REST4Mobile: A Framework for Enhanced Usability of REST Services on Smartphones

ABSTRACT

Considering end-user research and proliferation of smartphones and REST interfaces, we envisage smartphone owners can innovate to compose applications on the small screen. This paper presents the design and evaluation of a REST service development framework (namely REST4Mobile) with the aim to enhance the usability when consuming on smartphones. Our design process employs the usability factors identified in our previous work as primary constraints for modeling the framework and a corresponding composition tool. Thus, sample REST services are developed with and then without the framework; and usability of composing the services on smartphones is evaluated. Evaluation was conducted by deploying the component REST services, the composition tool, and the resulting composite apps on a local machine. As the task of service composition is conducted directly on the smartphone's screen, the evaluation process is designed to be repeatable on remote servers and on the cloud. Results showed that constraints can be added into the REST architectural style based on the influences of domain specific terms and human cognitive capabilities on the naming and size of the URIs. In addition, the principles embodying the framework are found to be influential factors in enhancing the usability of REST services on smartphones.

Keywords*:* Usability, REST Service, Smartphone, Service Composition, Semi-automatic, Framework, Cloud Computing, End-user development

1. INTRODUCTION

Study in (Lieberman *et al*., 2006; Wajid *et al.*, 2010) confirmed that end-users having little or no programming background can innovate and develop new applications. Thus, together with the proliferation of smartphones and REST interfaces, we envisage that end-users can compose REST services on the small screen real state of the smartphones. For example, a supplier can use his phone to compose an application to find customers having best price offer for products to sale, negotiate a contract, arrange billing details, and logistics for shipment; each of which exposing their services via REST interfaces. However, the REST services must be designed taking this new demand into account.

On the other hand, as pointed out in (Mesfin *et al.*, March 2017), the design of REST services for the demand mentioned above must comply with the constraints of the REST architectural style. Furthermore, Li (2011) describes the design goals of the REST in terms of scalability of component interactions, generality of interfaces, independent deployment of components, and provision of intermediary components.

According to Pautasso (2014), significant number of software development frameworks have adopted REST service constraints following the conception of the REST architectural style (Fielding, 2000). These frameworks attempt to implement REST service design principles such as prerequisite gathering, resource identification, definition of resource representation, and URI design. Accordingly, the resulting REST services are exposed through the web (or provided as cloud services) and composed by REST client developers, as shown in figure 1. Figure 1

demonstrates an interaction scenario between the REST service providers and consumers as follows:

- The service providers expose their services as per the principles of the REST architectural style. The service exposition can be on conventional web servers in companies' own data centers; or they can be hosted on a cloud as described in (Christensen, 2009).

- The REST service consumers, in turn, discover the services and orchestrate them to develop apps tailored to specific business needs. The resulting composite application is then hosted for future use either on the consumers' devices, remote servers, or on the cloud. Christensen (2009) emphasizes on the cloud hosting describing its importance for mobility by exploiting the REST's linking role.

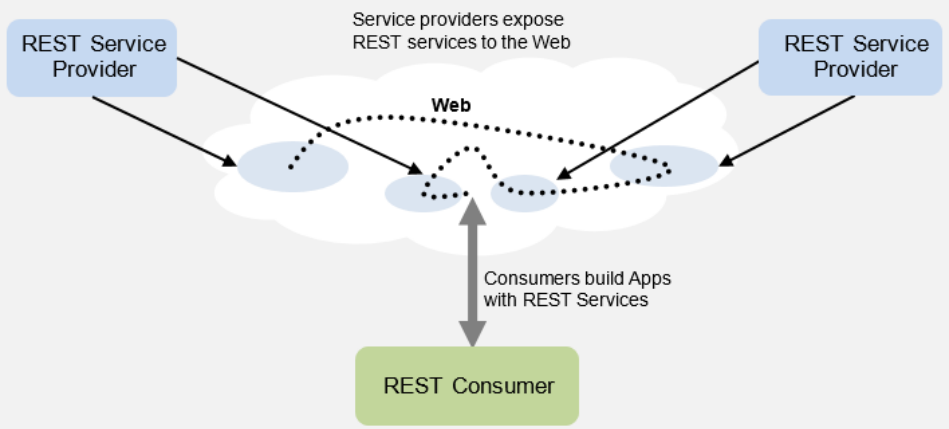

Figure 1. A Scenario for REST Service Composition

As elaborated in (Li, 2011), the frameworks attempt to automate the tedious tasks in the process of REST service development and composition. In addition, they introduce architectural solutions to recurring problems and workflows; and features of the existing REST service frameworks and their shortcomings with respect to usability on the smartphone environment (see section 2).

Overall, the existing frameworks strive to implement the REST principles to different levels of rigor such as deploying on mobile phones, and improving the user experience to integrate REST services into web servers or application servers. However, they lack to consider the need for exposing REST services for enhanced usability (compose-ability) on the small screen real state of smartphones.

In this paper, we employed the design science research methodology described in (von Alan *et al.*, 2004) and the factors identified in our work in (Mesfin *et al.*, March 2017) to describe new framework called REST4Mobile. In addition, the framework's usability evaluation is conducted using systems usability scale (SUS) (Sauro, 2014). Thus, the paper is organized as follows. Background information related to the REST service development and composition frameworks is provided in 2; a description of our framework, prototype design, and evaluation are provided in sections 3, 4, and 5 respectively; finally, section 6 provides the conclusion.

2. Related Work

Many of the REST service frameworks described in (Li, 2011; Pautasso, 2014) strive to implement the REST constraints in a similar approach but with varying programming languages. Here, we briefly describe a few Java-based REST service frameworks – Restlet, Jersey, and RESTEasy.

Restlet[1] is an object-oriented, open source web framework in Java that provides a set of classes and capabilities to leverage programs with the constraints of the REST architectural style (Louvel *et al.*, 2013; Li, 2011). It focuses on web applications (i.e., web services, websites, and web clients) and is capable of exposing and consuming web resources using HTTP features like conditional methods, and content negotiation. Restlet is composed of a core component and extensions where the core is a standalone JAR, which contains APIs and an engine. It supports standards like JSON, RDF, and Atom; connectors like POP3, SMTP, and FTP; and can be integrated with other frameworks such as Jackson, JAXB, Spring, and Velocity (Louvel *et al.*, 2013).

Jersey[2] is another open source REST service development framework in Java in which the Java code can be published as REST service using source code annotations. The Jersey framework provides support to and serves as a Java API for REST services (JAX-RS) reference implementation, an API standard for REST service development (Pautasso, 2014). It works together with a Java library called Jackson[3], a high-performance JSON processor used to convert Java object to JSON and vice versa.

RESTEasy[4] is a REST service framework in Java that implements JAX-RS and is based on source code annotations. It specifies the REST services development based on metadata grammar of the JDK (Li, 2011). RESTEasy can run in any Servlet container, and integrates tightly with an application server for improved user experience. It can also integrate with EJB and the Spring framework.

In general, existing REST service development frameworks strive to implement the REST principles to different levels of rigor (Louvel *et al.*, 2013; Li, 2011). Some frameworks (e.g. Restlet) have also made efforts to deploy REST services (and clients) on mobile platforms. In fact, the cloud has become the de facto deployment platform for mobile apps (Grønli *et al.,* 2012; 2013) which is even more preferred when the REST interfaces are used for the interconnection (Christensen, 2009). While other frameworks like RESTEasy have attempted to improve the user experience to integrate REST services into web servers or application servers (Li, 2011; Louvel *et al.*, 2013). However, current REST services development and composition practices lack the capability to expose REST services which are usable (compose-able) on the smartphone platforms.

Our work in (Mesfin *et al.*, March 2017) indicated that the depth and breadth of URIs namespace; number of request parameters and response messages, and the naming of URIs, parameters and response messages are found to be the factors influencing the usability of consuming REST services on a smartphone. Accordingly, we use these factors to describe a framework that enhance usability of REST services.

