of data transferred. However, even a simple SOAP [33] message often contains a large chunk of XML data, which consumes a lot of bandwidth and the transmission can cause major network latency. In addition, the SOAP message contains mostly XML tags that are not all necessary for mobile clients. Mobile clients are thin clients [23] with limited processing power. The limitations are intrinsic to mobility and not just the shortcomings of current technology [26, 28]. In addition, many mobile platforms do not include necessary libraries for SOAP-based services. SOAP-based services are heavy-weighted services which are not applicable for mobile services in comparison to light weighted RESTful services. Encoding and decoding of XML-based SOAP messages consumes resources (i.e., battery, processor speed, and memory).

REST is an architectural style derived from the Web, and its architectural elements and constraints aim at collecting the fundamental design principles that enable the great scalability, growth, and success of the Web. A RESTful service is provided as a resource which is meaningful concept and can be addressed in the Web. RESTful services use existing features of HTTP protocol. The communication between requests and responses are built around the transfer of representations of resources. The advantages of RESTful services over SOAP-based services are simplicity, interface flexibility, interoperability, and scalability [22]. The features of HTTP (e.g., caching, authentication, and content type negotiation) can be fully utilized by RESTful services. REST resources can easily interact with other Web resources (e.g., Web pages) using an HTTP library. Due to these benefits many of the service providers, such as Google, Yahoo, Amazon, and eBay have adopted RESTful services [30]. Pautasso et al. [6] studied the architectural decisions to make between the SOAP-based and RESTful services. The study found that the developers make less architectural decisions in REST-based services.

Migration of SOAP-based services to RESTful services makes the services more pervasive, faster, and suitable for thin clients. In addition, RESTful services are invoked using HTTP methods. Hence it is easy for non-professional developers (i.e., users) to consume and compose services using REST-based approach [5]7]. In this paper, we provide a framework to migrate SOAP-based services to RESTful services. More specifically, we analyze the WSDL documents to identify resources and their corresponding HTTP-methods (i.e., GET, POST, PUT, and DELETE). Our approach automatically generates the configuration files needed to deploy the RESTful services, and wrappers for accessing SOAP-based services in the REST architecture. To enable the interaction between SOAP-based services and RESTful services, we establish a mechanism that dynamically converts messages between both types of services. Coexistence of the RESTful and WSDL-based services help service providers to support the old customers using WSDL-based services and new emerging customers who prefer RESTful services.

The remainder of the paper is organized as follows. Section 2 introduces some background information. Section 3 presents our approach for migrating SOAP-based services to RESTful services. Section 4 gives an overview of the prototype. Section 5 describes the case study and threats for validity. Section 6 discusses the related work and finally Section 7 concludes the paper.

2. BACKGROUND

In this Section, we discuss some background knowledge on SOAP based services and RESTful services, necessary to understand the migration process.

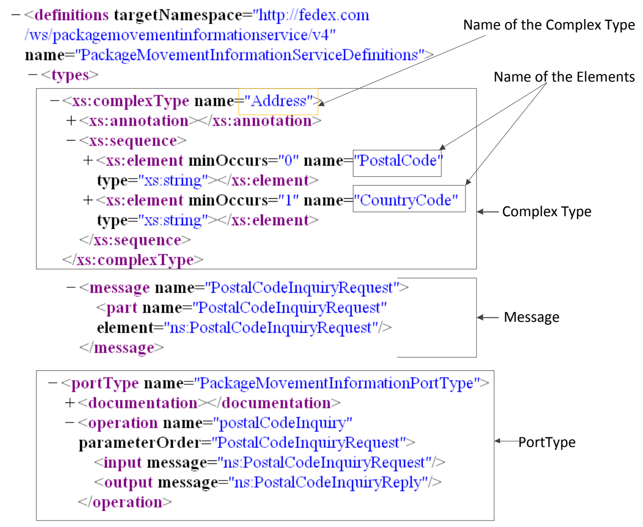
2.1 SOAP and WSDL

Web Service Description Language (WSDL) is used to describe the service interfaces [34]. More specifically, WSDL builds on XML Schema by making it possible to fully describe Web services in terms of messages, operations, interfaces, bindings, and service endpoints. A service can support multiple bindings for a given interface, but each binding should be accessible at a unique address identified by a Universal Resource Identifier (URI), also referred to as a Web service endpoint. WSDL is an interface centric model. A Web service client must know the communication protocol to use for sending messages to the service, along with the specific mechanisms involved in using the given protocol, such as the use of commands, headers, and error codes. The WSDL binding element describes the details of using a particular *portType* with a given protocol. A binding also influences the way that abstract messages are encoded on the wire by specifying the style of the service (i.e., document or RPC) and the encoding mechanism (i.e., literal vs. encoded). Figure 1 shows different parts of a WSDL document. The names used in the WSDL document (i.e., the name of the complex-type, the name of the elements) are very important from the service providers' point of view. The names used in a message are the same within a WSDL document or a number of WSDL document from the same service provider.

SOAP is a messaging layer protocol independent of any underlying transport protocols, such as HTTP [33]. SOAP ignores the semantics of the operations of underlying protocols. When HTTP is used to transfer a SOAP message, it is tunneled through a POST operation. For example, *getCapital()* function in Figure 2(a) could be GET, PUT or DELETE. The argument is that in order for SOAP to correctly leverage the HTTP protocol in a RESTful manner, the operation in Figure 2(b) would actually be bound to the HTTP GET operation, not an HTTP POST as required by the SOAP/HTTP binding. Figure 2(a) and 2(b) show the differences between service invocation in RESTful and SOAP-based services. SOAP ignores the semantics of the underlying protocol like HTTP and contains extra XML content in the request.

