RESTful Web Service Integration Using Android Platform

Mohammed Husain Bohara
DA-IICT
M.Tech in Computer Network
Gandhinagar, Gujarat, India
201111050@daiict.ac.in

Madhuresh Mishra
SRF at IIT Mandi
Mandi,Himachal Pradesh
lordmacrolle@gmail.com

Sanjay Chaudhary
DA-IICT, Gandhinagar, Gujarat
sanjay_chaudhary@daiict.ac.in

***Abstract*—Internet is providing the platform for accessing different kinds of services in a distributed and heterogeneous environment. Software components are available on Internet in the form of Web Services. Service Oriented Architecture (SOA) possesses various characteristics by executing Web Services using Simple Object Access Protocol (SOAP). SOAP has certain limitations such as tightly coupled invocation, performance in terms of response data, non-uniform interface and no hyperlink support. These limitations can be resolved by implementing principles of Representational State Transformation (REST). REST is an architectural style implemented by resources known as RESTful Web Services. At present, light-weight RESTful services are dominating the development of services due to simplicity of RESTful architecture. Service access and invocation is simple in RESTful framework as it uses Uniform Resource Identifier (URIs) and Hyperlinks. In this paper, we explore the idea of RESTful Service Mashup by integrating individual Web Services which can satisfy end users' requirements. For this work, we propose a recursive algorithm. The implementation work is on Android 2.2 (froyo), API level 8 and above. It is supported on most of Android based mobile devices.**

***Index Terms* – REST, SOA, Service Composition, Service Mashups**

I. INTRODUCTION

Web Service is a network-addressable and platform-independent software component of Service Oriented Architecture (SOA) to perform specific task. Web services make it possible to invoke information in a distributed environment of the Internet. SOA supports various characteristics to build broker based model known as SOA 1.0 [1][2]. The broker is a central entity of SOA. Every service provider publishes services to the broker based registry in Web Service Description Language (WSDL) format. A broker generates composition plan for user by discovering and selecting Web Services to accomplish users' prerequisites. Although SOA supports loose coupling and distributed processing but has some demerits such as message overhead and programming complexity. The emerging term Representational State Transfer (REST) was introduced by Roy Fielding in his PhD dissertation [3]. It was introduced to overcome few of these issues. The Web Services on which the REST principles are implemented are called RESTful Web Services. REST is gaining attraction because of following characteristics:

- *Light-Weight*: The HTTP request and response messages do not contain any encryption/description overhead which makes REST as light-weight.
- *Simplicity*: REST is much simpler than SOAP because it does not require any interface definition like WSDL [4].

In this paper, we propose RESTful web service integration algorithm to support aforementioned characteristics of RESTful Web Services.

A. REST Principles

Representational State Transfer (REST) was introduced by Roy Fielding in 2000. He identified set of principles to design a new REST architecture. The clients use RESTful Web Services to implement REST architecture over the Internet which follows REST principles known as REST constraints:

a) Identification of Resources: In REST, everything is defined and referred as resources. A resource is a software artifact supporting specific data, provided by the resources. Every resource is identified by unique URI. The resources are deployed over the servers and published in the form of URIs which are accessible globally. The client needs to know URIs for accessing RESTful Web Services but sometimes it is required to discover services dynamically. For example, Table I represents a list of URIs for Farmers Data.

Farmer Data	URI
Get Farmer details with name "mohan"	http://www.example.com/farmer/mohan
Post Farmer details	http://www.example.com/farmer

TABLE I. RESTFUL WEB SERVICES URI

b) Manipulation of Resources through Representations: The REST principle is implemented directly on HTTP protocol. This allows the same interface to be used to design every resource by a fixed set of operations. These operations are accessed by the same set of four methods called HTTP methods given in Table II. After the identification of resources by URI, each method performs a specific operation for HTTP request response message.