---

[1] http://www.restlet.org/
[2] http://jersey.java.net/
[3] http://jackson.codehaus.org/
[4] http://www.jboss.org/resteasy.html

This paper is driven by the findings of our previous study in (Mesfin *et al.*, March 2017), wherein we identified the breadth and depth of URIs namespace, number of request parameters and response fields, and the naming of URIs, parameters and response fields as factors influencing the usability of consuming REST services on smartphones. We employed the design science research as our study methodology (von Alan *et al.*, 2004). Accordingly, a description of our approach for designing a framework is provided next.

Prior to describing the approach, let's briefly explain what a framework is. Jabareen (2009) describes a framework as a network of interlinked concepts. These concepts are a set of distinct, heterogeneous and inseparable components that together provide a comprehensive understanding of a phenomenon in varying disciplines. Hence, in this study, we concentrate on a software framework.

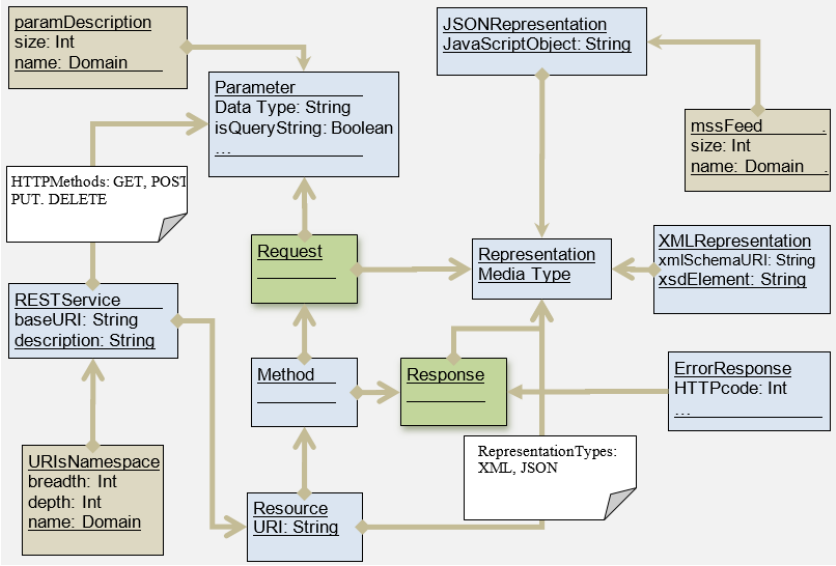

Figure 2. REST's Extended Conceptual Model in UML Class Diagram

A software framework is an abstraction of a universal and reusable software functionality (a system of components and their interactions) which facilitates the development of application-specific software through additional user-written code. Software framework is different from normal libraries in that it is extensible into specific functionality, its code is not modifiable, and its flow of control can only be dictated by the framework instead of the caller (Riehle, 2000). Accordingly, our framework is designed taking these framework characteristics into account.

Our software framework is elaborated based on an abstract of the concepts which describe a REST service as a meta-model and provides a consistent conceptual model with the model-driven principles (Hang and Zhao, 2013; Valverde and Pastor, 2009). Thus, we use the factors discussed in (Mesfin *et al.*, March 2017) and provide an extended conceptual model of REST services for a smartphone as shown in the class diagram in figure 2. Here, the resources labeled *URIsNamespace*, *paramDescription* and *mssFeed* (mobile simple syndicate) are the new objects encapsulating the principles constituting our framework and they are added into the conceptual model. These resources are described by the size and naming of the URIs namespace, parameters and response fields.

3. A FRAMEWORK OF REST SERVICE DEVELOPMENT FOR MOBILE USERS - REST4MOBILE

In this section, we present a set of principles and best practices that describe a framework of REST service development for mobile users; hereafter referred to as REST4Mobile. The REST4Mobile framework takes into account the business-to-business (B2B) integration needs (Hogg *et al.*, 2004; Martens, 2003), and realizes it using REST services and end-user service composition on smartphone platforms.

The B2B integration requirement establishes a distributed business process which can be realized by the composition of a set of REST services as depicted in figure 3 (Martens, 2003). Figure 3 (a) shows a graphical box of the REST service's conceptual model representing a local sub process. According to Marino *et al.* (2013), the Inlet and Outlet points in the box are connected by links that represent buses for data exchange with some transformations (T) between REST services during service composition (see figure 3 (b)).

Figure 3. A REST Service's Model (a) and Composition (b) Scenario (Marino *et al.*, 2013)

3.1. REST4Mobile Design Principles

The REST4Mobile framework is composed of different components. The factors used to describe these components constitute a guideline for designing REST services for enhanced usability. Thus, the set of design principles constituting the guideline are presented next.

Our first principle is associated to the fact that the size and naming of a URI's namespace, request parameters and response messages are used as design constraints to expose REST service APIs representing the categories of web resources in organizations. That is, the design of REST service APIs must be goal-driven and distributed to developers to implement organizational concerns and enhance their learnability, discoverability, composability, and ease of remembering. Accordingly, the size and naming of URIs, request parameters, and response messages shall be designed in a similar way:

Represent REST service APIs of similar organizational concern in similar ways (1)

In the REST4Mobile framework, the use of domain-specific terms to name URIs' nodes in path annotations of resources, request parameters and the fields of response messages enable composers to easily guess identifiers and messages. Thus, organizational terms shall be used to write the code and documentation of REST service APIs and enhance learnability, discoverability, composability, and ease of remembering. These are embodied in the second principle:

Take up the terms and map strategic goals of the problem domain to describe the REST service APIs associated to each organizational concern (2)

The third principle is related to a desirable design constraint called clean URIs which may result in the effort to limit the query string. In this regard, each clean URI represents REST services having smaller granularity. Thus, complex REST services may be split into simpler ones with better usability, as stated below.

Split bigger web resources to a level of REST service APIs that can be identified using clean URIs and with the aim to solve smaller organizational concerns (3).

In the REST4Mobile framework, the number of fields and records of a response message are used as a design constraint. This constraint enhances usability not only for REST service composers, but also for end-users of the composite application. For example, limiting the number of fields and querying to retrieve only relevant records reduces the need for end-user's horizontal and vertical scrolling on the smartphones' small screen, respectively. In general, bringing too much data onto the smartphone is not desirable and is stated as follows.

Avoid too much data that users do not really need. That is, database queries made by a REST service should bring only the most relevant columns and records (4).

The fifth principle is associated with the composition tool that orchestrates REST services on smartphones. Although MSS feeds and MVC respectively reduce coding for data transformation and provide a flexible user interface for a diversity of composers, a resulting generic tool is still a usability challenge for composers. Thus, designing composition tools for specific patterns of problems would further enhance usability and is stated as:

A generic composition tool would further limit usability of REST services on smartphones. Thus, its design must focus on solving certain patterns of problems (5).

In general, the service composability principle requires that a web service should be flexible in order to facilitate the different data exchange requirements for similar concerns (Erl, 2008). However, this requirement contradicts with the simplicity attribute of usability (Bore and Bore, 2005), and many of the above stated principles. Thus, such a situation is a principle on its own and is stated as:

Simplicity and flexibility are design trade-offs in the design of a REST service for enhanced usability on smartphones (6).

However, the above-mentioned principles, on which the REST4Mobile framework is based, would require further description into *Principle profile* as discussed in (Erl, 2008). In addition, further study is required to show if the mapping of Service Oriented Architecture (SOA) principles into strategic goals works for the mapping of Resource Oriented Architecture (ROA) principles too, and if the strategic goals can be mapped to usability principles (Erl, 2008).

3.2. Organizational Setting of URIs, Parameters and Response Messages

Currently, the most common use of REST services is to build data mash-ups (Lathem *et al.*, 2007). That is, the use of REST services to develop applications under the notion of ROA

(equivalent to SOA) is at a premature level. In addition, to the best of our knowledge, the use of ROA on a smartphone is almost inexistent.

In REST4Mobile, however, we take the ROA approach into account to integrate REST services on a smartphone thereby bring about B2B interoperability. In addition, we consider the organizational knowledge where the REST service is exposed from or composed in (Arpinar *et al.*, 2005).

