An Efficient Web Service Registry Framework for Mobile Environments

Rohit Verma, *Student Member, IEEE,* and Abhishek Srivastava

Abstract—The advancements in technology have transformed mobile devices from a mere communication widget to a versatile computing device. Proliferation of these hand held devices have made them a common means to access and process digital information. The integration of service oriented architecture (SOA) with these omnipresent mobile devices has the potential to further empower mobile device owners to become information providers. However, one of the challenges in this integration is the lack of an efficient service registry that caters to the issues associated with dynamic and volatile mobile environments. Existing service registry technologies designed for traditional systems fall short to accommodate the issues and complexities of mobile environments. In this paper, we propose a novel approach to manage service registries 'solely' over mobile devices, thus realizing SOA in mobile environments without requiring high-end computers and high management cost. The approach manages a service registry in the form of a light weight roster and provides efficient operations to manage the registry. We assess the feasibility of our approach by engineering and deploying a working prototype on real mobile devices. A comparative study of the proposed framework and the traditional UDDI (Universal Description, Discovery, and Integration) registry is also included. The evaluation of our framework has shown propitious results in terms of battery cost, scalability, hindrance with native applications.

Index Terms—Mobile Web Services, Web Service Registry, Web Service Discovery, Service oriented architecture.

1 Introduction and Motivation

Continued evolution in technology made computing devices an integral part of one's life. The most common manifestation of this is the 'Mobile Phone'. Modern technology has transformed the mobile phone from a mere communication device to a versatile computing device. These hand-held devices have enabled us to not just access information, but also to provide information to others on the move. Modern mobile phones, equipped with powerful sensors, have endowed capabilities to provide and create near real-time information. This real-time information is useful for oneself and for others. An established approach for sharing and provision of information in a distributed environment is Service Oriented Architecture (SOA) [1] [2]. Integrating SOA and mobile devices has the potential to convert mobile phones owned by common people from *information subscriber* to *information providers*.

One of the advantages of this integration is that, it can be used in a scenario where there is little or no preexisting infrastructure. Examples of such scenarios are include War-front, Post-disaster relief management and several others. In such scenarios, mobile based SOA would enable ground teams to provide runtime information to commanding units, help teams at the disaster site to exchange data, analyze damage study and examine other related statistics using mobile devices. Furthermore, a coalition of SOA and mobile devices would enable service providers, service consumers and service registries to provide all related information and operations using hand held widgets.

Web services are a proven way to implement "Service Oriented Architecture". The use of this technology is so popular that accessing web services from mobile devices is very common today. For instance, mobile devices are commonly used for ticket booking, shopping over e-commerce sites, accessing map services. Further, advancement in mobile device technology has motivated researchers to explore the coalition of web services and mobile devices, thereby realizing service oriented systems in mobile environments. Consequently, there have been substantial works towards enabling mobile devices to host web services [3] [4] [5] [6] [7]. However, an important aspect of service oriented system, "service discovery" remains a challenge in mobile environments. Indeed, service discovery for distributed environments exist in literature [8] [9], but one catering totally to mobile environments is still lacking. Several challenges specific to hosting web services over mobile devices need to be taken into account in such service discovery mechanisms. These include and are not limited to battery and network constraints, limited computational power of mobile devices, dynamic service registry management. In such environments, traditional web service based SOA technologies such as WSDL, UDDI cannot be directly adapted. To the best of our knowledge, this work is the first attempt to comprehensively investigate these issues and design a service registry that facilitates service discoveries in mobile environments.

The other major issue in mobile environments is the 'volatility' of web services. With the ever increasing

• *Rohit Verma, and Abhishek Srivastava are with the Discipline of Computer Science and Engineering, Indian Institute of Technology Indore, India. E-mail: phd12110101@iiti.ac.in, asrivastava@iiti.ac.in*

numbers of mobile devices and web services hosted over mobile devices, it is not uncommon for hundreds of devices and in effect services to enter or leave the system at any instant of time. In this context, a centralized management of web services through service registries such as UDDI, is neither feasible nor practical. Chief among the reasons is failure of mobile device (due to network outage, battery issues, physical damages) prior to notifying the service registry. Further, in mobile environments relying on a centralized scheme is not the best of the ways. This is owing to the fact that a central registry is a central choke point, failure of this registry can make the discovery process moot. Moreover, with services updating their status frequently, there is high probability of the outdated information present in the registry. In this respect, a decentralized registry should be the way forward. This is because when multiple mobile devices maintain a localized registry, there is no central choke point. Further, the need to frequently update a centralized directory is eliminated. Moreover, in this scheme scalability can be handled quite efficiently.

There are two types of service discovery mechanisms: 1) Static Discovery or Design Time Search 2) Dynamic Discovery or Run Time Search. Service discoveries in mobile environment typically falls in the latter category, dynamic discovery [10]. The reasons being, as discussed earlier, the volatile and dynamic nature of mobile devices that often result in unavailability of the service provider, entry of new and more competent service providers, glitches in existing services, change of context of usage such as location, expertise. A legacy service directory system such as UDDI is inadequate for such frequent dynamic discoveries. This is due to the fact that it uses a very exhaustive data model, that is hard to analyze and parse for mobile devices at run-time.

Though service discovery is not a new problem, several works have been done to enable the service discovery in wired and wireless environments. However, to the best of our knowledge, they do not focus unorganized mobile service providers. Further they rely on traditional service registries for discovering services. This naturally results in scalability problems, increased network overload, and increased search time. To the best of our knowledge, mobile based service registry and service discovery is still untreated in literature. Therefore, our objective in this paper is to overcome these issues and facilitate service registry specifically for and over mobile devices. Moreover, we aim to realize the service oriented architecture over mobile devices without the involving high end servers. In the context of the issues outlines previously, we propose to employ a service registry in the form of a *roster* or *buddy-list*. The registry includes information on the web services along with their presence information articulated in the form of *available* or *unavailable*. The presence information or the availability of services is of utmost importance owing to the uncertainty associated with mobile devices.

The service directory of the proposed approach can be viewed as being analogous to the "buddy-list" in an instant messaging application. The proposed approach suggests managing a roster at each mobile device that act as a local service registry. This local service registry is traversed for any service discovery request, prior to search over the network. The advantages of managing a local roster are twofold: First, it improves the turnaround time for service discovery. Second it reduces data traffic over the network. This factor is important considering the battery constraints with mobile devices.

Moreover, the local service registry managed at the mobile device can be traversed by the external registry requester. In this paper, we have presented several operations to manage service registry and to provide registry's services to service requester, such as service discovery, service registration, service update, service binding, availability updates. These operations are performed using just three types of XML stanzas. Lesser numbers of XML stanzas have made the proposed architecture light weight and suitable for the mobile devices. Hence, the proposed architecture would provide all the registry related information and operations using mobile devices itself, without requiring any high-end computers or high management cost. Further, in order to support scalability, fault tolerance, and fault localization, we propose a distributed and category based service registry. We have categorized registered services in the form of service groups. The mobile devices that manage service registries of same service group, are identified by a common identifier - *Group ID* and registers services in the service registries are identified by their unique identifier - *Service ID*.