HTTP Methods	Method Description
GET	GET the resource and fetch data
PUT	Update data to resource
POST	Create new data and submit it to resource
DELETE	Delete the resource

TABLE II. HTTP METHODS FOR RESOURCE MANIPULATION

c) Self-Descriptive Messages: Resources are self descriptive in nature, it means that request message contains all necessary information to complete operation or task and response message carried into multiple representation such HTML, XML, JSON [5]. RESTful messages are constructed such that client and serve can easily understand them. It is necessary to find a resource representation for particular request and actual business operations are encoded into the request. The self descriptive nature of REST inherits from the HTTP methods which are described as read only or write only messages.

d) Hypermedia As The Engine Of Application State (HATEOAS): HATEOAS is a very important concept of RESTful architecture. Each response message should contain a hyperlink to navigate to the next request message so that neither client nor server needs to store information about the state. In this way, a RESTful applications enable the server to inform the client of the possible ways to change the state of the application via hypermedia.

B. Advantages of RESTful Web Service

The REST principles cause RESTful architecture to provide following advantages :

Easy-Accessibility: URIs used for identifying RESTful WSs can be shared and passed around to any dedicated service clients or common purpose applications for reuse. The URIs and the representation of resources are self-descriptive and thus makes RESTful WSs easily accessible. RESTful WSs have been widely used to build Web 2.0 applications and mashups.

Scalability: The scalability of RESTful Web Services comes from its ability to naturally support caching and parallelization/partitioning on URIs. The responses of GET (a side effect free operation) can be cached exactly in the same way as web pages are currently cached in the proxies, gateways and content delivery networks (CDNs). Additionally, RESTful WSs provide a very simple and effective way to support load balancing based on URI partitioning. Compared to ad-hoc partitioning of functionalities behind the SOAP interfaces, URI-based partitioning is more generic, flexible, and could be easier to realize [6].

C. Mashup

The RESTful web sevice integration is known as Mashup. It can be defined as a Web application that aggregates multiple services and its data resources into a single integrated application to achieve a new purpose. Conceptually, mashups are new web applications used for repurposing existing web resources and services.

II. Related Work

The SOA having different components where Web Services are basic entity to implement SOA characteristics by WSDL, SOAP & UDDI [7]. It helps us to understand execution flow of architecture and relation between components. The Web Service composition is a challenging issue to compose different services and give an abstract plan which aims to satisfy end users' prerequisites. The two phase algorithm gives service composition solution by implementing forward (first phase) and backward (second phase) approach. The resultant services are non redundant [8], obtained by the traditional input output parameter matching approach. There are different types of composition approaches including static (design time), dynamic (run time), Ontology [6] and semantic based automated composition [9]. At present, we are using REST based composition approach called RESTful Web Service integration. The REST principles are self explanatory and adequate to build a RESTful architecture where resources are basic entities to implement it. REST is a client server architecture [10] so there must be a standard way of communication between two different users' platform. The XML [7] make Web Services technology interoperable. WSDL describes the contract between a service provider and service consumers. The REST represents services in XML or JSON format [5] where XML is used to represent user define tags to convey information about data while JSON is the hierarchy of objects having lightweight and easy to parse values [11]. Because of lightweight property of JSON data types, REST architecture use for representation of RESTful services. The REST client server architecture communicate using HTTP protocol and the resources are manipulated by the HTTP methods [12], by performing specific actions encoded in the users' requests. Every resource is identified by unique address having all necessary information to reach particular resource, method and actions over the resource called URI [13]. The three elements such as URI, HTTP methods and Media types (XML or JSON) are uniform contract design elements for designing REST [14]. In REST architectural style, composition term is replaced by integration of Web Services. The RESTful architecture integrates multiple Web Services APIs and generate mashups [15][16].

III. Motivation Scenario

The motivation for developing RESTful services and service integration is to create an architecture model for how the Web should work, such that it can serve as the guiding framework for the Web protocol standards. There are some issues related to traditional approach of accessing Web Services by SOAP protocol such as tightly coupled, performance to return data, non uniform interface and no hyperlink, which can be resolve by RESTful Web service implementation.