2.2 RESTful Services

REST is an architectural style for network-based systems [21]. In REST approach, resources are consumed by the clients using HTTP methods. A resource is accessed via a Universal Resource Locator (URL). The state of a resource is transferred using its representation. A resource provides a set of design constraints, such as identification of resources, manipulation of resources through representations, self-descriptive messages, and the use of hypermedia as the engine of application state [22]. Such constraints are intended to make services scalable, reliable, reusable, resilient, and other desired features of the Web as a network-based system. RESTful services are self-descriptive and hypermedia control, which allow us to treat an application as a state machine. For example, for the request of getting a blog post, the response embeds URIs to create a comment and edit the post. Therefore, an application can be considered as a state machine with a page representing a state and links representing every possible transition from the current state. WSDL2.0 and Web Application Description Language (WADL) [35] can be used to describe RESTful services. RESTful services must fulfill the following concepts of resource, representation, and unified interface.



```
- <definitions targetNamespace="http://fedex.com
  /ws/packagemovementinformationservice/v4"
  name="PackageMovementInformationServiceDefinitions">
  - <types>
    - <xs:complexType name="Address">
      + <xs:annotation></xs:annotation>
      - <xs:sequence>
        + <xs:element minOccurs="0" name="PostalCode"
          type="xs:string"></xs:element>
        + <xs:element minOccurs="1" name="CountryCode"
          type="xs:string"></xs:element>
        </xs:sequence>
      </xs:complexType>
    - <message name="PostalCodeInquiryRequest">
        <part name="PostalCodeInquiryRequest"
        element="ns:PostalCodeInquiryRequest"/>
      </message>
   - <portType name="PackageMovementInformationPortType">
     + <documentation></documentation>
     - <operation name="postalCodeInquiry"
       parameterOrder="PostalCodeInquiryRequest">
         <input message="ns:PostalCodeInquiryRequest"/>
         <output message="ns:PostalCodeInquiryReply"/>
       </operation>
```

Figure 1: Different parts of WSDL document

Resource

A resource is a relevant abstraction within the domain. The service designer may choose any domain objects, from concrete to abstract ones. Resources are unique and identified using URL (e.g., http://foo.org/customers/121)

Representation

Theoretically, a representation is any useful information about the state of a resource. Technically, a representation is a resource serialization in a given format, such as XML, JavaScript Object Notation (JSON), and Resource Description Framework (RDF).

Table 1: Different HTTP-method and their characteristics

Method	Safe	Idempotent
POST	X	X
GET	O	O
PUT	X	O
DELETE	X	O

Unified Interface

In the RESTful paradigm, an action is defined by a HTTP method. The HTTP protocol mainly offers four principal methods, namely GET, POST, PUT, and DELETE. A HTTP method can have two properties:

i. safe property means that the invocation of a HTTP method does not cause modifications to any resources; and
ii. idempotent property means that the side effect of one or more than one identical requests is the same as a single request.

Despite the simplicity of REST, the properties are extremely powerful, since REST defines a unified interface to all possible services. Table 1 lists the properties of HTTP methods. If a client knows the resources offered by a given service, the client can automatically know how to retrieve, create, update, and delete these resources.

```
GET /country/capital?x0=Nepal HTTP/1.1
Host: 130.15.19.65:8080
Accept: application/xml
```

(a) RESTful Request

```
POST /country/capital HTTP/1.1
Host: 130.15.19.65:8080
Content-Type: application/soap+xml
<? xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope        xmlns:soapenv="http://schemas.xmlsoap.org/
soap/envelope/"    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <soapenv:Body>
        <GetCapital xmlns="http://org.myservice">
        <country>Nepal</country></GetCapital>
        </soapenv:Body>
</soapenv:Envelope>
```

(b)Soap based Request

Figure 2: Requests in Operation-oriented and resource-oriented service Invocation

3. STEPS FOR MIGRATION

Figure 3 shows the overall steps for migrating SOAP-based services to RESTful services. Initially, we analyze the WSDL document of a SOAP-based service and build a dependency graph which describes the relations between operation names, input and output parameters of an operation defined in the interface of a SOAP-based service. We identify and group similar operations from the dependency graph. Each cluster of operations is analyzed to identify resources and the HTTP methods associated with the resource. Furthermore, we manually refine the identified resources and HTTP methods. Once the resources are validated and verified by a user, the wrappers and configuration files are automatically generated. The migrated RESTful service is ready for deployment.

3.1 Identification of Similar Operations

Each resource in a RESTful service is associated with four opeations (i.e., GET, PUT, POST and DELETE). We need to identify the operations that manipulate the same resource. We build a dependency graph which connects input and output parameters of each operation along with the semantics (i.e., nouns found in an operation-name) of an operation. Figure 4(a) shows the dependency graph between operations, input and output parameters. A circle denotes an operation. A square represents an input or an output parameter. A triangle means a noun in the name of an operation. For the operations and parameters, the edges pointing to the operations denote message input and the edges pointing to the parameters represent the message output.

Cluster analysis is the process of organizing objects into groups whose members are similar in a certain way [3]. We use cluster analysis to group the operations that manipulate the same resource. Each cluster represents a resource. Cluster analysis helps to narrow down the number of resources to be identified and helps to identify the HTTP methods related to the resource.We use the distance metrics between the operations to similar operations. The distance metric is defined in Equation (1). Similarity between two operations is calculated as the ratio of the sum of common parameters and the common nouns in the name of the operation to the sum to the total number of parameters and the total number of nouns in the operation name.

We group operations with the value of the simlarity metric greater than or equal to some threshold value. We choose the threshold value by manually varifying different threshold values for the same WSDL document. Given a dependency graph, we form the clusters of similar operations. Figure 4(b) shows the three different clusters identified from the dependency garph shown in Figure 4(a). For example, the function *getCapital* in Figure 4(a) takes input as *country* and returns *capital* as a result. The noun of the name of the function is *capital*. Similarly, the function *addCapital* has *country* and *capital* as input and *status* as output. Using Equation (1), the similarity between *addCapital* and *getCapital* is

$$similarity(addCapital, getCapital) = \frac{2 * (2 + 1)}{3 + 2 + 1 + 1} = 0.85.$$

As operations, *addCapital* and *getCapital* fall above threshold value, they belong to the same cluster. We perform a similar approach on all operations and group the similar operations. Based on this criterion, the cluster of operations dealing with the same resource is formed as shown in Figure 4b. We analyze each cluster to find the name of the resource and the HTTP method associated with it. The next state is shown by the links between the operations. The clusters show the element that can be included as the link. For example, when a user calls a service to add the city, the link about updating the city, getting all the cities of that country and deleting city links are provided to the user.