Accordingly, we develop a resource-oriented development model depicted (see figure 4) by adopting the component-based development model described in (Pressman, 2005). In figure 4, we depicted a mapping of the domain knowledge of an organization and the REST service lifecycle (Getahun *et al.*, 2007), where domain ontology is used to semantically describe the REST services' APIs for B2B integration on the smartphone.

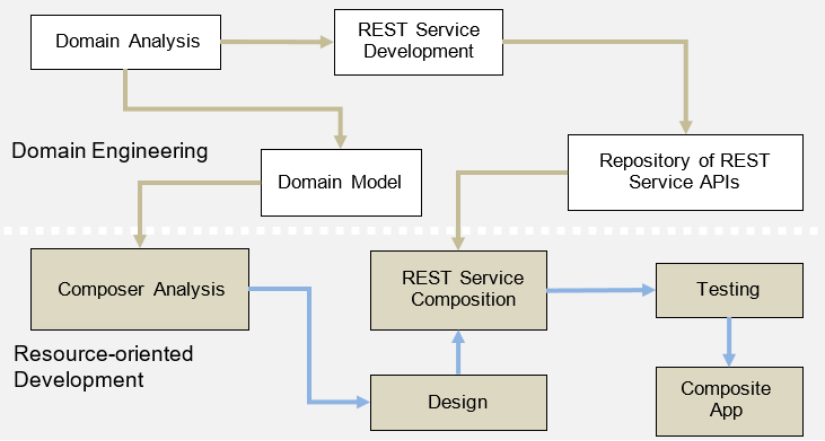

Figure 4. Resource-Oriented Development Process Model (Adapted from Pressman (2005))

Beaton *et al.* (2008) describe that the fundamental phases in the lifecycle of a web service are service discovery, composition, and invocation. Thus, our focus here is on the discovery and composition operations.

Organizations may expose and or compose REST services as per their strategic goals and the requirements of their business processes respectively (Dino *et al.*, 2012). Here it requires making the REST services discoverable and then consumable by business processes, as depicted in figure 5.

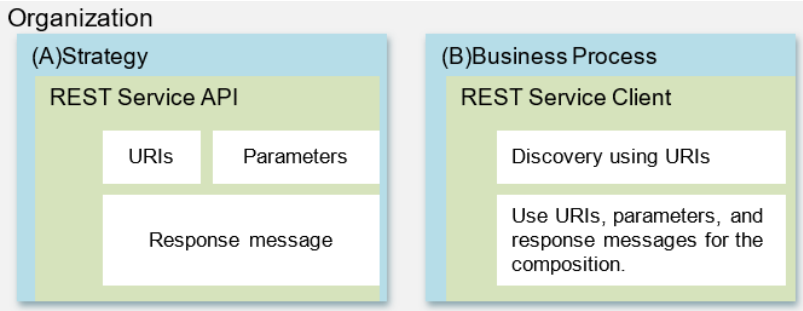

Figure 5. Organizational Setting of URIs, Parameters and Response Messages

Figure 5 shows the exposition of a REST service's API through publishing its URI and parameters based on the strategic goals (A) and a REST service client (B) implementing a business process discovering the API using the URI and performing service composition via the URI, parameters and response message.

Lanthaler and Gütl (2010) stated that developers spend most of their time in searching, analyzing, reformatting and combining information. Accordingly, the issues in the discovery and composition of REST services and the influence of organizational knowledge are described below.

Discovery - service providers commonly expose their online services on their homepages to push data to composers in the form of RSS and ATOM. However, the REST services URIs and the data expected as input and as output are generally manually described for human readability only (Lanthaler and Gütl, 2010).

Composition - once suitable services are found, they are usually combined into composite applications that cross organizational boundaries. This operation requires mediation, i.e. a layer to translate the data formats between different services (Lanthaler and Gütl, 2010). In addition, entering the base URIs and parameters in the composition is generally manual and the small screen size and constrained input mechanisms of the smartphone aggravates the challenge.

In general, REST service discovery and composition are manual operations. Thus, describing URIs and parameters using domain-specific terms (as in figure 6) and limiting their corresponding sizes facilitates the manual search, particularly on smartphones.

Moreover, Fielding (2000) did not mention the size and naming of URIs and parameters as constraints in the REST architectural style. To address this issue, among others, we provide an extension of the existing REST frameworks through the REST4Mobile framework (figure 6). In this framework, the naming, and size of URIs and parameters are integrated based on domain-specific terms, and users' cognitive capability, respectively. A detailed design description of the URIs namespace and parameters in the REST4Mobile framework are provided next.

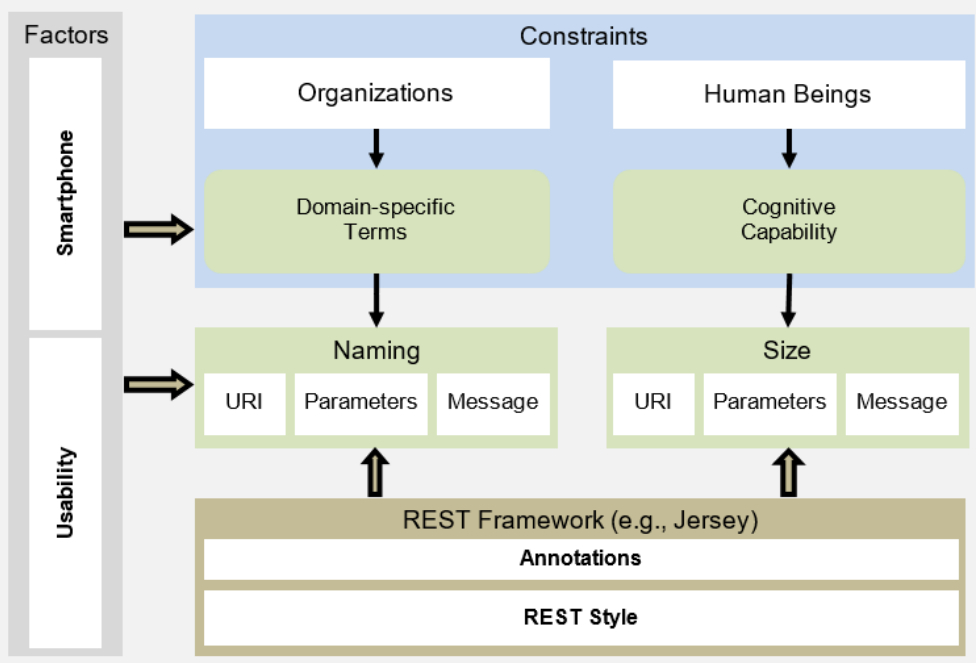

Figure 6. The REST4Mobile Framework

3.3. URIs Namespace

A URI is used to identify a web resource. The URI namespace, just like the DNS namespace, is hierarchically organized as a rooted tree emanating from a domain name (Tanenbaum and Van Steen, 2007); we adapt the labeling of nodes in the DNS namespace for describing the labeling of nodes of the URIs namespace in the REST4Mobile framework. Thus, a label in the URIs namespace is described as a case-insensitive string made up of alphanumeric characters; with a maximum length of 63 characters; and the length of a complete URI path name is restricted to 255 characters.

However, a REST service's API (hence the URI's namespace) is required to be designed and exposed as editable as possible so that human users can easily understand and retrieve the desired content (Lanthaler and Gütl, 2010). This issue is further described in (Rodriguez, 2008) stating that the REST services' URIs should be intuitive to the point where they are easy for the developer to understand and guess through self-descriptive design of the interface, and should not be merely a collection of slash-delimited strings. With this usability requirement, the above mentioned restriction in number of characters is never enough. Accordingly, we employ the usability constraints in (Mesfin *et al.*, March 2017) for the REST4Mobile framework and present the description of the breadth and depth, the naming of the URI namespace, and response messages as follows.

3.3.1. Breadth and Depth of URIs Namespace

Recall that an organization's domain-name is unique and its sub units are also given unique Internet addresses using the caret symbol (~) as a sub-domain name. Thus, an organization can have several URIs namespaces for addressing resources hosted on its domain name or subdomains and each namespace is defined by a unique root node.