IV. RESTful Web Service Integration Scenario

The term Integration is related to composition. In case of RESTful Web Services, we use integration of Web Services. We are also trying to get efficient solution through which we can built an integration system. In this thesis, we propose an algorithm that follows the Web Service integration scheme which takes input-output parameters, and gives integrated Web Services satisfying end user requirements. We can define the problem of Web Services integration as follows.

Let WS is a set of Web Services and q is a user query. W_i is a set of Web Services where i $(1 \leq i \leq n)$ and n is the number of the Web Services mashup for different user query q, W_i include Web Services invoked by input parameters of q and output parameters of Web Services stored into p. W_i is a Web Services mashup having dependent services which execute a sequence of services and fulfill end user requirements.

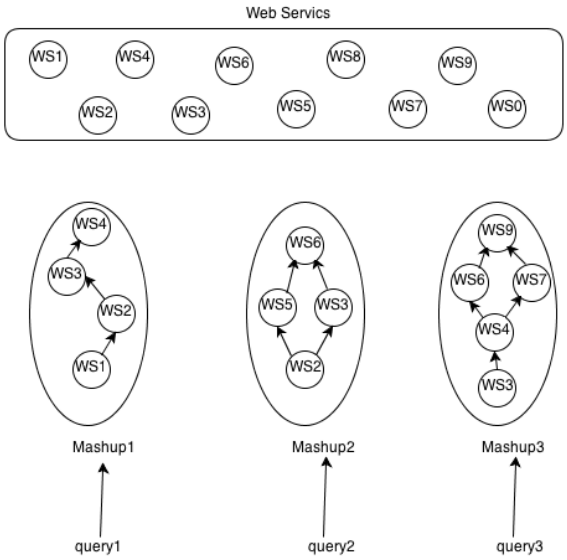

Fig. 1. RESTful Web Service Integration

The Figure 1 represents abstract view of creating RESTful Web Service Integration. The Web Service set contain different types of services associated with input-output parameters. On the basis of user query, the proposed algorithm executes and creates mashup of Web Services. There are three different queries associated with three different mashup. Once mashups are created, it can be used further for the same queries. We are also trying to create mashup from the existing mashups.

A. Steps Towards Creating Web Services Mashup

The following steps are applied over Web Service set to create mashup of dependent services as under :

1. We tried to solve above problem definition by proposed Web Service composition algorithm which can generate composition sequence.

2. At initial state, the algorithm searches for services which are available in WS set and selects a set of services which can satisfy the user request.

3. The selected services are put into the mashup W_1. We assume that services are already discovered and we perform searching over the discovered services.

4. Algorithm generates mashups having Web Services which are input-output dependent. During mashup formation process, first check input-output dependencies between new or existing Web Services according to the condition in the algorithem.

5. Services are connected by AND-parallel and XOR-parallel pattern. Formation of mashups are efficient because independent Web Services can be executed in parallel. Basically algorithm generates static composition plan on the basis of user requirements.

V. Service Composition with REST

The challenge is to find efficient service composition based on input-output parameters of Web Services. It is described as under:

A. Problem Definition

Let WS is a set of Web Services and q is a user query. W_i is a set of Web Services where i $(1 \leq i \leq n)$ and n is the number of the Web Services mashup for different user query q, W_i include Web Services invoked by input parameters of q and output parameters of Web Services stored into p. W_i is a Web Services mashup having dependent services which execute a sequence of services and fulfill end user requirements.

B. Approach Towards RESTful Web Service Integration

There can be different Web Services associated with input-output functionality. The services are invoked by users' requests and proposed algorithm find the dependent services until it can meet users' requirements. The basic description of Web Services are shown as under the Figure 2 :

- **Crop Recommendation**: It accepts the source latitude and longitude and returns a set of suggested crop types and crop varieties.
- **Data Weave**: It accepts a crop type and crop variety, returns a list of state, location, min Price and max Price for a crop.
- **Geo Coding**: It accepts a location and returns destination latitude and longitude.
- **Google Map Service**: It accepts source latitude and longitude and destination latitude and longitude and returns Google Map of that area.