3.2 Identification of Resources

Each resource is uniquely identifiable by a URI. Nouns are used to represent resources. The maximum of four operations (i.e., *GET, PUT, POST, and DELETE*) can be associated with each resource. For each cluster, we identify the name for the resource. Natural language processing (NLP) provides the techniques to find the root word by stemming [9]. NLP also provides techniques to identify part-of-speech (POS) to identify lexical categories (e.g., nouns, verbs, and adjectives). We filter nouns and ignore other lexical forms. Input and output parameters represent the data entities and are considered as nouns. Semantic relationships between the operations, input parameters, and output parameters help to link words to form the resource. We use WordNet [10] to find relationship between words. WordNet is a rich set of lexical knowledge. It defines different kinds of relationships between the words.

- **Hypernym** represents a “kind of” relation. For example, car is a hypernym of vehicle. A hyponym relation is converted as a subclass relation in the ontology.
- **Hyponym** means that a word is a super name of another. For example, vehicle is a hyponym of car. Hyponym is the inverse of hypernym.
- **Holonym** describes a whole-part (*i.e.*, partOf) relation. For example, city is a holonym of country.
- **Meronym** is the inverse of holonym and represents part-whole relation. For example, window is a meronym of building.

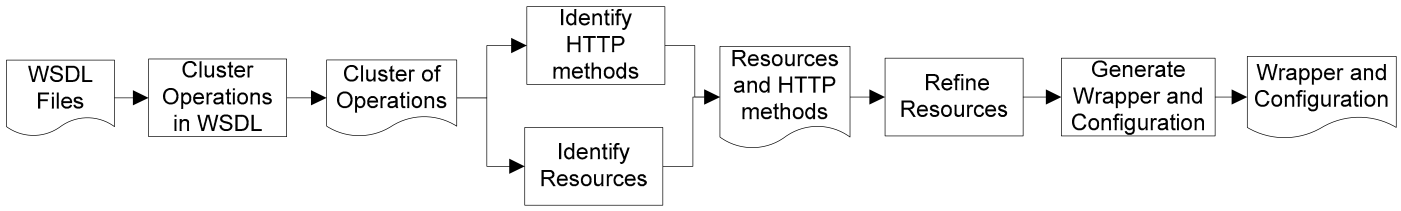
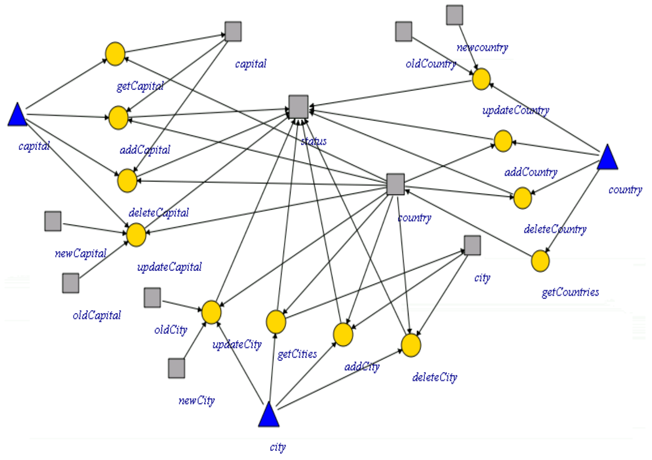

WSDL Files
Cluster Operations in WSDL
Cluster of Operations
Identify HTTP methods
Identify Resources
Resources and HTTP methods
Refine Resources
Generate Wrapper and Configuration
Wrapper and Configuration

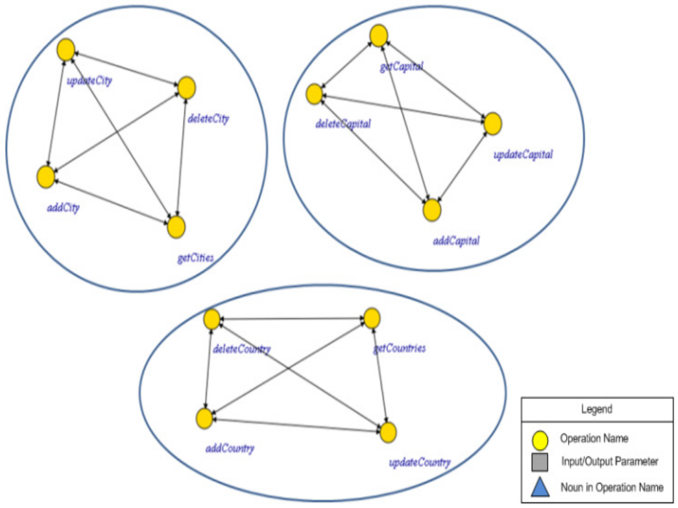
Figure 3: Overall approach of identifying the resources

$$similarity(x,y) = \frac{2 * \left(common_{i/o}(x,y) + common_{noun}(x,y)\right)}{Total_{i/o}(x) + Total_{i/o}(y) + Total_{noun}(x) + Total_{noun}(y)} \qquad (1)$$

where x, y are the operation., $common_{i/o}(x,y)$ is the number of common input and output parameters between x; and $common_{noun}(x,y)$ is the number of common nouns between name of x and y, $Total_{noun}(x)$ is the total noun in the name of operation x, $Total_{i/o}(x)$ is total input and output parameter of operation y.



(a)Dependency graph



(b) Clusters of Operation

Figure 4: Clustering operation based dependency graph

For each cluster as shown in Figure 4(b), we separate words in the operation name, remove the stop words (e.g., a, an, and the), use stemming to find the root word and identify the lexical categories to filter nouns. Each clustered words are categorized into three sections: nouns used in the name of the operation (i.e., called as semantic-relation), the input parameters and output parameters of the operation. We use the rules listed in Table 2 to decompose the words in an operation name.