To demonstrate the feasibility of the proposed approach in the actual deployment, we have engineered a prototype. To the best of our knowledge, a service registry for mobile device has always lacked a real world prototype. This work is the first, wherein we have designed a framework and have experimented with real mobile devices. We deployed heterogeneous and loosely coupled mobile devices in a collaborative manner to manage service registry along with native hosted services. We employ several mobile devices, owned by volunteers, and analyzed the performance of the prototype in real situations. Our approach suggests that each mobile device can act as a web service provider or service registry as well as service consumer. In this paper, we compare and contrast proposed approach and traditional UDDI approach for managing service registry from the perspective of mobile devices. Our evaluation have shown propitious results such as, approach have acceptable battery requirements, the data communication cost is less, approach shows promising scalability and approach does not hinder to the native applications of mobile device.

The rest of the paper is organized as follows. Section 2 presents unique mobile service registry requirements. Section 3 presents the proposed approach and design concepts. Prototype implementation details and inline

comparison with UDDI is presented in Section 4. Section 5 gives experimental evaluation of the approach, followed by Section 6 that presents Related Work. Finally Section 7 concludes the paper with a brief discussion on future work.

2 Mobile Service Registry Requirements

Web services hosted over mobile devices are mainly useful in peer-to-peer, personal or crowdsourced environments. Mobile device in such environments are mostly distributed arbitrarily and make service discovery and management of service registry a cumbersome process. The focus of this paper is therefore towards facilitation of service registry management in such volatile and dynamic environments.

The technologies from W3C [11] such as UDDI [12] (for service discovery) and WSDL are ill suited to such dynamic environments. UDDI provides a tightly coupled architecture comprising that UDDI data entities (businessEntity, businessService, bindingTemplate, tModel, publisherAssertion, subscription), various UDDI services and API sets, UDDI Nodes for supporting node API set, UDDI Registries. This base architecture makes UDDI difficult to host on mobile and resource constraint devices. Hence, a new approach is required to facilitate service registry in mobile environments that take into account the specific nature and features of the environment.

Before discussing the proposed approach, we illustrate the requirements of service registries. There are several requirements for web service registries in general that are well discussed by Schahran et.al. [13]. Requirements unique to registries in mobile environments are discussed below:

R1: Management of transient web services: The very nature of mobile device makes hosted web services repeatedly and randomly enter and leave the network. A service registry should be such that it supports such arrival and departure of service providers, facilitating easy join, provision service and exit from the network.

R2: Easy integration: The registration and deregistration process for service providers on the registry should be easy and simple. At the time of registration, minimal information should need to be logged about the mobile service providers.

R3: Minimum communication overhead: Given the battery constraints in mobile devices, emphasis should be towards a registry system with minimum communication overhead.

R4: Distributed service registry: As the number of mobile devices is increasing exponentially, a centralized service registry system has limited utility and gets outdated very quickly. Hence, a distributed service registry system is required to support scalability of the mobile devices.

R5: Enabling run time search: An important factor in enabling mobile based service oriented system is the support for run time search. This is necessary owing to the regular availability of a new and more competent services and/or failure of the existing service providers.

Conforming to the above points could potentially ensure a service registry suitable for mobile environments.

3 Proposed Approach

In mobile environments, each mobile device can act as a service provider and a service consumer. As a service consumer discovers web services and invokes them after negotiating with the providers. As a service provider mobile devices host services and publish hosted services with service registries (as shown in Figure 1). Hence, the proposal of a dynamic service registry managed over mobile devices.

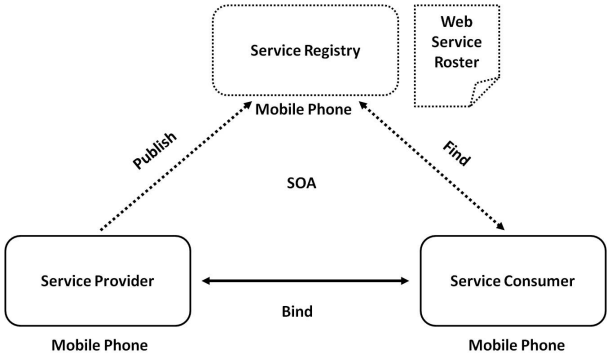

Fig. 1: Service Oriented Architecture Triangle in Mobile Environments

Our approach as discussed earlier, suggests managing a light-weight registry server at each participating mobile device. The approach comprises two types of the modules for service registry management:

1) Advertising Module: This module advertises the hosted service. The module is present at all mobile devices hosting web services. For advertising, this module generates the advertising query to update the service registry.
2) Query Responder Module: This module responds to incoming service discovery request by other mobile devices. The module keeps listening for the query requests and responds by passing the requested web service information from the service registry.

The service registry is managed in the form of a roster that includes among other details the availability information of services. The approach has defined operations of web services registries in the form of XML stanzas or messages. In this paper, we have scrutinized our approach on the basis of requirements discussed in section 2.

It is worthy that needs to be pointed out that the proposed approach is not limited to mobile environment, it also blends well with non-mobile service deployments. The proposed work is generic and independent of the web service implementation, hence can be used with