Literature in (Miller, 1956; Bouwers *et al.*, 2011) shows that human beings can recognize only 7±2 things can be recognized; our results in (Mesfin *et al.*, March 2017) align to this precept, highlighting the need to limit the breadth to a range between five and nine. Thus, we deduce that the optimum breadth of a URIs namespace is 7±2 leaves.

In figure 7, we show that a tree representing the maximum breadth of a URIs namespace with direct links from the root to the leaves.

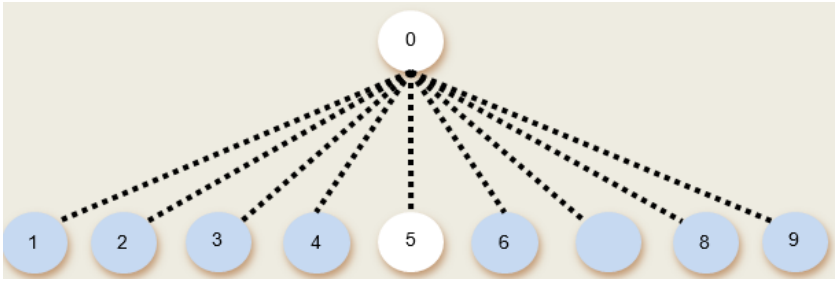

Figure 7. Breadth of a URIs namespace

In practice, an organization will have many categories of web resources that need to be exposed with REST service APIs. This implies that, the categories of resources need to be identified with multiple URIs namespaces each having its own root.

For example, an organization is commonly structured into units like accounting, and human resources each representing the categories of web resources, hence the roots of the corresponding URIs namespace (Osterwalder, 2004). In addition, each category can be further sub-categorized; for instance *Accounting* can be split into *Assets*, and *Procurement* which is used to label the nodes in between the root node and the leaves.

Thus, by applying binary partitioning on the tree in figure 7, we can see that the maximum depth of an URI's namespace is three, as depicted in figure 8. This confirms to the need to limit the depth of a URIs namespace to less than five (Deitel and Deitel, 2011).

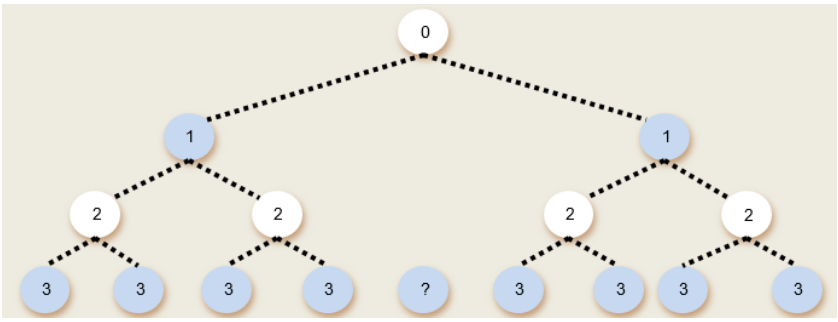

Figure 8. Depth of a URIs namespace

In the case of figure 8, the root node labeled "0" signifies the organization's domain name such as "http://example.org"; nodes labeled "1" and "2" for the possible category and sub category of resources such as *Accounting* and *Procurement*; and the leaf nodes labeled "3" for the actual resources such as *Bid*. This would form a URL string like "http://example.org/accounting/procurement/bid" which is not too complex through which to discover a REST service and compose it on the limited screen size of the smartphone.

In general, we can see that the overall size of an URI's namespace influences the usability of REST services on smartphones. Accordingly, the different categories of web resources of an organization need to be designed and exposed as REST service APIs in such a way that the breadth and depth of the URIs namespace is restricted.

3.3.2. Naming the Nodes of URIs Namespace

Web resources are not statically stored in a server's folder; instead they are dynamically generated. That is, the corresponding URI's nodes do not necessarily represent the folder structure on which the resource is located. As a result, URI's nodes may not be named appropriately which makes the discovery and composition of REST services challenging.

As illustrated in (Guinard *et al.*, 2011), the connectedness constraint of the REST style is supposed to enable REST services to follow hyperlinks they find in resource representations in order to discover and interact with other services. Here clients are assumed to use standardized identifiers and well-defined media types in the discovery process thereby explore services without the need for dedicated discovery formats. However, although a few technologies exist such as hREST and SA-REST which would be used to index and semantically annotate respectively with a goal to enhance discoverability, the common practice is to go to websites like ProgrammableWeb[5] where the REST services are published by category (Kopecky *et al.*, 2008; Lanthaler and Gütl, 2010; Lathem *et al.*, 2007).

However, there exists no technology for standardized discovery or for the composition of REST services on smartphones (Valverde and Pastor, 2009). Hence, the naming (labeling) of nodes of the URIs namespace using domain-specific terms emanating from the organizations' strategic goals and objectives, and/or the causal relationships of activities in business processes would fill the mentioned technological gap by enhancing usability of the discovery and composition operations of REST services.

The naming of nodes of a URIs namespace requires an ontological approach where agreements must be reached using content-specific terms, concepts, components and relationships for communication among REST service providers and composers on a particular organizational domain (Osterwalder, 2004). The strategic goals and business processes of an organization are usually documented as strategic plan and workflow manuals (or organizational structure).

As described in (Pahl and Zhu, 2006), we describe a distributed model that specifies the ontological elements, namely, concepts and roles. In our case, concepts are categories of web resources (as can be described in the strategic plan and business process documents) having the same properties and the roles are their interrelationship.

Accordingly, we formulate a Petri net model of an ontology-based naming of nodes in the URIs namespace which enables to capture properties of REST services and support composition description and reasoning (see figure 9). The model takes an organization's strategic plan and business process documents as inputs and delivers domain-specific terms as outputs.

[5] http://www.programableweb.com/

As used in (Martens, 2003), we describe the model using Petri net N = (P, T, F) where "T" stands for a set of transitions represented with boxes; "P" a set of places in ellipses; and "F" flow relations in arcs. In this regard, a transition is an activity; while a place (*Pi, where 0 ≤ i ≤ 7*) is the states between activities such as communication channels or resources.

The distributed URIs namespace model takes the size constraint of the URIs namespace (described in the previous section) into account and provides the ontological terms constituting categories of resources and causal relationships of activities of business processes (Champin, 2013). In addition, the use of short and familiar names can significantly contribute more to the enhancement of usability.

In general, the domain-specific terms resulting from the model (a component of the REST4Mobile framework) are used to formulate a URI string for annotating the path while designing the REST service, and also enable end-users such as domain experts to easily discover (and or guess) resource identifiers and use in the composition.

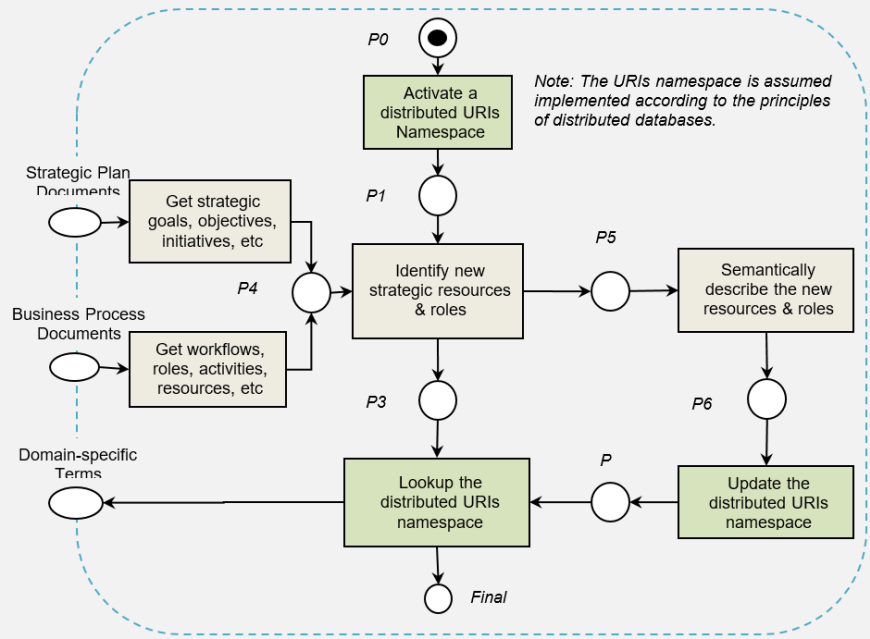

Figure 9. A Distributed and Ontology-based Model for the Naming of Nodes in URIs Namespace

3.4. Request Parameters

A client has to send a self-contained request message to the REST service so as to perform stateless communication (Fielding, 2000). Thus, in addition to the URI, a request may include additional parameters to be sent encoded in the URI's query string to access and or modify a resource.