Table 2: Rules used to decompose the words

Rule	Name	Word
CaseChange	FindCity	Find, City
CaseChange	addCountry	add, Country
Suffix Containing Number	City1	City
Underscore Separator	Find_city	Find, City

From the initial set of words, we choose the common words from the semantic part of an operation, the input parameters and the output parameters of the operation. For example shown in Table 3, we take the cluster *{addCity, updateCity, getCities, deleteCity}* shown in Figure 4(b). The operations contain common word, i.e., *city*. The input parameters have the word, *country* in common. Therefore, the initial set of words is *{city, country}*. We organize these words to form a resource. The organization is done on the basis of the position of the word in the name of an operation, the semantic relations between the words. For example, parental words (e.g., country in *{country, city}*), words that already are resources are given a higher priority as shown in Figure 5. The following steps rank the elements in operation-name and input-output parameters.

1. Rank the words according to the place of occurrences. Words in a semantic region are given the highest priority, input parameters as second and output parameters as third.
2. Check if the word is already used to name a resource. If so, such a word is given a higher priority.
3. Check the relations between the words and prioritize them according to the relationship.

4. Combine the words in the initial set to form the resource.

Table 3: Semantic, Input Parameters and Output Parameters in a cluster

	getCity	addCity	deleteCity	updateCity
Semantic	city	city	city	city
Output Parameters	city	Status	Status	Status
Input Parameter	country	{city, country}	{city, country}	{country, oldCity, newCity}

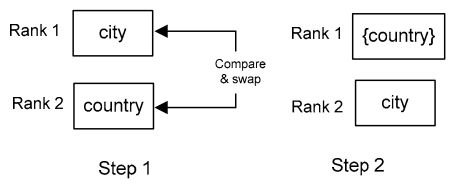


Figure 5: Ranking between different words

For the name of the operation that is not able to categorized, we put the name of the operation in the semantic region and form the URL for the operation as */operation-name/{input}* and turn the flag for editing. The editing flag helps the user to clearly distinguish such resources before generating REST equivalent services.

In case of more than one word for the same position in the resource name hierarchy, we examine the semantic relations between the words and the frequency of words in the WSDL document. For example, two words that exist as resources have the same priority. We use semantic relations between the words to identify the position. If no relation exits, we order the words according to the frequency of the word in the WSDL document. To illustrate the process, let us take a cluster shown in Figure 4(b). Table 3 shows the semantic, input, and output parameters in the cluster. The common elements among them are *{city, country}*. A *city* is a-part of country. Therefore, there is a semantic relation between *city* and *country* and *country* comes before *city* in the resource name. Since *country* occurs in input parameters, we denote *country* within *{}*. Hence, the resource for this cluster is *{country}/city*.

3.3 Identification of Resource Methods

In this step, we identify allowable HTTP-methods for each resource. Ideally, each cluster should contain four operations that are mapped to a single resource and four HTTP-methods. Resource methods are identified by analyzing the verb part of the operation name. We identify the HTTP method associated with operation by analyzing the semantics of the operation name. If HTTP methods cannot be identified by analyzing the semantics of an operation name, we calculate the fan-in and fan-out of the operations. Fan-in and fan-out give the number of parameters taken as an input and the number of the output parameters given by the operation. If the ratio of fin-in to fan-out is less than one, we assume the operation is the retrieval of a resource and hence is associated to GET. If the ratio of the fin-in to the fan-out is less than one, we assume it as the modification or the creation of a resource and further encourage the user to validate the judgment. It is important to note that a resource cannot have the repetition of the same method name.

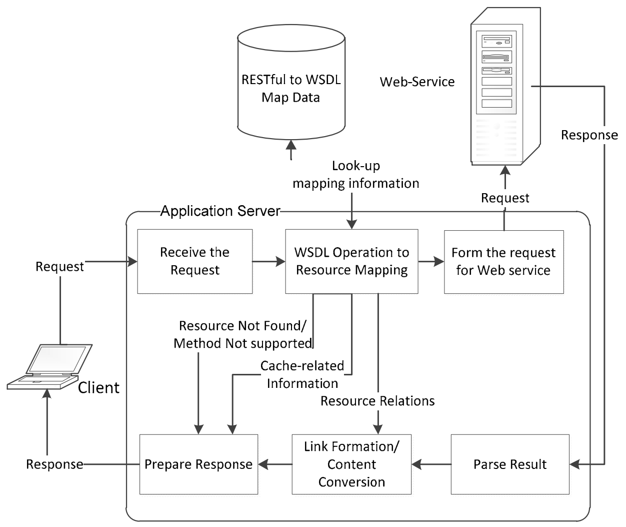

RESTful to WSDL Map Data
Web-Service
Response
Look-up mapping information
Request
Application Server
Request
Receive the Request
WSDL Operation to Resource Mapping
Form the request for Web service
Resource Not Found/ Method Not supported
Cache-related Information
Resource Relations
Client
Response
Prepare Response
Link Formation/ Content Conversion
Parse Result

Figure 6: Process of transforming messages between the services

In case where two operations in the same cluster have the same HTTP method, we revisit the resource identification to check if the cluster contains two different resources. For example, in case of two operations *getLatestBlogs()* and *getBlog(),*our approach may identify the same resource *blog* and the same method *GET*. In such case we revisit the resource identification process and include the excluded part in the semantics of the operation name called the latest to form a new resource *blog/latest* for *getLatestBlogs()*. If different resources cannot be formed, we tunnel the operation through the POST attaching the name of the operation in the resource. POST is considered unsafe and idempotent. It is the best to use this method when we are not sure which method can be associated with the resource. Such resources are flagged, so that it can be easily identified in the user interface and allows the service provider to modify it.

For example shown in Figure 4(b), the cluster *{getCity, addCity, deleteCity, updateCity}* contains the resource identified as *{country}/city*, we analyze the semantic meaning of the operation and identify the corresponding resource methods (i.e., *GET*, *POST*, *DELETE*, *and PUT*). There is no conflict. The exact match is found, and hence the fan-in and fan-out of the operation are not calculated.