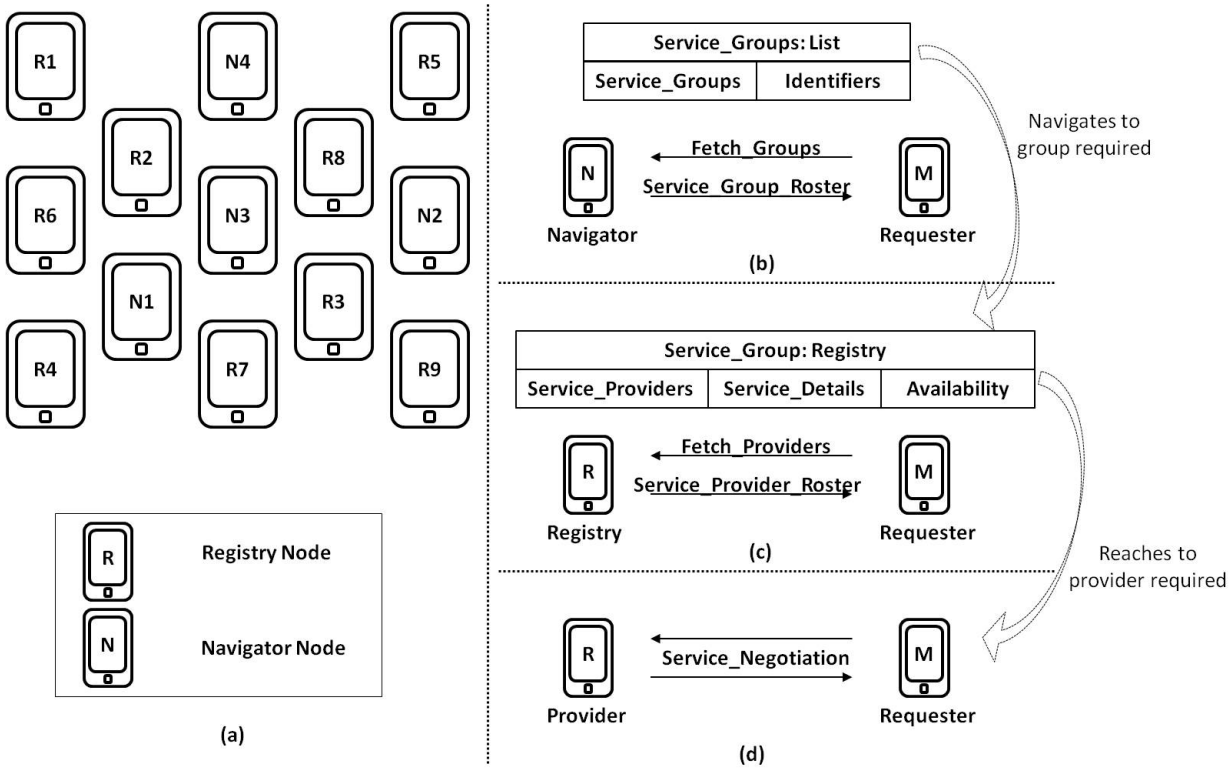

Fig. 2: Service Registry Architecture - (a) Roles of Mobile Devices (b) A requester fetches for service group at the Navigator Node (c) A requester fetches service required at the Registry Node (d) Requester contacts directly service provider returned from the Registry Node.

either SOAP based or REST based web services. Model provides registry service and subsequent service negotiations would be between consumer and provider. This service negotiation is out of the scope of this paper.

In next section we discuss about the design concepts of the architecture.

3.1 Design Concept

As stated earlier, in the proposed approach, each mobile device (hosting a web service) participates to provide and manage the service registry in a distributed manner. A mobile device potentially performs the following roles in our approach: Navigator Node and Registry Node (as shown in Figure 2.a). Navigator nodes comprise the list of service groups along with their identifiers offered by various service providers. The registry nodes represent service providers and their service details that help in managing the distributed service registry.

1) Navigator Nodes: Navigator nodes manage list of service groups present in the network. These service groups have an identifier that is somewhat analogous to a *group ID* in an e-mail system. We rely on existing ontological approaches [14] [15] [16] to categorize the registered mobile services on the basis of their domains of offering (or offered service type) and to manage these service groups at navigator nodes. For example, service providers hosting services for: a.) Latest score of a *football* match and b.) Statistical facts of a *football* match, share the same service group: *football* and are identified by the same identifier. These identifiers can be assumed to be pointers to the service registry of a specific type of service group.
2) Registry Nodes: Registry nodes manage the distributed service registry using dispersed mobile device in the form of rosters of the registered web services. There could be multiple registry nodes that belong to a service group, hence these registry nodes share the same *group ID* as the service group. The service registry or service roster comprises service details (further discussed in Section 3.2) that are just enough to manage and identify the registered web services. Of these details, the real time availability information of the registered services is what mainly contributes to overcome the uncertainty of mobile environments. This availability information managed at the registry node gives the much needed reliability to the services hosted over the mobile devices.

As shown in Figure 2.b whenever a requester needs to access a service registry, it first fetches the service group at the navigator nodes. The navigator provides the requester a list or roster of the service groups along with the corresponding identifiers. Upon selecting required

service group, requester then contacts to the registry node of the service group as shown in Figure 2.c. Registry node provides the requester with the service roster and requester node finally contacts the available service provider from roster as shown in Figure 2.d.

A service requester can be an external mobile device or a mobile device that is itself a registry provider. If the *requester is external*, the discovery process starts from the Navigator node. If a *registry node is a requester* then it can look into its local roster for the service in the same service group or it can contact the navigator to check for other service groups. The local roster helps in reducing the turn around time.

3.2 Components

The major primitive components of the architecture are shown in Figure 3:

- Roster Engine: The Roster Engine is the heart of the architecture. It manages the web service roster (or registry of the web services) along with the availability information. The availability information includes whether a web service is available or not at an instance of time. This is an important feature of the proposed architecture, given the fact that the availability of a web service hosted over a mobile device is mostly uncertain, owing to the dynamic nature of mobile devices. The *Advertising module* as discussed in previous subsection manages web service roster using the Roster engine. Furthermore, every time a web service or a hosted mobile device becomes unavailable, the roster engine gets notified. This parses and manages the identifier for the service provider.
- Matching Agent: This component parses the result of the service query and evaluates them against the required parameter (if any). Every time a discovery request is made externally or locally at the roster, the result of discoveries is shortlisted by this component. Matching agent keeps track of the request and response by using "id" as shown in Figure 4. The matching agent matches the registered services to the various service groups as discussed previously.
- Query Agent: This component accepts and processes the queries from other mobile devices. Following are the main functionalities of this agent:
  - Query Processing: Incoming queries are accepted and processed. These incoming queries are the ones that seek mobile web services and their availability status.
  - Query Generation: When the native mobile device (Roster engine) needs to know the status of a remote web service or need to update the web service roster, the query agent generates the query on behalf of the mobile device.

  The query agent has an external interface. This interface is the point of contact for any communication from external mobile devices. The queries generated are sent to the external mobile device via this interface and responses are sent back through the same. The *Query responder module*, discussed in Section 3, is run in this component.
- Web Service Roster: The service directories are managed in the form of rosters. Each mobile device is equipped with the local roster that also acts as the cache of the roster and reduces the turnaround time for service discovery. The roster contains an entry for each service provider as:
  `<service_name, access_point, service_description, service_groups, availability, service_location, other_info >.`

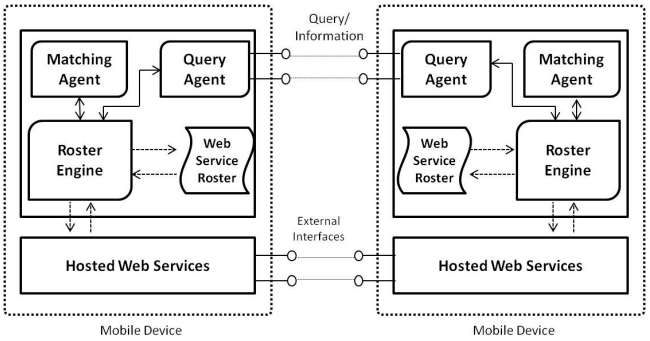

Fig. 3: Service Registry Architecture for Mobile Environment

3.3 System Specification

This subsection presents identifier and communication related specification of the architecture presented in 3.2.