Request messages having complex data structures may also be explicitly described using representations. However, overly describing the messages would result in a request having too many parameters. For example, a request can either be without any parameter and as clean as "/books/author-name/book-title" or a query string like in "/books/search.jsp?author=author-name&title=book-title" where it ranges from a few parameters to complex data structures.

We can describe these two cases as follows. The first case is a desirable property representing clean URIs (i.e., without any query string) while the second one has a query string having at least one request parameter (e.g. the above mentioned URI has two parameters).

Our survey result in (Mesfin *et al.*, March 2017) showed that limiting the number of request parameters to less than five and naming them with domain-specific terms enhances the usability of REST services on smartphones. Accordingly, our first attempt for the REST4Mobile is to avoid any request parameter by converting the query string into clean URIs, each representing REST services having smaller granularity than the original.

However, when some request parameters must persist, they have to be limited to less than five and named using short, self-descriptive, and familiar domain-specific terms using the distributed namespace described above (see figure 9). Thus, instead of using direct URIs such as in "/book-{call-number}" or "/books/{call-number}", hierarchical data must be described either in clean URIs or explicitly parametrized. For example, a book may be exposed depending on the intended granularity of the REST service as "/books/100102" or "/books/search.jsp? call-number='100102'".

3.5. Response Messages

A response message is a resource representation containing data and the metadata describing the data (Fielding, 2000). Information about the resource corresponding to the representation may also be included in the response message as resource metadata.

The metadata is a name-value pair, where the structure and semantics of the data (or *Value*) in the response message is defined by the *Name*. It can also be used to describe the response message with hyperlinks to related resource representations which enables the composer to discover related resources in a service providing organization using the base URI only. Here, we can see that such reduction of number of URIs entries contributes to the enhancement of usability of REST services on smartphones.

However, when carefully designed, the metadata can play more roles in the composition of REST services. For example, appropriate design of the metadata can reduce the effort required to develop an adapter that transforms response messages and close the compatibility gap during integration (see figure 3). Hence, limiting the number of name-value fields and using domain-specific terms for the naming can help achieve a higher usability goal, particularly on the smartphone platform.

The study in (Mesfin *et al.,* 2017) recognized that the JSON messaging format is one of the elements in our stack of appropriate technologies for the smartphone. Thus, in the REST4Mobile framework, we make use of JSON to design a schema with less than five fields of objects and self-described using domain-specific terms shown in figure 9 above.

Our JSON schema (figure 10) addresses the issue identified in (Mesfin *et al.,* March 2017) that realizing the *Self-descriptive messages* constraint of the REST style enhances the usability of REST services on smartphones. Accordingly, the schema is implemented by adding the *AS* keyword in the *SELECT* SQL construct to describe the response message using short, self-

descriptive, and familiar domain-specific terms as names for the metadata without affecting any legacy database schema. In figure 10, we illustrate a schema called mobile simple syndicate (MSS) which is an RSS or Atom like format used to describe a response message in JSON. In the MSS format, the first metadata is called *Object* referring to object of application domains like accounting, and human resources (Zhao and Doshi, 2009).

```
{
  "object": "object-name",
  "max-size":4,
  "fields": {

      "field1": {
        "type": "type1"
      },
      "field2": {
        "type": "type2"
      },
      "field3": {
        "type": "type3",
      },
      "field4": {
        "type": "type4",
      }
  },
  "required": [fields, …],
  "detail": "domain-name/object"
}
```

Figure 10. Generalized Schema of an MSS Feed

The *Max-size*, *Fields*, *Required* and *Details* fields represent the maximum number of fields of the object in the message, its properties, required fields, and hyperlink respectively; and object-specific schemas are defined under *Fields*. Thus, the MSS can be regarded as a model of domain-specific syndicate targeting the implementation of ROA on the smartphone platform.

In general, limiting the number and naming of the name-value fields of a response message enhances the usability of REST services for mobile service composers. In addition, limiting the number of fields would also limit the number columns displayed on the smartphone's screen, thereby minimizing the need for an end-user's horizontal scrolling. Similarly, the queries contributing to response messages of the REST service must also be carefully designed to retrieve only relevant records and thus minimize the end-user's vertical scrolling on the smartphone's screen.

3.6. A Composition Tool for Smartphones

A composition tool represents the IDE used to realize a distributed business process for B2B integration. It enables users to compose a set of web services that implement the interaction with partners (Martens, 2003). Our study in (Mesfin *et al.,* March 2017) indicated that the composition tool in use influences the usability of REST services on smartphones. Accordingly, we describe a REST services composition IDE for smartphone, where a composition results in a web application that orchestrates the services. The composition tool integrates components from the REST4Mobile framework such as the distributed URIs namespace as shown in figure 11. In addition, figure 11 depicts a toolbox for recently used URIs, a search facility for REST interfaces, and a composition canvas as part of the composition tool.

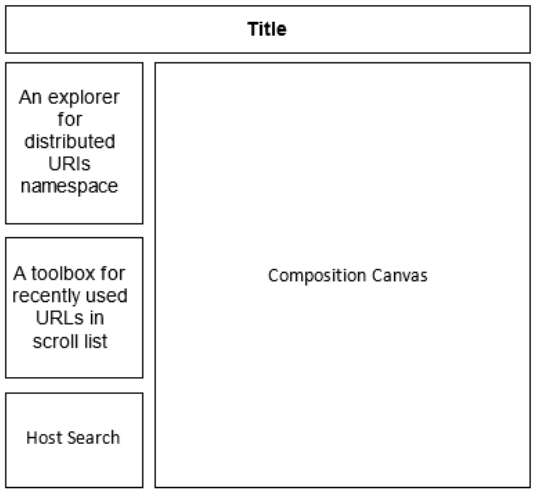

Figure 11. Features of the REST Service Composition IDE

A REST service consumer may expand and pick items like the URIs or recently used URLs from the boxes on the left panel and drop them on the composition canvas on the right. In addition, a copy of the distributed URIs namespace can be downloaded into the local storage of the smartphone and use code accelerators on the canvas to enter the domain-specific terms constituting the URLs. This represents an introduction of semantic design as per the experts' opinions presented in (Mesfin *et al.,* March 2017). In any case, the limited size, and short and familiar naming of nodes of URIs namespace and request parameters described above, improve the usability of REST service composition on the smartphone.

The composition canvas is a component that represents the service composition algorithm on which the workflow is implemented. Here, we specify REST service APIs as the data source and data sink resources, and the MSS feed described above eases the ability to change data format of the sources in the integration of services.

The model-view-controller (MVC) is a design pattern that enables separation of presentation logic from other data logic and provides flexible user interfaces to end-users (Sommerville, 2009). The experts' opinions presented confirmed this notion that the MVC pattern helps develop a REST service composition tool. In addition, composition tools may be designed for specific patterns of problems like a retrieve-change-update, and criteria-select-update scenarios.

4. FRAMEWORK IMPLEMENTATION

In this section, we present the implementation perspective of components of the REST4Mobile framework, namely, URIs namespace and MSS feed, and the composition tool. Our main goal is to automate certain components and provide a usability evaluation setting for the REST services designed based on the framework.

4.1. Component Design

Two sets of sample REST services have been developed with and without the guidelines of the REST4Mobile framework aiming at implementing path annotations at the class and method levels, altogether constituting the URIs of the sample REST services that are used in the evaluation. The

complete URL of the REST services is therefore a concatenation of the hosts' domain names, and the URIs of the specific resources. In addition, design descriptions of the other features of the framework such as the naming component and the schema for an MSS feed of a specific object are briefly explained in the subsequent sections.