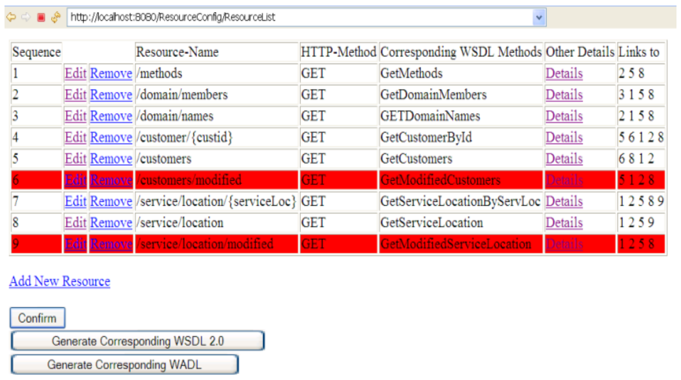
3.4 Message Conversion

Our migration approach utilizes the SOAP-based service and gives it the capability of RESTful services. Our approach does not affect the operation of traditional services. Instead, all the RESTful requests are converted to corresponding SOAP-based service operations. We convert the message back and forth. The RESTful client gives the input as HTTP parameters which are converted to the SOAP-based service format. Correspondingly, the output of the SOAP-based service is converted into the one as required by the client in the content-type attribute of a HTTP header. For example, we convert the input shown in Figure 2(a) to Figure 2(b) before sending it to a Web service. The link formation is completed at this stage to guide the client to the next state. Figure 6 shows the framework to convert the message passing between a SOAP-based Web service and RESTful services.

A SOAP operation is looked up by the "WSDL operation to Resource mapping" module shown in Figure 6 for the resource requested. The mapped SOAP-based operation is invoked and the received result is parsed and changed to a web form as requested by client in content-type header. The framework uses the standard HTTP Error messages if error occurs. For example, if a user uses a HTTP method that is not supported by the resource, the error called "405 Method Not Allowed" is generated. The "Prepare Response" module in the framework shown in Figure 6 is responsible for correct response code and adding the cache related headers in the response. If the information regarding resource relationships is available, the link formation between resources is prepared in this step.

4. PROTOTYPE

Ambiguities among resources need to be removed. Such ambiguities include more than one resource competing for the same name or HTTP methods. In addition, our approach may misidentify the resource or HTTP-methods. A user can manually validate and modify the resource, HTTP method and cache related information. HTTP has the standard way for caching validation using Entity tag (E-Tag) which allows a client to make a conditional request. The service provider may additionally set a header called "expires" to denote the content is stale after the given time frame.



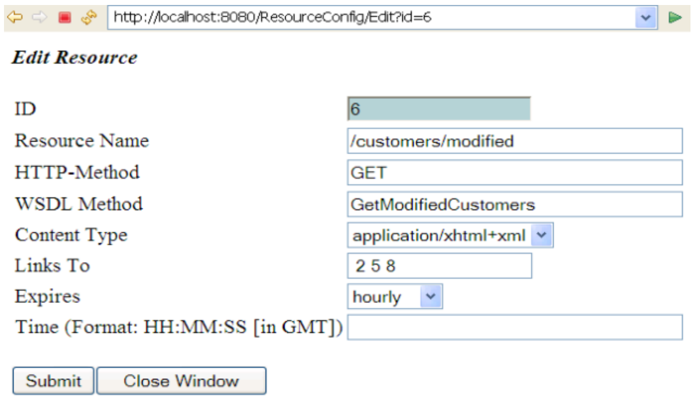
http://localhost:8080/ResourceConfig/ResourceList

Sequence			Resource-Name	HTTP-Method	Corresponding WSDL Methods	Other Details	Links to
1	Edit	Remove	/methods	GET	GetMethods	Details	2 5 8
2	Edit	Remove	/domain/members	GET	GetDomainMembers	Details	3 1 5 8
3	Edit	Remove	/domain/names	GET	GETDomainNames	Details	2 1 5 8
4	Edit	Remove	/customer/{custid}	GET	GetCustomerById	Details	5 6 1 2 8
5	Edit	Remove	/customers	GET	GetCustomers	Details	6 8 1 2
6	Edit	Remove	/customers/modified	GET	GetModifiedCustomers	Details	5 1 2 8
7	Edit	Remove	/service/location/{serviceLoc}	GET	GetServiceLocationByServLoc	Details	1 2 5 8 9
8	Edit	Remove	/service/location	GET	GetServiceLocation	Details	1 2 5 9
9	Edit	Remove	/service/location/modified	GET	GetModifiedServiceLocation	Details	1 2 5 8

Figure 7: Screenshot for listing all the predict resources

We provide a simple Graphic User Interface (GUI) where the user can examine the resource names and methods along with the corresponding the WSDL operations. Figure 7 shows a screenshot of the prototype. The GUI provides the functionality to generate the WSDL2.0/WADL description of the corresponding RESTful service. We may need to modify the resources that do not exactly match and are tunneled through the HTTP POST operation. We highlight all the resources that are tunneled and have inappropriate naming as identified in Section 3.2. Moreover, the GUI also shows the links of other resources that the representation of a resource may contain. The service provider can modify the link to other resources.

If the service provider wants the resource to be different, the GUI can be used to edit the resource name and HTTP methods. The name in *{}* , however, has to match to one of the input parameters. Figure 8 shows the user interface where a user can edit the fields except the ID of the resource. If the service provider has a cache-related specific requirement, the requirement can be modified. We provide more readable means for describing caching terms in terms of hour, day, week, month and year along with the specific time of a day. Once all information is verified and the user confirms the resource name and HTTP method, the prototype automatically generates the necessary configuration files, servlets and wrappers. The RESTful service is ready for deployment.



http://localhost:8080/ResourceConfig/Edit?id=6

Edit Resource

ID	6
Resource Name	/customers/modified
HTTP-Method	GET
WSDL Method	GetModifiedCustomers
Content Type	application/xhtml+xml
Links To	2 5 8
Expires	hourly
Time (Format: HH:MM:SS [in GMT])	

Figure 8: Screenshot for allowing user to edit the predicted resource

5. CASE STUDY

Table 4 lists our initial survey of 713 service descriptions. It shows the existence of both types of services. Based on our observation while collecting services from the Internet, we found that RESTful services are more popular than SOAP-based services. Especially, RESTful services are popular in data-oriented services. In most of the RESTful services, the developer network is maintained in the form of blogs, social-network, and forums engaging developers around the world. The involvement and interaction with different developers is one of the reasons of growing importance of RESTful services.