Identifiers:

Each mobile device within the network is addressed by an identifier. We use two types of primitive identifiers: service identifier and service group identifier. Identifier is similar to an email address and is uniquely addressable. Format of an identifier is:

`[ServiceGroupID]@[DomainID]/[ServiceID]`

`DomainID` specifies the domain or network of a service registry, as there can be multiple service registries that are collocated globally. `ServiceGroupID` is the identifier of the service group. `ServiceID` identifies the services offered by the mobile service provider.

This enables mobile devices on the network to look out for only 'interested' web services. Service groups are analogous to the chat groups in chatting applications, where people with common interests can find and communicate with each other.

For example, a service group can be "movies" where offered services are related to movies. A service registry can be for the network XYZCity. This depicts the service group identifier as: `movies@XYZCity`. This service group is shared by all the service providers of service group movies. A unique identifier for a mobile service

provider offering services related to the "Latest movies" can be `movies@XYZCity/LatestMovies`. Moreover, domain identifiers could be chosen using an ad-hoc networking technology, such as *multi-cast domain name system* as suggested in RFC 6762, dynamic host auto configuration IP range as suggested in RFC 5735 and 3927. ".local" in the domain suggests the domain is local. These identifiers are managed by roster engines and query agents of mobile devices distinctly.

Communication:

We make use of XML stanzas for the service registry related communications. The term stanza is borrowed from [17], [18]. These stanzas comprise XML based discrete units of structured information sent over XML streams. There are mainly three stanza types used in the architecture: `<message />`, `<presence />`, `<iq />` [17], [18].

Message stanza works on the "push" mechanism. This stanza pushes information from one entity to another entity. For the purpose of managing and updating the service registries the message stanza is used. The message stanza may contain other information about the web service as shown in Figure 4. Message stanza is primarily used for storing and editing service information at the service registry. The binding information about the service by service provider also make use of message stanza.

Presence stanza is used to update the availability information of service hosted over mobile devices. Each presence stanza includes brief information on the hosted service (status message), along with its availability information. We use 'available' and 'unavailable' as the primitive presence types in the proposed approach. The status of a web service hosted over the mobile device is propagated through the roster engine and this is reflected on the web service roster (refer Figure 3).

IQ stanza is short for Information/Query stanza. It is based on a request-response mechanism and guarantees a response to a query. The nature of request in the IQ stanza is represented by `type`. Data content request is represented by `get` and it is similar to HTTP `GET` method. Any communication involving query agents primarily makes use of the IQ stanza. These play an important role in getting information from other mobile devices that host web service registry. Navigator nodes and registry nodes make use of IQ stanza for push and pull based requests for updates among themselves. This keep all the nodes updated.

4 IMPLEMENTATION

To analyze our approach, we implemented a prototype of the proposed approach and deployed it on real world mobile devices. The objective of using real mobile device is to measure and assess the applicability and the feasibility of the proposed approach in practical scenario. We designed the prototype to meet the requirements mentioned in the section 2. We performed a set of experiments to evaluate our approach, including various queries related to service registration, registry updates, runtime service and group discoveries incorporating dynamic mobile environments.

4.1 Prototype Detail

We developed a prototype for mobile devices in order to evaluate the proposed approach, using android SDK (ADT 23.0.2) and oracle java (version 1.7.0_72). The prototype was built to realize the architecture as presented in section 3. *The deployment of prototype neither require any modification to the device nor it needs root permissions to run*. We made use of general purpose smart phones and tablets (Samsung Galaxy S Duos with android 4.3, Sony Xperia M with android 4.3, Google nexus 7 with android 4.4, Asus zenfone 5 with android 4.4) for deployment. We devised our prototype to provide: i) Seamless intercommunication among various mobile devices and other computer systems. ii) Handling service registry and related operations on mobile device itself without any dependence on high-end systems. iii) Minimum hindrance with existing system and hosted web services. The prototype is independent of the native application server and web server that is required to host the web service. However, it makes use of separate socket to provide registry services. In our prototype, we made use of XML streams and documents to exchange and manage communication.

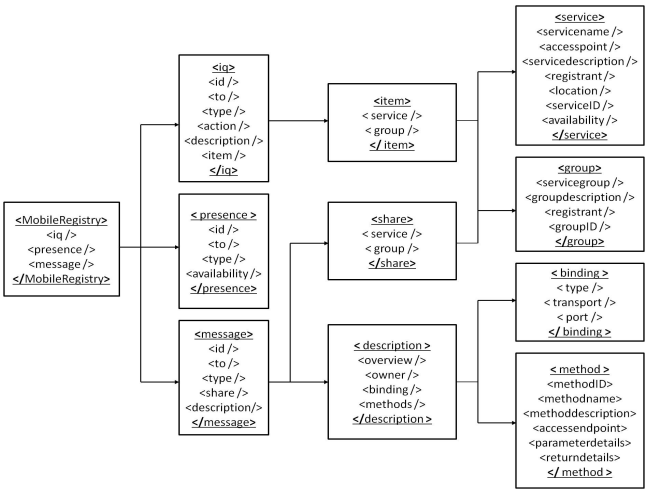

Fig. 4: XML Streams Used in Proposed Approach

Figure 4 shows XML structure used for communication in the prototype. As discussed previously in section 3.3, we make use of three primitive stanzas: IQ, Presence and Message. Each communication stream is having unique identifier shown by 'id' in the Figure 4. Tag 'type' represent the type of communication stream with the values: `get`, `set`, `result`. The XML stream have been discussed in detail in our previous work [19]. In the subsequent section, we discuss about the various operations performed by our prototype.

4.2 Prototype Operations

In this section we describe the operations and functionalities engineered in the prototype to provide service registry services such as registration, discovery, service updates, and service binding. An inline comparison with UDDI is also presented.

4.2.1 Registration:

In the our prototype, we have two types of registrations: 1) Service Group registration (at navigator node) 2) Service registration (at the registry node).

Service Group Registration : Service group is registered at the navigator node. A mobile device requests the navigator node to register the new service group that is not yet offered in the network. Query agent generates the query using IQ stanza (as shown in Figure 4) for sending the service group registration request. Communication is initiated using IQ stanza with 'type' value as 'get'. Navigator responds with the 'type' as 'result' and list of required information for group registration. Upon the successful registration a "groupid" is sent to the registered mobile device. The same is updated at the roster engine via query agent.

Service Registration : The roster engine on behalf of the hosted web service sends a request for registration to the query agent. The query agent sends the IQ stanza to the registry node of the service group. The service registration operation is similar to group registration, the only difference being in 'serviceID'. After the completion of the registration process the web service is updated to the service roster. This roster is updated (pull based manner) with other devices as required.