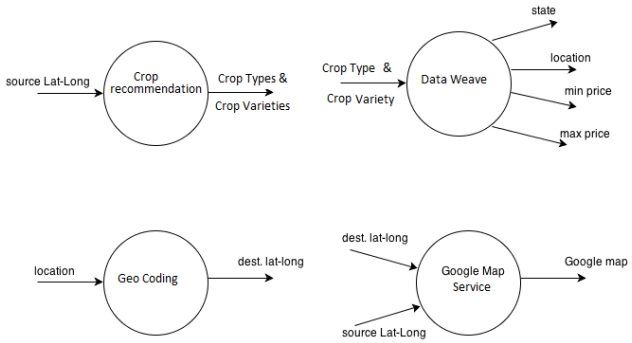

Fig. 2. Web Services with Input-Output Parameters

C. Different Parts of Algorithm

Algorithm 1 Find Service by Input Parameters

```
WebService(q_in, WS)
W_1 ← φ;
p ← φ;
for each ws in WS do
    if (q_i.in ⊆ ws_in) then
        W_1 ← W_1 ⋃ ws_i;
        WS ← WS − ws_i;
        p ← p ⋃ {ws_output};
        break;
    end if
end for
Composition(p, W_1, WS, q_out);
```

First part of algorithm explains how W_1 mashup will be filled up by Web Services. Initially W_1 is empty. Within the for loop, the input parameters are checked with each Web Service input parameters, if parameters are matched then W_1 is filled up by matching Web Services. It may be possible that the input parameters are matched partially then service will not be selected. The output parameters of selected service stored into the p. The above algorithm calls for creating each mashup according to users' queries. If we have N number of Web Services then the algorithm will be executed N times to find out services in a worst case. It takes O(N) time in a worst case.

Algorithm 2 Recursive Composition Function

```
1: Composition(p, W_1, WS, q_out)
2: Input p, WS, q_output;
3: if (q_out ⊆ p) then
4:    return;
5: else
6:    for each web service ws_i do
7:       for each web service ws_k in WS do
8:          if Composable (ws_k, ws_i) then
9:             W_1 ← W_1 ⋃ ws_k;
10:            WS ← WS − ws_k;
11:            p ← p − (ws_k.out ⋂ ws_i.input);
12:            p ← p ⋃ {ws_output};
13:            break;
14:         end if
15:      end for
16:   end for
17: end if
18: Composition(p, W_1, WS, q_out);
```

First part of algorithm calls recursive composition function by passing important parameters. The second part of algorithm handles the recursive function by finding the service dependencies with existing services in the W_1 mashup to the existing web services in set WS. If input parameters (existing web services in WS set) are matched with parameters stored into p then dependencies are found and that particular Web Service is added into the W_1 mashup. The output parameters of newly added Web Services are stored into p. Recursive composition algorithm having two for loops. In the worst case, first for loop runs up to a number of Web Services in W_1 times and second for loop runs up to (N-1) times for remaining Web Services in the WS set. It takes O(N^2) time overall for a worst case.

Algorithm 3 Check Dependencies between Two Services

```
1: Composable(ws_k, ws_i)
2: Input ws_k, ws_i;
3: if ((ws_k.out ⋂ ws_i.input) == φ)AND(ws_i.input ≠ φ)
   then
4:    return false;
5: else
6:    return true;
7: end if
```

The third part of algorithm is basically used to check dependencies between two web services. The composable function has two input parameters (Web Services). It checks for input-output parameter dependencies between a pair of services. It returns true, if both Web Services are dependent otherwise it returns false. It takes O(1) time in a worst case.