Table 4: Different types of Services

Category	RESTful Services	SOAP-Services
Internet	160	55
Shopping	97	17
Search	84	11
Financial	64	57
Enterprise	74	29
Government	61	4
Total	540	173

To validate the effectiveness of our approach, we conduct a case study to 1) assess accuracy of our approach of identifying resources from existing SOAP based services; and 2) evaluate if the migrated RESTful services can fully use the benefit of the REST architecture. We compare the benefits in terms of response time of the migrated RESTful services and the response time of the SOAP-based services.

5.1 Setup for Case Study

To validate our approach for identifying the resources from SOAP-based Web services, we collect 61 WSDL documents from various categories, such as finance, government, travel/tourism, and e-commerce. Table 5 list the categories, the number of WSDL document in each category and a short description of the category. Our classification of categories is based on an online Web service listing site [31]**Error! Reference source not found.**. We identified resources and HTTP methods from each WSDL as discussed in Section 3. We analyzed the WSDL documents manually to examine the accuracy of the identified resources and the corresponding HTTP methods.

Table 5: Service used in used in case study

Category	#	Description
Finance	12	Services related to financial management and banks.
Government	6	Services provided by government organization.
Travel/Tourism	17	Services that are related to travel and tourism e.g., flight book, hotel booking, and taxi reservation.
Ecommerce	13	Services provided by online business e.g., Amazon, BestBuy and EBay.
Others	13	Services from domains such as weather, music search, content sharing and aggregation.

Table 6: Mapping between SOAP-based operations and resources

Resource	WSDL Methods	HTTP Methods
addCountry	/{country}	POST
getAllCountries	/{country}	GET
deleteCountry	/{country}	DELETE
updateCountry	/{country}	PUT
addCity	/{country}/city	GET
getCities	/{country}/city	GET
updateCity	/{country}/city	PUT
deleteCity	/{country}/city	DELETE
getCapital	/{country}/capital	GET
addCapital	/{country}/capital	POST
updateCapital	/{country}/capital	PUT
deleteCapital	/{country}/capital	DELETE

To test the performance of our REST wrappers for the SOAP-based services, we first used our approach to find the resources and HTTP methods. There are three different clusters shown in Figure 4 (b). We identified the resources and the HTTP methods for each cluster as described in Section 3. Table 6 shows the result of the identified resources, HTTP methods and the WSDL operations of a SOAP-based service. We generated the necessary wrappers and configuration files to deploy the RESTful service. Both RESTful and SOAP-based services can be invoked. We deployed both services on the same computer. The computer has Intel Core 2 Duo of 2.66 GHz, and RAM 4 GB. The server is Apache Tomcat server of version 6.0.29 and AXIS of version 1.3. We developed client applications for each service. The clients are executed in a network other than the one used to host the services. The client invokes the services at the fixed number of times.

5.2 Evaluation Criteria

We measure the effectiveness of our approach on identifying resources using precision and recall. Precision can be seen as a measure of exactness or fidelity [15]. Precision is defined in Equation (2). It measures if any irrelevant resources are misidentified as the resource. Recall is a measure of completeness [15]**Error! Reference source not found.**. Recall is defined in Equation (3). It evaluates whether our approach can correctly identify all resources without omissions. We could not compare the precision and recall with other work. To the best of our knowledge, there is not publicly available experimental data in migrating SOAP-based services to RESTful services.

$$precision = \frac{\#correctly\ identified\ resources}{\#identified\ resources} \quad (2)$$

$$recall = \frac{\#correctly\ identified\ resources}{total\ \#\ resources} \quad (3)$$

We are interested in evaluating the accuracy of our approach for identifying resources and the performance of the migrated RESTful services. Equation (4) gives the measure of our accuracy. Accuracy of our approach is given by the ratio of the difference between the number of identified resources and the number of resources modified to the total number of resources.

$$Accuracy = \frac{|\mathrm{R}_{identified} - \mathrm{R}_{modified}|}{\mathrm{R}_{total}} \quad (4)$$

where R_{total} is the total number of resources obtained manually, $R_{identified}$ is the number of resource indentified from our approach and $\mathrm{R}_{modified}$ is the number of resources modified after manually examining

For the performance, we measure the time difference between sending a request and receiving the response from the server for both types of services. Performance is the ratio of difference between response time taken by a SOAP-based service and a RESTful service by the response time taken by the SOAP-based service. Equation (5) gives the performance measure.

$$\mathrm{Performance} = \frac{\mathrm{SOAP}_{\mathrm{Average}_{\mathrm{time}}} - \mathrm{REST}_{\mathrm{Average}_{\mathrm{time}}}}{\mathrm{SOAP}_{\mathrm{Average}_{\mathrm{time}}}} \quad (5)$$

where $SOAP_{Average_{time}}$ is the average response time for SOAP operation and $REST_{Average_{time}}$ is average response time for REST resource

5.3 Analysis of the Results

Table 7 lists the result for identifying resources from SOAP-based services. The average precision is above 84%, meaning that our approach can correctly identify the services most of the times. The recall of our approach is 75%. The recall shows that our approach may fail to identify the resources correctly. We were not able to identify the resource correctly when the names of the input and output parameters are very generic. Due to unclear name in the operation, tunneling the operation name through the post operation was identified. The uses of ambiguous words and words not available in WordNet cause difficulties in identifying the resources correctly.