Web Service Registration in UDDI:

The registration process in traditional UDDI is done using the *publisher APIs set* exposed by the UDDI, such as save_service, save_business, save_binding, save_t_model. These APIs are used to save detailed information on the web service, which may not be necessary in case of mobile based web services. Moreover this information would tend to become heavy for mobile devices to process or transport.

Figure 5 shows the information stored in a typical UDDI registry [20]. The information is produced by the web service provider to the UDDI registry through publisher APIs (The information is transported as XML tags, we are not showing the XML for the UDDI structure owing to space constraints here. Interested readers may refer: http://goo.gl/cn8VaP). Deploying such a UDDI registry over a mobile device would tend to become heavy owing to limited computational power and network constraints. The UDDI registry would also lag in managing the dynamic nature of mobile devices.

4.2.2 Web Service Discovery:

Web service discovery in the prototype is meant to suit machines as well as humans. Humans can search and select a web service from the service roster, which is similar to a buddy list in instant messaging application. The web service directory is managed in the form of a roster along with the presence/availability information on the services. This feature of presence information is pertinent to dynamic mobile environments. Multiple mobile web service providers join and leave the network dynamically. We therefore manage a web service roster on each device. This enables the mobile devices to perform service discovery locally. If information on a web service is not available in the local roster, the discovery request can be passed to the other registry nodes via the query agent. This communication is analogous to [21].

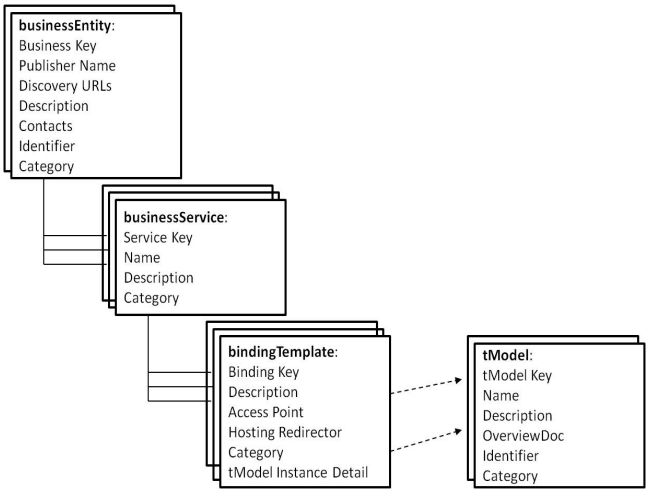

Fig. 5: UDDI Registry Entry

We devised two types of service discovery: Direct Discovery and Category based Discovery. These service discoveries are performed using the IQ stanza. Direct discovery is performed by parsing the local roster and is based on the required web service from a provider. An example of direct discovery is discovering the web service in local roster providing "weather statistics". In direct discovery, roster engine searches for the service in service roster and final result is matched in matching agent. Category based discovery for a web service is performed, when a list of available services of a particular category is required. The IQ stanza with 'type' value 'get', 'action' tag with value 'query' and 'description' tag with query description is used. The response have the 'type' value as 'result'. Once the web service consumer has discovered the appropriate provider, the consumer receives the binding information from the provider itself.

Web Service Discovry in UDDI:

Web service discovery in a UDDI registry is done via public inquiry APIs of the UDDI, such as find_service, find_binding, find_business, find_tModel. The service discovery is performed centrally by the UDDI registry server, which requires high computational capability. This is because the consumer requests the UDDI registry

server which in turn does the query search centrally and responds to the consumer with the results. The complexity and structured nature of the UDDI data structure as shown in Figure 5 further makes searching difficult. Processing a discovery query over a mobile device using UDDI would therefore be cumbersome. Moreover UDDI registries make use of deployed database systems. (For example Apache jUDDI makes use of Derby database (as packaged component) for managing and providing service directory.) These database systems would drastically slow down the mobile devices. Though traditional UDDIs enable consumers to query the registry and are effective in centralized system, they are ill suited to mobile environments.

4.2.3 Web Service Binding:

Web service binding information is necessary to use a particular web service. It includes the technical information on a web service, such as the access endpoint, required parameter values, return type. In the prototype, 'message' stanza is used for this purpose. Once a web service provider is discovered by the service consumer, the consumer retrieves the binding information from the provider. Proposed approach enables management of the service registry independent of the type of service implementation (SOAP or REST). This helps in lightweighted service registry and abstracts registry from the change of web service at the provider side.

Service binding description is specific to a service and would vary between web services. In the XML stream used in prototype (shown in Figure 4), minimum information is included from mobile device perspective, more details could be included depending upon the nature of service. However this information is between the service provider and the service consumer.

Web Service Binding in UDDI:

In the case of a UDDI registry, the service binding information is retrieved from UDDI registry server using t_model and WSDL documents. Moreover, several service providers do not register with UDDI registries due to the complexity involved (or due to unavailability of global UDDI registry). Global UDDI registries are also not updated owing to the fact of broken SOA triangle [22]. Hence, as a general practice, consumers/developers performs service/binding information discovery via web search engine with query parameter for *filetype* as *wsdl* (for SOAP based web services) or *wadl* (for REST based web services) for an unknown and new web service. Some query may result in multiple technical and binding documents. Selection of an appropriate web service may need human intelligence and analysis. Furthermore result selection may be dependent on ones choice and analysis. Besides, retrieval of wsdl/wadl document from service provider is also a common practice.

Our architecture eases mobile web service consumers from manual search and analyzing multiple non-relevant binding information of various web services. Moreover, discrete web services from new service provider are presented at a common platform. Web service consumer/developer can search services from the service roster directly or in category.

TABLE 1: Summary of Operations Performed in Proposed Approach and UDDI.

Operations	Our Approach	UDDI
Service Registration	IQ Stanza	APIs: save_service, save_business, save_binding, save_t_model
Service Discovery	IQ Stanza	APIs: find_service, find_binding, find_business, find_tModel
Service Binding	Message Stanza	t_model and WSDL documents

Table 1 summarizes the discussed operations performed in proposed approach and UDDI. Subsequent discussed operations, such as presence notification, roster sharing and roster update are specific to our approach.

4.2.4 Presence Notification:

This is one of the novel features of the proposed architecture. In this work, presence information and availability information are used interchangeably. The service roster manages presence information for a service. This presence information is managed for each service identifier. This service identifier may identify a web service or service provider. The presence information is similar to the availability information in an instant messaging application. The presence is shown as 'Available' or 'Unavailable' within an availability tag. This request is of type "set" and it does not seek any confirmation from the registry. The 'presence' stanza is used for this operation.