As described previously, the naming component of the REST4Mobile framework is an important part of the proposed automated "@Path" annotation wizard. The naming component can be implemented based on domain-specific terms of the business in question. Thus, the principles embodied in the resource description framework (RDF) is semantic technology from the semantic web framework for Java (Jena). These are general purpose technologies which can be applied for any application domain. In this research, we concentrate on the relationships between terms found in strategic plan and organizational structure documents relating to the demand-supply of products and services for use by a smartphone user.

RDF enables to model domain-specific terms associated to the demand-supply domain as web resources in a similar way to conceptual modeling in the classical entity–relationship diagrams. RDF makes statements about web resources in triples, namely, the subject–predicate–object expressions denoting the resource, its traits relating to the object, and object itself, respectively (Champin, 2013). For example, we can apply the namespace constraints in the REST4Mobile framework and provide hypothetical demand-supply terms for Addis Ababa University (AAU) as shown in table 1. The hypothetical RDF triples in table 1 can also be described using RDF graph as depicted in figure 12.

Table 1. Hypothetical RDF Triples for AAU

Subject	Subject's URI	Predicate	Object	URI
AAU	aau.edu.et	supplies	service	aau.edu.et/supply
supply	aau.edu.et/supply	has a	product	aau.edu.et/supply/product
supply		has a	service	aau.edu.et/supply/service
product	aau.edu.et/supply/product	has a	item	aau.edu.et/supply/product/code
service	aau.edu.et/supply/service	has a	item	aau.edu.et/supply/service/code
AAU		demands	service	aau.edu.et/demand
demand	aau.edu.et/demand	has a	product	aau.edu.et/demand/product
demand		has a	service	aau.edu.et/demand/service
product	aau.edu.et/demand/product	has a	item	aau.edu.et/demand/product/code
service	aau.edu.et/demand/service	has a	item	aau.edu.et/demand/service/code

The classical RDF/XML can be implemented on the REST service developer tool. However, large footprint size could be a bottleneck for the smartphone's composition tool as more RDF data is entered. Thus, the compact representation of RDF is more suitable.

Once the domain terms are encoded in RDF, it can be uploaded into a shared repository as distributed data for use by other programmers in different organizations to add their content. It can also be downloaded and is easily parsed with i.e. JavaScript by REST service consumers (including smartphone users) during service composition.

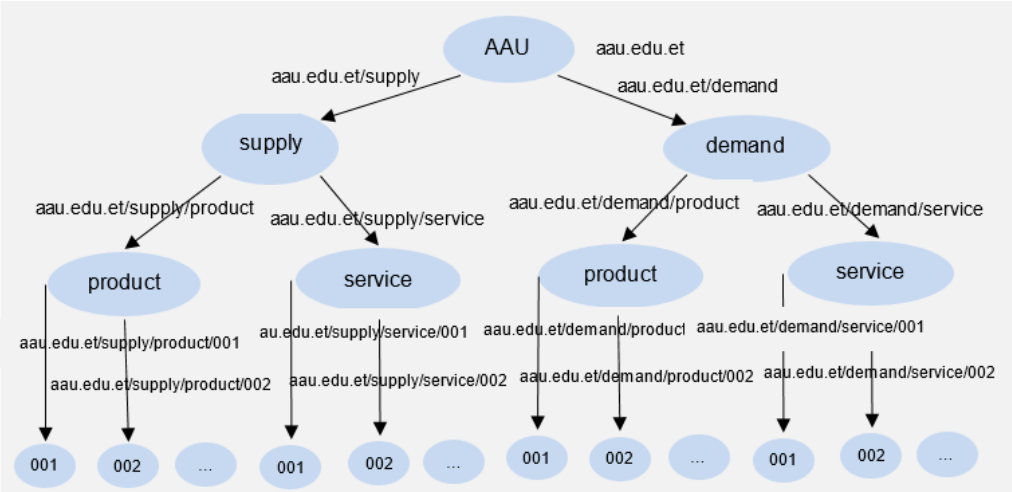

Figure 12. Hypothetical RDF Graph for AAU

4.1.2. Messaging

The MSS feed messaging is an important component of the REST4Mobile framework as a schema for describing data structures. Thus, MSS feed is defined by a JSON file containing object name, number of fields, list of object fields (like name, quantity, price, and remark). As described in figure 13, the organizations who want to supply or procure products or services, first expose their data to their customers using the schema in MSS feed. Next, the smartphone user provides the REST service's URIs of the data source (and update site) for use in the remaining composition process to generate a REST based (reusable) application. Finally, when the REST based application is in use, it invokes the REST service and uses the MSS reader to validate the data. Thus, the data is used for further transactions as per the application's specification.

```
{
  "object": "product",
  "max-size":4,
  "fields": {
        "name": {
          "type": "string"
        },
        "quantity": {
          "type": "integer"
        },
        "price": {
          "type": "number",
        },
        "remark": {
          "type": "string",
        }
  },
  "required": ["id"],
  "detail": "example.com/product"
}
```

Figure 13. MSS Feed Schema for Product

4.2. REST Client Development

In the classic scenario, a REST client can be designed for either a specific purpose application or as a complex composition tool such as the mash-up editors. In addition, the prospect towards

end-user development of REST client applications on smartphones has been discussed in (Mesfin *et al.*, 2016). Thus, for the smartphone, we describe a design of simple tool that involves end-users in the composition of REST services (see figure 14).

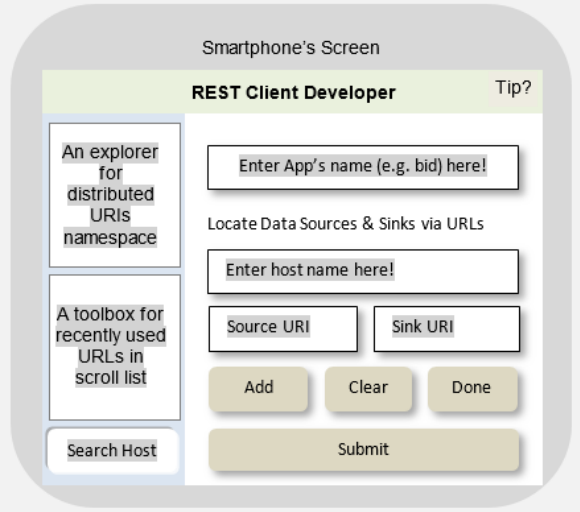

Figure 14. User Interface of a Composition Tool for Retrieve-Change-Update Type of Problems

Figure 14 describes a user interface of our composition tool designed for problems having a retrieve-change-update pattern. For example, a customer can retrieve *Product* data from different suppliers and send *Order* to the preferred ones by filling the corresponding quantity. Similarly, a supplier can retrieve *Product* data of bids floated by different customers and sends *Quotation* to the preferred ones by filling the corresponding prices.

As per our recommendation in (Mesfin *et al.*, 2017), HTML5 and JQuery (containing JavaScript APIs) are used to develop the tool. Hence, the *getJSON* function in JQuery is used to invoke the REST services' APIs.

As a cross-platform tool, the source code is debugged on the Android SDK using fairly large number of JavaScript lines. Thus, we adapted Android manifest and configuration files for HTTP access, and Cordova PhoneGap plug-ins, in a similar approach to our work in (Mesfin *et al*., 2016).