Table 8 summarizes the results of accuracy of the identified resources. The accuracy is 74% showing that our approach can successfully identify the resources in most of the cases. There are 410 operations in those 61 WSDL documents, for which there are 284 resources. Therefore, each resource may not always have four operations exposed for the users. We found most of Web services in our study to be read centric as most of the resources expose retrieval HTTP-verb (i.e., GET).

Table 7: Results of Identifying Resources from WSDL

Category	# Methods	#Predicted Resources	#misidentified Resources	#total Resources	Precision	Recall
Finance	59	35	8	40	0.77	0.67
Government	39	25	2	27	0.92	0.85
Travel/Tourism	132	97	16	110	0.83	0.73
Ecommerce	102	80	14	90	0.82	0.73
Others	78	47	6	53	0.87	0.77

.Table 8: Summary of WSDL to RESTful Resource identification

Number of WSDL documents Analyzed	61
Total Number of Operations	410
Number of Resources Identified	284
Misidentified Resources	46
Total Number of Resources	320
Average Accuracy	0.74

Table 9 shows the result of the service response time for both SOAP-based and RESTful services. The performance improvement is calculated using Equation (5). Even with the delay introduced by the wrappers, the performance of RESTful services is faster than of the corresponding SOAP-based services. The faster response time for the RESTful services favor mobile and thin clients. It is clear from the results that RESTful services have performance benefits compared to WSDL/SOAP based services.

5.4 Threats for Validity

In this subsection, we discuss the limitations of our approach and the different types of threats which may affect the validity of the results of our case study.

External Validity: External validity handles the issue of the generalization of the results of our study. The main threat of our experiment that could affect the generalization of the presented results relates to the number of WSDL documents analyzed. We have analyzed 61 SOAP-based services from a wide variety of industries. Nevertheless, further validation of our approach requires analyze a larger set of WSDL documents. A service provider may have multiple interdependent WSDL documents; analysis of the dependent files together may affect the resource identification results.

Internal Validity: Internal validation is concerned to the issues related to the design of our case study. There are different ways to denote a resource. We used a way to make resources more readable, following the hierarchical rule in NLP, but resources can be denoted in other ways as well. The network congestion and traffic are not taken into account for the performance evaluation of the invocation of SOAP-based and RESTful services.

6. RELATED WORK

A few approaches for migrating legacy systems to Service-Oriented Architectures (SOA) in general have been proposed in [13][11][1]. These approaches use static reverse engineering techniques for identifying potential services. RESTful services are relatively new in the area of SOA and do not have much work done in migration. There are a few approaches [2, 14, 20] to model the REST based services. Kopecky et al. [14] present hRESTS as a solution for missing machine-readable Web APIs of RESTful services. They argue that a microformat is the easiest way to enrich existing human-readable HTML documentation. They introduce a model for RESTful services, but with a focus on documentation and discovery. Alarcon and Wilde [20] introduce a metamodel for describing RESTful services as the basis for the Resource Linking Language. The authors identify links as first class citizens and focus on service documentation and composition. There are a number of work providing guidelines on designing RESTful services [16, 25, 32] which provide some basis for our migration.

Charles et al. [4] present an industrial case study to migrate a transportation Web service to a RESTful service. Charles et al. also describe the issues encountered in designing and implementing a set of RESTful services to extend and replace the traditional Web services. Markku et al. [18] describe an approach of abstracting application interfaces to REST-like services in three major steps: analyzing a legacy API, abstracting it to a canonical form with constraints in place, and generating adapter code for the abstraction.

Michael et al. [17] provide a model-driven approach in identifying REST-like resources from legacy service descriptions. Using the information contained in the descriptions of the available functionality (in the form of WSDL or message schema specifications), authors propose a way to model service operation signatures into a MOF model, called Signature Model.

Table 9: Performance of using WSDL client and RESTful version of the same WSDL service

SOAP-based Service	Identified RESTful Service		SOAP-based Service Average Response Time (ms)	RESTful Service Average Response Time (ms)	Performance Improvement in Response time
Operation Name	Resource	HTTP Method			
addCity	/{country}/city	POST	78	52	0.33
getCitiesByCountry	/{country}/cities	GET	79	49	0.37
deleteCountry	/country	DELETE	80	46	0.42
updateCountry	/country	PUT	85	65	0.23

All the approaches of identifying the resources, methods, and message conversion between the RESTful services and traditional Web services are limited to identifying the resources. Our work is a semi-automatic technique, which helps to migrate the traditional Web services to RESTful services. The presence of RESTful services does not affect the operation of traditional Web services allowing both the services to co-exist.

Several messaging optimization approaches have been introduced [8, 19, 24, 26 and 27] to address Web service performance overhead for mobile clients. Current Web Service communication models in mobile computing may result in unacceptable performance overheads. A typical Web application requires the transmission of four to five times more bytes if it is implemented as a Web service compared to the same service implemented as a traditional dynamic program using ASP or PHP [19]. In our research, we migrate SOAP-based services to RESTful services and hence reduce the XML overheads during the service communication.

7. CONCLUSION

In this paper, we propose a semi-automatic technique for migrating a SOAP-based service to RESTful services. We use cluster analysis and natural language processing to identify resources and the associated HTTP methods. The result of the case study shows that our approach can identify the resources with high precision. The case studies also show that our RESTful approach has more performance benefits compared to SOAP based services.

In future, we plan to improve the performance of our resource identification approach. We will validate our approach with a large set of WSDL documents. We will also extend our approach for migrating other legacy systems to RESTful services.

8. ACKNOWLEDGMENT

This work is financially supported by NSERC and the IBM Canada Centre for Advanced Studies (CAS) Research.

9. REFERENCES

[1] A. Almonaies, J.R. Cordy, T.R. Dean, "Legacy System Evolution towards Service-Oriented Architecture", Proc. International Workshop on SOA Migration and Evolution (SOAME 2010), Madrid, Spain, pp. 53-62.