A node queries the registry node for the current availability status of the required service provider. This query has the type attribute value in the presence stanza as "get" The query is received by the query agent (discussed in Section 3.2) at the registry node. The query agent processes the query request and forwards the query request to its roster engine along with the required service. The roster engine parses the service roster for the required service and returns the current availability status to the query agent. This response (presence stanza with type as "result") is sent to the requester by the query agent. The response from the registry node reaches the query agent of the requester. The response is then processed by the query agent and the response is forwarded to its roster engine. Thereafter the same is reflected in the service roster. A similar procedure is also employed in [23].

Another possibility in this regard could have been a system for event triggered presence notification. Presence notifications are generated for the service directory, on the occurrences of events that cause the service to become unavailable. A few examples of these events are lower battery levels of the mobile devices, drop in

network availability, critical overload of requests at service providers. These events can be easily implemented programmatically using API's exposed by modern operating systems of mobile devices [24] [25].

4.2.5 Roster Sharing:

Roster sharing is required for managing the latest information on services in rosters of various registry nodes. The "message stanza" is used to perform roster sharing. Message stanzas are exchanged for: Addition, Deletion, Modification of service items to service roster. We suggest to make use of "action" attribute with add or delete or modify values. The message stanza has a "share" tag, which contains services or groups for sharing.

Roster sharing facilitate distributed service registry to share among various nodes. This sharing is of particularly more importance, when a new mobile device joins the service group. In the move to provide, the service registry to the new joiner the message stanza is exchanged. Any modification on the service roster of the registry node is also updated in the group using action attribute as discussed earlier. This functionality is extensible and can be adopted for timely updates or event driven updates to manage synchronization in the service registry of the various registry node.

4.2.6 Roster Update:

As our approach is mobile service provider-centric and mobile environment is dynamic, hence a service provider tend to change its configuration on the run. Roster is required to update whenever context of service is changed, such as location, access point. service descriptions. For the roster update IQ stanza with 'type' as 'set' and 'description' tag with 'info' value is used.

Unregister: Unregister is a roster update with IQ stanza having 'action' value as 'delete'. Mobile service providers are free to discontinue their service offering. Whenever a service provider do not want to provide hosted service or started new service, they unregister themselves from the service registry using 'delete' action. Hence, we have devised the unregister operation in our prototype.

4.3 Experimental Setup

Our experimental setup consists of three mobile devices (including Samsung Galaxy S Duos with android 4.3, Sony Xperia M with android 4.3, Google nexus 7 with android 4.4), one laptop (Intel i3 2.13 GHz with 3GB of RAM) and few running instances of prototype running on virtual instances of android devices running on laptop. These devices had the engineered prototype. The setup also had multiple services and service groups. All our experiments were carried for varied network size with multiple service providers joining and leaving the network.

Though stanzas in our approach apparently resembles with the XMPP, however we developed our prototype from scratch. The reason being, XMPP servers such as apache Vysper [26], openfire, ejabberd provide surplus functionalities that adds to the various overheads (battery, memory, data communication, CPU) when deployed on the mobile device.

The comparisons shown in section 4.2 were done in restricted lab environment. For the UDDI registry server jUDDI [27] version 3.1.5 was used. We then deployed UDDI server on a Cent OS Linux Server 6.2 with Apache Tomcat version 6.0.26. This server has Java run time environment version 1.7.0_45. SoapUI [28] version 4.6.3 was deployed on a Windows 7 machine for invoking and inspecting web services offered by UDDI, this simulated web service consumer behavior.

5 Evaluation

We have demonstrated the basic concept of our approach and the idea of managing service registry over "average" mobile devices that would facilitate the implementing an SOA in a mobile environment in an earlier paper [19]. To evaluate our approach, we have implemented the proposed architecture over real mobile devices. We solicited volunteer participation to host our prototype over their personal mobile devices. This enabled us to analyze the feasibility of our approach in a practical scenario. We also made use of existing an android application [29] to trace the battery consumptions by the prototype and in effect establishing its practical efficiency. We established an experimental wireless network within our institute building to connect the volunteers' mobile devices, laptop, and virtual instances. During the experiment, volunteers were doing their routine work and hence the mobility of the devices followed random patterns. We repeated this experiment with varying numbers of service providers, service registration requests, service discovery queries, with the intent of emulating uncertain situations in practical scenario.

The first experiment evaluated the effect that hosting a registry server had on the battery of the mobile device. For this, we observed the battery during: 1) An idle phase: when the registry nodes were idle and no requests were sent to them. 2) Working Phase: When suddenly multiple devices tried to register their multiple services on a single registry node. For analyzing the *idle phase*, we started our prototype comprising both registry/navigation node (Server_App)(Figure 6) and a registry client (Client_App)(Figure 7), and observed its effect on the battery for 24 hours as shown in Figure 8. We found the total battery consumption to be less than 1% of the total (by all applications) during the idle phase.

During the *working phase*, we sent (50*4=) 200 new service registration requests and (50*4=) 200 new service group registration requests from other mobile devices and virtual instances to a single mobile node. We performed this experiment over a time interval of 30 minutes. Though the bombardment caused an increase in usage of the battery, this was well within the accepted

norms shown. As shown in the Figure 9, the battery consumption by the prototype was 17.8% of the overall consumption during the experiment. Our experiment caused a total reduction in the overall battery from 75% to 66% (as shown in Figure 8 and Figure 9), hence the effective consumption by our prototype was 1.6% of overall battery strength (17.8% of the overall reduction caused from 75 to 66 i.e. 9). Also noteworthy here is the fact that the mobile devices were continuously moving with the volunteers, hence the devices were randomly joining and leaving the network.

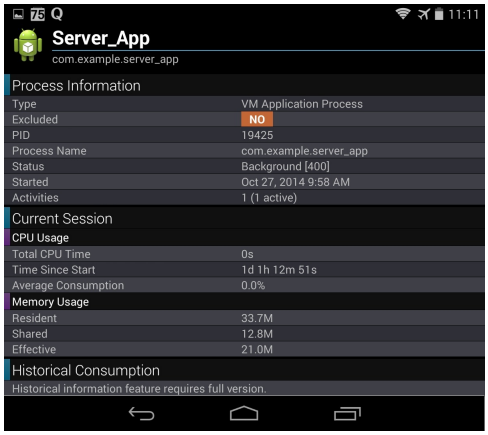

Fig. 6: Registry Server of Prototype during Idle Phase

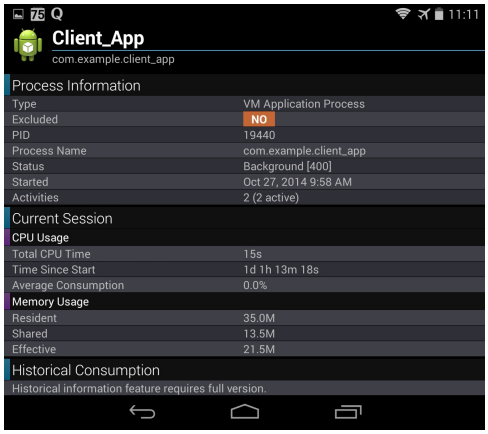

Fig. 7: Registry Client of Prototype during Idle Phase

The second experiment evaluated the effect of directory size on the discovery time. We registered multiple services in a group on the registry node. In order to test the scalability of our prototype, we ran service discovery operations for various directory sizes: 100, 250, 500, 1000, 1500. We discovered that the discovery time increases as the size of the directory increases. 1500 mobile web services were registered with the registry node for the experiment. We sent 20 requests from four mobile devices and virtual instances that made a total of (20*4=) 80 discovery requests for each directory size and the response time was calculated for these requests. Subsequently, we unregistered 500 services and