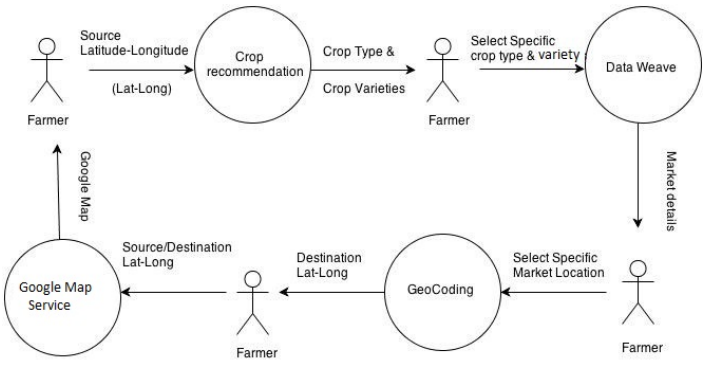

Fig. 3. Sequence of Execution of RESTful Web Services

VI. Experiments and Results

The proposed algorithm has been tested on Android 2.2 (froyo), API level 8 [17][18]. The queries of farmers have been executed by the previously created Web Services and results are shown on Android based mobile devices.

The Table III describes different RESTful Web Services and their input-output parameters. These services are helping to achieve our motivation to create a RESTful mashup. We have tried to integrate those services in a particular manner to satisfy end users' requirements. The flow of execution is shown in Figure 3.

Service Name	Input Parameter	Output Parameter
FarmerInformation	IMEI Number	Farmer Name, Survey Number, Taluka, District, State
GeoCoding	Location	Latitude-Longitude
CropRecommendation	Source Latitude-Longitude	Crop Type,Crop Variety
DataWeave	Crop Type	Location, State, Maximum Price, Minimum Price
Google Service	Latitude-Longitude	Google Map

TABLE III. RESTful Web Services

Let us assume that a farmer wants to sell his crops with best current price and nearest location of the market. What could be suitable crop cultivation practices based on Agro-ecological zone and environmental parameters like temperature, humidity etc?

The farmer can use application on a mobile device from his/her farm. International Mobile Equipment Identity (IMEI) number is extracted from the mobile device and it is appended into the URI to refer Geographical Information System (GIS) server. It invokes a RESTful service which gives farmer's personal information as the output. The latitude-longitude values are extracted using Global Positioning System (GPS) or the mobile network carrier which can then be used to show the farmer's current location.

The selected Web Service and service output parameters are passed into recursive algorithm to find dependent Web