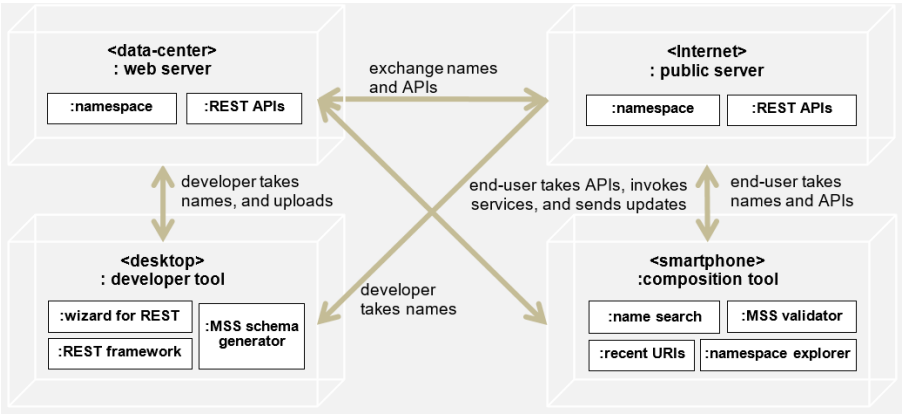

Figure 15. UML Deployment Diagram for the REST4Mobile Framework

In general, this setting represents a specific implementation scenario of the generalized REST service composition IDE described in previous sections. That is, the design canvas depicted in figure 14 is tailored to solve the pattern of problems described above.

In summary, the naming and path annotation (a guideline in a wizard), and messaging components of the REST4Mobile framework are particularly pluggable into the REST service developer tool while the REST client development environment describes the end-user composition tool for the smartphone user. The naming, however, is a component on the web that serves both. Figure 15 depicts a high-level design of these components and their interactions in UML deployment diagram.

In figure 15, we consider a developer machine, a client machine, a web server on local data center, and public servers on the Internet as of the deployment infrastructure for certain components of the framework. However, as mentioned in (Gertner, 2013; Lemos *et al*., 2016), the shift into more specific types of compositions for certain patterns of problems and the increasing significance of composition of cloud services can a lend itself to cloud adoption for the REST4Mobile framework. Thus, the framework's basic and composite services; programming and execution environment; and computing, storage, and network requirements can be mapped into the software as a services (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) components of the cloud, respectively.

5. Framework Evaluation

In this section, we provide an evaluation of the REST4Mobile framework. The goal of our evaluation process is to validate the claimed usability enhancement of the REST services designed per the framework's guideline.

As a case study, the URIs are made clean as described in the previous sections. Our evaluation also focused on the guidelines for the size and naming of URIs and the composition tool as features of the REST4Mobile framework. The sample REST services are exposed from an apache-tomcat server on localhost, and the composition tool, and the resulting composite apps are all deployed on a local machine. Thus, the following two sets of REST services are implemented representing the design of URIs with and without the REST4Mobile framework.

For each REST service API in one set there is another on the second set performing the same function but different in its URI design. In the first set, the REST services are assigned conventional URIs where each node in the path is named with non-domain specific terms including numbers, hyphens and underscores. The number of nodes in each URI was also set to more than three.

The URIs of the second set are assigned accordingly to the guidelines in the REST4Mobile framework. In each set, three REST services are implemented - to retrieve products' list and details from bid announcements, and for updating them with customers' identity for their consent of participation in the bid on specific products (see table 2). Moreover, the composition tool described above was used to evaluate these sample REST services and this evaluation is provided next.

Table 2. URIs of Sample REST Services

Function	Method	REST4Mobile URIs	Conventional URIs
List products	GET	/demand/product	/services/procurement/announcement-number-12/bid/category-7
Show detail	GET	/demand/product/001	/services/ procurement/ announcement-number-12/bid/category-7/001
Send customers' consent	POST	/demand/product	/services/procurement /announcement-number-12/bid/category-7

5.1. Evaluation Method

Usability evaluation can be conducted using methods such as models, end-user questionnaires, and experts' opinions. For example, Martens (2003) describes a usability evaluation approach that models web services as workflows in Petri net to show the compatibility and equivalence of constituent modules. Technology independent approaches like questionnaires in the systems usability scale (SUS) (Sauro, 2014), and interviews for experts' opinions are also in use for evaluating usability of graphical user interfaces.

In the REST4Mobile framework, the compatibility of services is automated as an MSS schema validator as part of the composition tool which was demonstrated for its usability in our work in (Mesfin *et al.*, 2016). Here, we focus only on one platform as a control for comparing usability of two sets of REST services. Thus, usability of entering base URIs into the composition tool using graphical user interface is more important, which can be easily explored using experts' opinions.

In the experts' opinion, an evaluator uses a product and assesses its usability against a set of principles or guidelines. That is, experts' opinion is not only convenient and cheaper but technical contributions can also be directly obtained from the participants. Thus, technical issues exposed by the experts are captured using our approaches in (Mesfin *et al.*, 2016) but tailored to experts' evaluation. To this end, a five-scale SUS checklist was provided.

Sixteen practitioners who have at least a bachelor degree and previous experience in web programming /mash-up tools were selected to evaluate the two sets of REST services. We also confirmed that the practitioners were willing to act as neutral assessors to provide valuable interpretation based on their experience. Accordingly, their responses were analyzed and the evaluation results is described next.

5.2 Evaluation Result

Implementation of the component that would tell clues of terms for the URIs nodes in the namespace (as described previously) is not in our scope here. However, the sample URIs and hostnames in table 2 were provided to the participants to type them into the composition tool in figure 14. In addition, a small "Tip" stated below was provided:

Please enter the domain names, sources and destination URIs of your client(s). If you want to add more clients, please push the "Add" button; push the "Done" button to complete, and then items along with check boxes will be listed in the drop down menu. Finally, tick on the items you want to bid for and push the "Submit" button.

The terms used to signify the URIs' nodes in the first set are described by their clear business terms, small number of characters and absence of whitespaces. In addition, all participants described the path length of the URIs of the first set shorter, altogether making them easier to understand, to use, and to remember. In addition, only two participants encountered errors during typing, as opposed to five encounters of errors in the second set.

As described above, our interview was accompanied by a checklist for participants for usability rating and these responses were then analyzed. Accordingly, participants' responses had high values for odd numbered questions and low for the even ones for the first set of REST service (REST4Mobile) and reversed for the second set. The response data was also further analyzed using the perceived usability and percentile ranking. Thus, the score and percentile rank of the first set of REST services was above 68 and grade *C* respectively, implying that their perceived usability is generally above average.

Single factor analyses of variance (ANOVA) of the mean score of the two sets were conducted for statistical significance (see table 3). In this setting, the composition tool was an independent variable while the SUS items as dependent. Thus, at 5% significance level, there is statistically significant difference in mean SUS scores. Table 3 show the mean responses of respondents to each question for the first set of REST services is above two on the converted[6] four-scale. This reconfirms that developing using the REST4Mobile framework can leverage an increased level of usability. In general, our results show that developing REST services according to the guidelines in the framework enhances their usability on smartphones.

Table 3. Interview Checklist and Results

No	SUS Question	REST4Mobile		Conventional	
		Mean	SD	Mean	SD
1	I think that I would like to use this system frequently.	4.3750	0.7440	2.7500	0.4629
2	I found the system unnecessarily complex.	2.1250	0.6409	4.3750	0.5175
3	I thought the system was easy to use.	4.5000	0.7559	1.7500	0.8864
4	I think that I would need the support of a technical person to be able to use this system.	1.6250	0.7440	4.0000	0.7559
5	I found the various functions in this system were well integrated.	4.3750	0.7440	1.8750	0.8345
6	I thought there was too much inconsistency in this system.	2.0000	0.7559	4.0000	0.5345
	I would imagine that most people would learn to use this system very quickly.	4.3750	0.5175	1.7500	0.7071
8	I found the system very cumbersome to use.	1.6250	0.7440	4.1250	0.6409
9	I felt very confident using the system.	4.5000	0.5345	1.8750	0.6409
10	I needed to learn a lot of things before I could get going with this system.	1.8750	0.6409	4.1250	0.8345

In summary, the evaluation results showed that the guidelines in the REST4Mobile framework can enhance usability of REST services on smartphones which encourages further studies on design approaches for enterprise level mobile end-users.

[6] SUS Scores coded as [response – 1] for odd questions, and [5 – response] for the even ones.