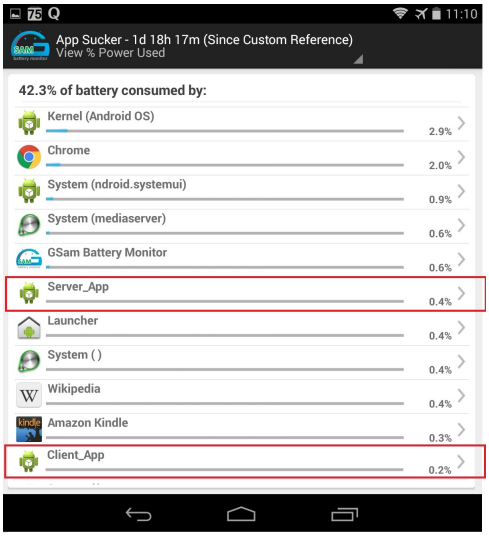

Fig. 8: Battery Behavior during Idle Phase

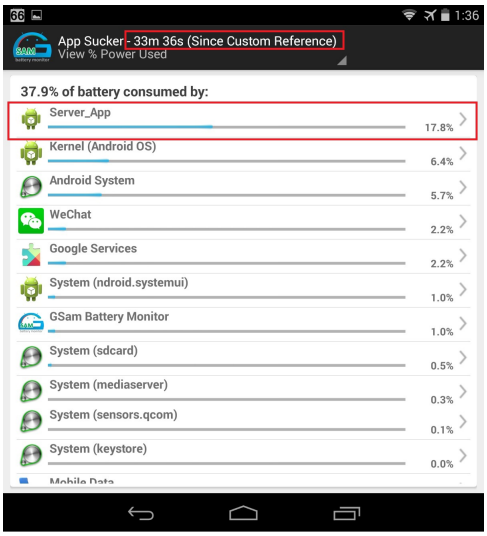

Fig. 9: Battery Usage During Working Phase

performed the same experiments with left 1000 registered mobile services at the registry. The process was repeated until we had 100 registered services left in the service directory. Through the whole experiment, the mobile device acting as the registry node was moving continuously within the network. Figure 10 shows the average response time for requests to service directories of various sizes.

Our third experiment aimed at evaluating the feasibility of the proposed approach. In this experiment, we conducted a comparative study on difference in response to requests - when a volunteer is answering a phone call and when the device is idle. For this experiment, we sent 50 requests for group registration on the volunteer's mobile device. First we observed the response behavior of the device when it was idle. During this phase the volunteer was randomly moving within the network. Next we made a phone call on the volunteer's mobile phone and observed the response time during the call. The

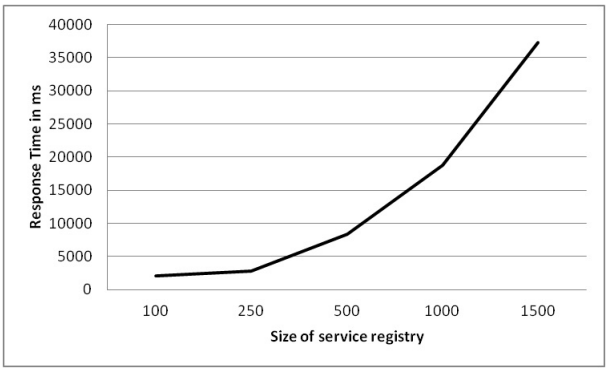

Fig. 10: Scalability of the Service Registry per Group

results demonstrate that there is a change in response time but this change is well within acceptable norms. There is an initial peak in the response time when the call is made. From the initial results, we could conclude that the mobile device takes a little time to respond to the first discovery request. This is due to the fact that some processing time is required to *awaken* the sleeping mobile application. We feel, therefore, that *piggybacking of incoming requests* could be a good approach to reduce the energy overhead.

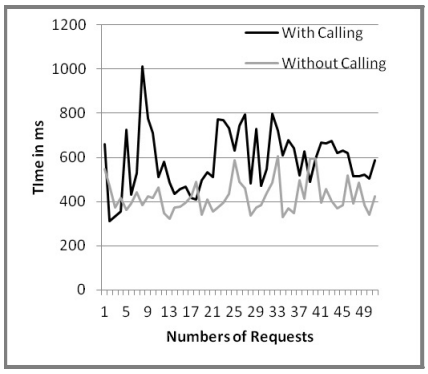

Fig. 11: Response Difference: With Calling and Without Calling

The fourth experiment involved an evaluation of the reliabilities of the approach. We toggled the availability status of service provider from *available* to *unavailable* and back to *available* with a time difference of 10 seconds. These toggles were repeated 1200 times at the registry node. During this time, we continuously probed the registry server from another registry client for the status of service provider. The initial results shown by the experiment have less than 1% false negative, where false negative implies - registry node returns availability information as *available* when the service provider has updated it to *unavailable* and vice versa. Although the scale of these experiments was limited, the results were promising. It will be interesting to explore the performance of this approach for a much larger numbers of service provider, and registry client and for much larger durations (few days).

Discussions

The proposed architecture caters the requirements of mobile environments. We have considered all the requirements as discussed in the Section 2 in our approach. We have proposed to manage availability information about each registered web service and dynamically updating it. This helps in satisfying requirement *R1* and keeping the service registry up-to-date irrespective of random entry and exit of transient web services. Further, we have made use of just three XML stanza in order to minimize communication overhead, thus satisfying requirement *R3*. During over experiments, we observed transfer of just few kilobytes for hundreds of request transfers. This minimized overhead also helped us in minimizing the battery utilization, that is reflected in Figure 9. Furthermore, we have used a single stanza "IQ Stanza" for easier service registration and deregistration. We seek minimal information that is just enough to uniquely identify and manage a web service from the registrant mobile device for registry related operations. This satisfied the requirement *R2*. The dynamic availability information, minimized data transfer, and faster response time helped in performing run time search(Requirement *R5*). Proposed approach manages the service registry over dispersed registry nodes, that satisfy the requirement *R4* and improves the fault localization. All these salient features, contributed in a light weight, autonomous service registry.