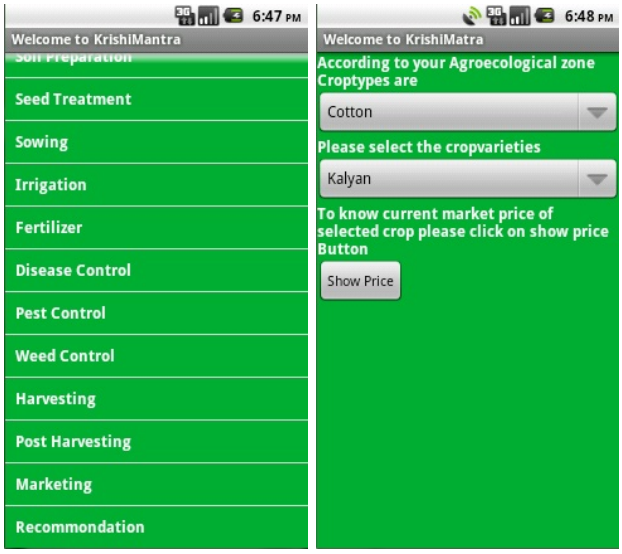

Fig. 4. Main Menu and Crop Type

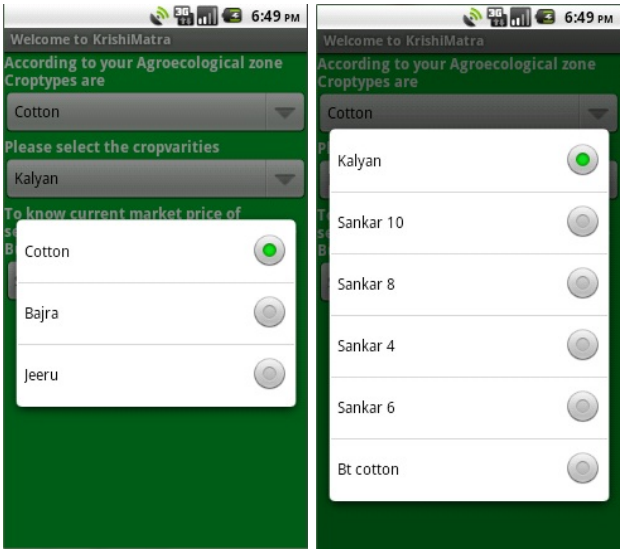

Fig. 5. Crop Type and Crop Variety List

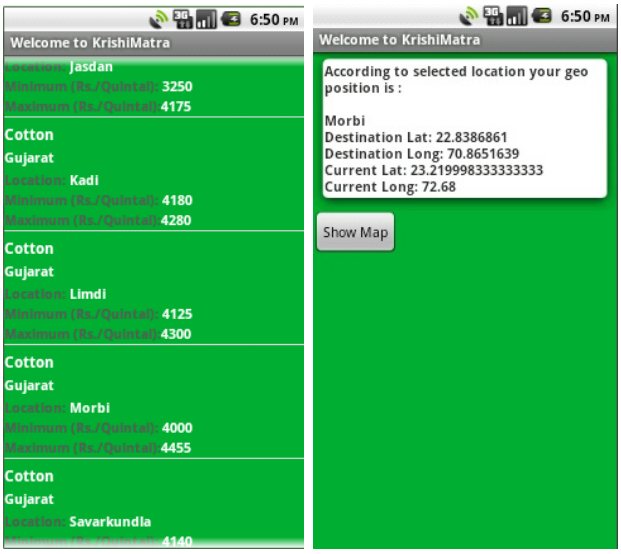

Fig. 6. Current Market Price and Latitude-Longitude

Fig. 7. Map and Satellite View

Services. The recursive algorithm calls for each selected Web service until it meets end user's requirement.

When a farmer takes an action on crop recommendation query then his/her latitude-longitude is appended into URI of RESTful GIS web service. This service retrieves farmer's Geo position from URI and finds out to which Agro-ecological zone a farmer belongs. The GIS service is integrated with temperature API which provides information about crops. On the basis of temperature and Agro-ecological zone of their region, crops are suggested dynamically as shown in Figure 4.

The RESTful GIS Web Service output is generated in the form of a JSON object. At the farmer's end, JSON parser parses the object and displays the crop type and crop variety as an output on Android device. By selecting a particular crop type and crop variety as shown in Figure 5.

A RESTful data weave Web Service is executed which gives current market price (Min, Max) of that crop in different markets of a region. The RESTful Web Service output in the form of a JSON object is received at farmer's end. By parsing the JSON output, the results are displayed on a farmer device in a list manner. The farmer selects a particular nearest market location in the list to show the route from his/her current location. Using RESTful Geocoding Web Service of Google which takes market location as an input and gives latitude-longitude of that market location as an output as shown in Figure 6.

The output of Geocoding Web Service is used as an input of Google Map's Java-Script API v3, which shows driving directions to the selected market location from the farmer's current location on a separate widget as shown in Figure 7.

VII. Conclusion and Future Work

The Web Service composition can be achieved by having information about Web Service input-output parameters,

service starting point and service ending point. The WSDL and ontology are two different formats for storing information about Web Services. Instead of previous approaches which depended on Web Service descriptions format, this approach is based on RESTful principles. The algorithm is implemented over the RESTful contract elements such as HTTP methods, URIs and media types to provide users' interaction without storing service descriptions. The users can interact at every step and perform the appropriate service selection to fulfill their requirements.

The proposed algorithm is able to create mashup of RESTful Web Services by input-output parameter matching. This algorithm is capable of processing complex queries using integration of different RESTful Web Services. At each step of execution, resultant output parameters are appended into URIs and dependent Web Services are invoked. The RESTful Web Service mashup follows the static service integration. The selection of services are static but the data retrieval is dynamic in nature.

We have tried to give a generalized solution of RESTful Web Service integration. This algorithm generates Mashup, related to farmers' queries. The results of queries are tested on Android 2.2 (froyo), API level 8 and above, which is supported by most of the Android mobile devices.