[2] A. Alowisheq, D. E. Millard, T. Tiropanis, " EXPRESS: EXPressing REstful Semantic Services Using Domain Ontologies." International Semantic Web Conference 2009: 941-948

[3] A. K. Jain, M. N. Murty, P. J. Flynn; Data clustering: a review; ACM Computing Surveys (CSUR) Volume 31 Issue 3, Sept. 1999

[4] C. Engelke and C. Fitzgerald, "Replacing Legacy Web Services with RESTful Services," WS-REST 2010 First International Workshop on RESTful Design

[5] C. Pautasso, "On Composing RESTful Services," In: Proc. of the Dagstuhl Seminar 09021 on Software Service Engineering, July 2009.

[6] C. Pautasso, O. Zimmermann, F. Leymann, "RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision, " Proc. of the 17th International World Wide Web Conference (WWW2008), Beijing, China, April 2008.

[7] C. Pedrinaci, J. Domingue, R. Krummenacher, "On the Integration of Services with the Web of Data", Technical Report kmi-09-06, Knowledge Media Institute, http://kmi.open.ac.uk/publications/pdf/kmi-09-06.pdf

[8] D. Sosnoski, "Improve XML Transport performance Part 1 and 2," IBM developerWorks Article, June 2004. http://www-128.ibm.com/developerworks/xml/library/x-trans1.html.

[9] E. Tzoukermann, J. Klavans, C. Jaquemin, "Effective Use of Natural Language Processing Techniques for Automatic Conflation of Multi-Word Terms: The Role of Derivational Morphology, Part of Speech Tagging, and Shallow Parsing." SIGIR '97. ACM 1997. P. 148 – 155

[10] G. A. Miller, "WordNet: A Lexical Database for English. "Communications of the ACM Vol. 38, No. 11: 39-41, 1995

[11] G. Canfora, A. R. Fasolino, G. Frattolillo, and P. Tramontana, "Migrating interactive legacy systems to Web services," in Proceedings of CSMR 2006. Washington, DC, USA: IEEE Computer Society, 2006, pp. 24–36.

[12] H. M. Sneed, "Integrating legacy software into a service oriented architecture," in Proceedings of CSMR 2006. Washington, DC, USA IEEE Computer Society, 2006, pp. 3–14.

[13] H. M. Sneed and S. H. Sneed, "Creating Web services from legacy host programs," 5th International Workshop on Web Site Evolution (WSE), pp. 59–65, 2003.

[14] J. Kopeck´y, K. Gomadam, and T. Vitvar," hRESTS: An HTML Microformat for Describing RESTful Web Services". In WI-IAT '08: Proc. Int. Conf. on Web Intelligence and Intelligent Agent Technology.

[15] J. Makhoul, F. Kubala, R. Schwartz, R. Weischedel, "Performance measures for information extraction," DARPA Broadcast News Workshop, Herndon, VA, February 1999.

[16] L. Richardson and S. Ruby," RESTful Web Services," 2007. O'Reilly Media, Sebastopol, CA.

[17] M. Athanasopoulos and K. Kontogiannis, "Identification of REST-like Resources from Legacy Service Descriptions, WCRE 2010.

[18] M. Laitkorpi, J. Koskinen, T. Systa, "A UML-based Approach for Abstracting Application Interfaces to REST-like Services," 13th Working Conference on In Reverse Engineering, 2006, pp. 134-146.

[19] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, J. Schiller, "Performance Considerations for Mobile Web Services," Elsevier Computer Communications Journal, Volume 27, Issue 11, Pages 1097-1105, 2004.

[20] R. Alarcón and E. Wilde. "RESTler: crawling RESTful services." In Proceedings of the 19th international conference on World Wide Web, WWW '10, pages 1051{1052, New York, NY, USA, 2010. ACM.

[21] R.T. Fielding. "Architectural Styles and The Design of Network-based Software Architectures". PhD thesis, University of California, Irvine (2000)

[22] R.T. Fielding, “A Little REST and Relaxation.” In: Jazoon 2007, The International Conference for Java Technology.

[23] S. Berger, S. McFaddin, C. Narayanaswami, M. Raghunath, “Web services on mobile devices-implementation and experience," Proc. Of 5th IEEE Workshop on Mobile Computing Systems and Applications, 2003.

[24] S. Cheng, J. Liu, J. Kao, C. Chen, “A New Framework for Mobile Web Services," Proc. of the 2002 Symposium on Applications and the Internet.

[25] S. Vinoski, “RESTful Web Services Development Checklist,” Internet Computing, IEEE 12, 96–95, 2008.

[26] S. Oh, G. Fox, “Optimizing Web Service Messaging Performance in Mobile Computing,” Community Grids Laboratory Technical Paper, 2006.

[27] S. Oh, H. Bulut, A. Uyar, W. Wu, and G.C. Fox, “Optimized Communication using the SOAP Infoset for Mobile Multimedia Collaboration,” In Proceedings of The Fifth International Symposium on Collaborative Technologies and Systems (CTS2005), St. Louis, Missouri, USA, 2005.

[28] W. Zahreddine, Q. Mahmoud, “An agent-based approach to composite mobile Web services,” Proc. of the 19th International Conference on Advanced Information Networking and Applications (AINA’05), 2005.

[29] Mobile Internet Research Report Shows Massive Growth, http://mobilebeyond.net/mobile-internet-research-report-reveals-massive-mobile-internet-growth/#axzz1M5vzN8G7, last accessed on 10 May, 2011.

[30] O'Reilly, T. (2003). REST vs. SOAP at Amazon. http://www.oreillynet.com/pub/wlg/3005, last accessed on May 10, 2011.

[31] Programmable Web, http://programmableweb.com, last accessed on May 10, 2011.

[32] RestWiki: REST Triangle, http://rest.blueoxen.net/cgi-bin/wiki.pl?RestTriangle, last accessed on May 10, 2011.

[33] SOAP, version 1.2, World Wide Web Consortium (W3C), http://www.w3.org/TR/soap12-part0/ last accessed on May 10, 2011.

[34] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, World Wide Web Consortium (W3C), http://www.w3.org/TR/wsdl20/, last accessed on May 10, 2011.

[35] Web Application Description Language, http://www.w3.org/Submission/wadl/, last accessed on May 10, 2011.