6. CONCLUSION

In this paper, we provided a description of our framework that enhances usability of REST services on smartphones, namely, REST4Mobile. In addition, implementation perspectives and evaluation of the framework was presented. The framework's design description is driven by six principles which led into REST service development guidelines. Accordingly, the design of a composition tool for certain patterns of problems, and a deployment architecture is provided. For the evaluation, sample REST services are developed with and without the framework's guidelines. The evaluation was conducted by deploying the component REST services, the composition tool, and the resulting composite apps on a local machine. As the task of service composition is conducted directly on smartphones' screens, the evaluation process is repeatable on remote servers and on the cloud. That is, to host the two sets of sample REST services and the composition tool on the cloud thereby accessing them via a mobile browser to conduct usability evaluation of performing service composition on the constrained screen real state. Accordingly, our framework evaluation showed a possibility for extending the REST architectural style for usability on smartphones based on the influences of domain specific terms and human cognitive capabilities on the naming and size of the URIs of REST services. In addition, the principles embodying the framework are found influential as factors to the enhancement of usability of REST services. Our results are based on the usability evaluation of the REST service which developed using merely unautomated guidelines of the framework. Thus, research can be pursued in future to further enhance the framework's usability by incorporating an automated distributed namespace and a wizard for the framework's guidelines, and the evaluation can be repeated on remote servers and on the cloud.

7. REFERENCES

Arpinar, I. B., Zhang, R., Aleman-Meza, B., & Maduko, A. (2005). Ontology-driven web services composition platform. Information Systems and E-Business Management, 3(2), 175-199.

Beaton, J. K., Myers, B. A., Stylos, J., Jeong, S. Y. S., & Xie, Y. C. (2008). Usability evaluation for enterprise SOA APIs. In Proceedings of the 2nd international workshop on Systems development in SOA environments (pp. 29-34). ACM.

Bouwers, E., Correia, J. P., van Deursen, A., & Visser, J. (2011). Quantifying the analyzability of software architectures. In Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on (pp. 83-92). IEEE.

Champin, P. A. (2013). RDF-REST: a unifying framework for web APIs and linked data. In Services and Applications over Linked APIs and Data (SALAD), workshop at ESWC (pp. 10-19).

Christensen, J. H. (2009, October). Using RESTful web-services and cloud computing to create next generation mobile applications. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications* (pp. 627-634). ACM.

Deitel, P., & Deitel, H. (2011). Java How to program. Prentice Hall Press.

Dino, N., Dico, A., Midekso, D., & Alemu, W. (2012). Case Study of Service Oriented Enterprise Architecture Framework. Computer Technology and Application, 3(3).

Erl, T. (2008). Soa: principles of service design (Vol. 1). Upper Saddle River: Prentice Hall.

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine).

Gartner. (2013). Top 10 Strategic Technology Trends for 2014. Retrieved from http://www.gartner.com/
newsroom/id/2603623.

Getahun, F., Tekli, J., Atnafu, S., & Chbeir, R. (2007). Towards Efficient Horizontal Multimedia Database Fragmentation using Semantic-based Predicates Implication. In SBBD (Vol. 2007, pp. 68-82).

Grønli, T. M., Hansen, J., & Ghinea, G. (2012). Mobility, Context and Cloud--Exploring Integration Issues in a Meeting Room Scenario. International Journal of Interactive Mobile Technologies, 6(2).

Grønli, T. M., Hansen, J., Ghinea, G., & Younas, M. (2013, March). Context-aware and cloud based adaptation of the user experience. In Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on (pp. 885-891). IEEE.

Guinard, D., Mueller, M., & Trifa, V. (2011). Restifying real-world systems: A practical case study in rfid. In REST: From Research to Practice (pp. 359-379). Springer New York.

Hang, F., & Zhao, L. (2013, June). HyperMash: A Heterogeneous Service Composition Approach for Better Support of the End Users. In Web Services (ICWS), 2013 IEEE 20th International Conference on (pp. 435-442). IEEE.

Hogg, K., Chilcott, P., Nolan, M., & Srinivasan, B. (2004). An evaluation of Web services in the design of a B2B application. In Proceedings of the 27th Australasian conference on Computer science-Volume 26 (pp. 331-340). Australian Computer Society, Inc.

Jabareen, Y. R. (2009). Building a conceptual framework: philosophy, definitions, and procedure. International Journal of Qualitative Methods, 8(4), 49-62.

Kopecky, J., Gomadam, K., & Vitvar, T. (2008, December). hrests: An html microformat for describing restful web services. In Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on (Vol. 1, pp. 619-625). IEEE.

Lanthaler, M., & Gütl, C. (2010, April). Towards a RESTful service ecosystem. In Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on (pp. 209-214). IEEE.

Lathem, J., Gomadam, K., & Sheth, A. P. (2007, September). Sa-rest and (s) mashups: Adding semantics to restful services. In Semantic Computing, 2007. ICSC 2007. International Conference on (pp. 469-476). IEEE.

Lemos, A. L., Daniel, F., & Benatallah, B. (2016). Web service composition: a survey of techniques and tools. ACM Computing Surveys (CSUR), 48(3), 33.

Li, H. (2011, September). RESTful Web service frameworks in Java. In Signal Processing, Communications and Computing (ICSPCC), 2011 IEEE International Conference on (pp. 1-4). IEEE.

Lieberman, H., Paternò, F., Klann, M., & Wulf, V. (2006). End-user development: An emerging paradigm (pp. 1-8). Springer Netherlands.

Louvel, J., Templier, T., & Boileau, T. (2013). RESTlet in action. Shelter Island, NY: Manning.

Marino, E., Spini, F., Paoluzzi, A., Minuti, F., Rosina, M., & Bottaro, A. (2013, May). HTML5 visual composition of REST-like web services. In Software Engineering and Service Science (ICSESS), 2013 4th IEEE International Conference on (pp. 49-55). IEEE.

Martens, A. (2003, December). Usability of web services. In Web Information Systems Engineering Workshops, 2003. Proceedings. Fourth International Conference on (pp. 182-190). IEEE.

Mesfin, G., Grønli, T. M., Midekso, D., & Ghinea, G. (2016). Towards end-user development of REST client applications on smartphones. Computer Standards & Interfaces, 44, 205-219.

Mesfin, G., Ghinea, G., Midekso, D., & Grønli, T. M. (2017). Web service composition on smartphones: The challenges, and a survey of solutions. *Systems and Software (On progress)*. Mesfin, G., Grønli, T. M., Ghinea, G., & Younas, M. (2017, March). Usability of Composing REST Services on Smartphones. The 31-st IEEE International Conference on Advanced Information Networking and Applications (AINA-2017) *(Accepted)*.

Miller, G. A. (1956). The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychological review, 63(2), 81.

Osterwalder, A. (2004). The business model ontology: A proposition in a design science approach.

Pahl, C., & Zhu, Y. (2006). A semantical framework for the orchestration and choreography of web services. Electronic Notes in Theoretical Computer Science, 151(2), 3-18.

Pautasso, C. (2014). RESTful web services: principles, patterns, emerging technologies. In Web Services Foundations (pp. 31-51). Springer New York.

Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave Macmillan.

Riehle, D. (2000). Framework design (Doctoral dissertation, Diss. Technische Wissenschaften ETH Zürich, Nr. 13509, 2000).

Rodriguez, A. (2008). Restful web services: The basics. IBM developerWorks.

Sauro, J. (2014). Quantitative Usability, Statistics, and Six Sigma: Measuring Usability. Retrieved from http://www.measuringusability.com/ sus.php.

Sommerville, I. (2009). SOFTWARE ENGINEERING. Addison-Wesley.

Tanenbaum, A. S., & Van Steen, M. (2007). Distributed systems. Prentice-Hall.

Valverde, F., & Pastor, O. (2009). Dealing with REST Services in Model-driven Web Engineering Methods. V Jornadas Científico-Técnicas en Servicios Web y SOA, JSWEB.

von Alan, R. H., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS quarterly, 28(1), 75-105.

Wajid, U., Namoune, A., & Mehandjiev, N. (2010, May). A comparison of three service composition approaches for end users. In AVI (p. 407).

Zhao, H., & Doshi, P. (2009). Towards automated RESTful web service composition. In proceedings of IEEE International Conference on Web Services (pp. 189-196).