The above work can be extended by using some of the following ideas :

- The dynamic RESTful Web Service integration can be accomplished by selecting Web Services from mashups. It will allow integrating Web Services from different mashups dynamically to satisfy end users' requirements.
- QoS parameters can be included in the integration of RESTful Web Services to improve service selection quality.
- GeoService REST API can be applied to the proposed algorithm. The GeoServices REST API series of standards provide a standard way for web clients to communicate with Geographic Information System (GIS) servers, based on Representational State Transfer (REST) principles.
- This algorithm can also be used along with ArcGIS Server REST API which provides a simple and open web interface to services hosted by a server. All resources are exposed by REST API through URL and each GIS service is published with the server.

ACKNOWLEDGMENT

This work is part of a research project on 'Service-Oriented Architecture for Spatial Data Integration and Spatial Reasoning', which is funded by The Natural Resources Data Management Systems (NRDMS) and National Spatial Data Infrastructure (NSDI), under the scheme: Geo-spatial technology based tools development for NSDI, Department of Science and Technology, Govt. of India for the year 2011-14. We acknowledge DST for their funding and support.

REFERENCES

[1] B. Wu, L.-F. Xi, and B.-H. Zhou, "Service-oriented communication architecture for automated manufacturing system integration," *International Journal of Computer Integrated Manufacturing*, vol. 21, no. 5, pp. 599–615, 2008.

[2] V. A. de Souza and E. Cardozo, "A service oriented architecture for deploying and managing network services," in *Service-Oriented Computing-ICSOC 2005*. Springer, 2005, pp. 465–477.

[3] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, 2000.

[4] C. Pautasso, "Soap vs. rest bringing the web back into web services," *Presentation, http://www. iks. inf. ethz. ch/education/ss07/ws soa/slides/SOAPvsREST ETH. pdf, Accessed*, vol. 17, 2010.

[5] D. Crockford, "Json: The fat-free alternative to xml," in *Proc. of XML*, vol. 2006, 2006.

[6] H. Zhao and P. Doshi, "Towards automated restful web service composition," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 2009, pp. 189–196.

[7] T. Erl, *Service-oriented architecture: a field guide to integrating XML and web services*. Prentice Hall PTR, 2004.

[8] J. Kwon and D. Lee, "Non-redundant web services composition based on a two-phase algorithm," *Data & Knowledge Engineering*, vol. 71, no. 1, pp. 69–91, 2012.

[9] S. Dustdar and W. Schreiner, "A survey on web services composition," *International Journal of Web and Grid Services*, vol. 1, no. 1, pp. 1–30, 2005.

[10] S. Allamaraju, *RESTful Web Services Cookbook*. Yahoo Press, 2010.

[11] K. Zyp *et al.*, "A json media type for describing the structure and meaning of json documents," 2010.

[12] R. Fielding, Roy Thomas, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Rfc 2616: Hypertext transfer protocol–http/1.1, 1999," *URL http://www. rfc. net/rfc2616. html*, 2009.

[13] R. T. Fielding, "Relative uniform resource locators," June 1995.

[14] C. P. R. B. H. u. W. D. B. Erl Thomas, Benjamin Carlyle, *SOA with REST: Principles, Patterns and Constraints for Building Enterprise Solutions with REST*, E. Thomas, Ed. The Prentice Hall Service technology Series from Thomas Erl, 2012.

[15] M. Treiber, K. Kritikos, D. Schall, S. Dustdar, and D. Plexousakis, "Modeling context-aware and socially-enriched mashups," in *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups*. ACM, 2010, p. 2.

[16] C. Pautasso, "Restful web service composition with bpel for rest," *Data & Knowledge Engineering*, vol. 68, no. 9, pp. 851–866, 2009.

[17] J. Steele and N. To, *The Android developer's cookbook: building applications with the Android SDK*. Addison-Wesley Professional, 2010.

[18] I. Darwin, *Android Cookbook*. O'Reilly Media, Incorporated, 2012.