6 Related Work

With the parallel growth in mobile technology and networking technologies, web services for mobile devices is becoming a new paradigm. The idea behind mobile web services is to enable mobile devices to host, access, provide and integrate web services and information. Substantial work has been done on enabling mobile devices to access web-services [30] [31] [32]. On the other hand, however, hosting web-services over mobile devices is a relatively new area of research. The potential of mobile web services was first discussed by Berger et al. [33]. Work has been done on hosting web services over mobile devices [3] [4] [32] [5] based on SOAP as well as REST [34] [6]. AlShahwan et al. [7] provides a comparison between the SOAP and REST framework from the point of view of utilizing them in mobile environments. These literatures present an overview of the substantial work that have been done for mobile devices as service providers and service consumers.

Efficient service registry and service discovery is an important milestone on the way to integrate "Service oriented Architecture" and mobile devices. Service discovery has been well studied and still an active area of research. These discovery researches can be primitively categorized into two major structures, centralized and decentralized. Centralized service discovery is a well established area and have its foundation in the networking such as Service Location Protocols [35], Sun's

Jini architecture [36], Service Discovery Services [37], Microsoft's Universal Plug and Play (UPnP) [38]. These service discovery infrastructures rely on a central registry for the service discoveries. Moreover, these have a clear distinction between a server and a client. A request is sent from client to server regarding the service discovery and server returns a discovered service as a response to the client. One of the well known example of centralized service registry architecture for web services is UDDI (Universal Description Discovery and Integration) [12]. UDDI is the specification of a framework for describing web service, registering web service, and discovering web service [20]. Several data structure and APIs have been published for describing, registering and querying the web services by the UDDI. ebXML (electronic business XML) [39] standard is another example of centralized web service registry architecture. The decentralized service discovery started getting attention with the evolution of pervasive computing. There are several service discovery architectures available in the various fields (peer-to-peer networks, ad-hoc environment) of pervasive computing such as Chord [8], Pastry [9]. These architectures primarily make use of distributed hash tables to store the indices of nodes having resources. Decentralize discovery architectures can be very effective for the peer-to-peer systems, however, they may introduce an overhead in case of node failures or dynamic node arrivals.

Most of the work done on mobile web services utilizes a standard directory system with UDDI for web service discovery. A few group have explored fresh ideas for web service discovery in mobile environments. Recent work by Elgazzar et al. [40] discuss about the concept of personalized web service discovery based on the use of context information and user preferences. However they made use of existing WSDL documents for context-aware web service discovery. Al-Masri et al. [41] propose a device aware service discovery mechanism, MobiEureka, for mobile environments. MobiEureka make use of input keywords and recommends relevent mobile services to the client devices. Sapkota et al. [42] discusses use of shared memory for web service discovery. Their work provide a shared memory for web services to publish their service descriptions and make it available for service discovery. This approach could be adopted for mobile devices. However, these approaches lacks the efficient service registry that could facilitate the efficient service discovery, yet considers all the limitations specific to the mobile devices. Robinson et al. [43] proposes idea of dynamic service registries in context of mobile devices, but, relies on the centralized registry, such as UDDI. Kinga Dziembowski [44] states dynamic service discovery along with availability management in a trivial manner without any proof of concept. However, the proposed work focus on the service registry specifically designed for mobile devices. These service registries are managed and used by mobile devices itself, which is one of the novel features of the approach.

The goal of our paper is to propose a novel approach for managing the service directory for web services hosted over mobile devices. This would enable mobile devices to manage service registries on their own, drastically reducing the cost and dependency on infrastructure. This could potentially also help in providing services in dynamic networks such as vehicular networks, mobile ad-hoc networks (MANETs). For achieving this goal, we made use of existing protocols and technology that are well established for mobile devices. XMPP is one of the way for providing messaging functionality and provide chat list with near real time availability information. XMPP stands for Extensible Messaging and Presence Protocol [17]. It is an open Extensible Markup Language (XML) based protocol for near-real-time messaging, presence, and request-response services. XMPP is the formalization of the protocol developed by the Jabber open-source community in 1999. As per RFC 3920 [17], XMPP is the protocol for streaming XML elements in order to exchange structured information between two network endpoints in real time. XMPP is used predominantly in instant messaging (IM) applications. Various extension for XMPP have been proposed over time such as extension for serverless mode [45], service discovery mode [46], ad-hoc command support (ad-hoc session support) [47], SOAP support for web services [48]. Bernstein [49] proposed to use XMPP for intercloud topology, security. authentication and service invocations, this work is closely resembles to ours. However, our work focuses on the directory management in mobile based service oriented architecture. We focus on managing service directory in distributed manner on resource constrained mobile devices. Proposed work is an extension of our previous work [19].

We propose a buddylist based web service directory over mobile devices. This service directory contains minimal information about the registered services required for service discovery. To the best of our knowledge, this is the first instance where an architecture is being proposed for web services directory over mobile devices.

7 Conclusions

In this paper, we have investigated an important aspect for facilitation an SOA in mobile environments i.e. 'Service Registry'. Our studies show that traditional approaches for service registry (such as UDDI) can not be directly adopted in mobile environments, owing to its dynamic, volatile and uncertain nature. We have proposed a novel approach to manage service registries 'solely' over mobile devices. The approach considers several issues specific to mobile environments and enables run time service discoveries.

We have evaluated our approach by designing a prototype and deploying it on real mobile devices. To make our evaluation closer to real world usage, we asked volunteers to deploy our prototype on their personal mobile devices and continue their routine. The experimental results indicate that the proposed solution is

an efficient enabler for SOA in mobile environments. We have performed several experiments to confirm this such as, time performance, battery consumption, effect of nomadic behavior within the network on service registry and discovery.

Future work will be aimed towards QoS aware mobile based service registries that focus on QoS factors unique to mobile environments. We also plan to extend this work towards service registries specifically designed for Service oriented crowdsourcing.

ACKNOWLEDGMENTS

The authors would like to especially thank Deepak Sattiraju, for his commendable efforts and contributions in engineering the prototype. We also thank Tanveer Ahmed, Dheeraj Rane and Abhinav Tripathi for their assistance in resolving technical issues and fruitful discussions. The anonymous reviewers of our previous work [19] have provided several valuable feedback and helped us in improving the design and quality of work. We would also like to thank the research community as a whole and open source software community, for sharing their works.

REFERENCES

[1] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *Internet Computing, IEEE*, vol. 9, no. 1, pp. 75–81, 2005.

[2] M. P. Papazoglou and W.-J. van den Heuvel, "Service-oriented computing: State-of-the-art and open research issues," *IEEE Computer. v40 i11*, 2003.

[3] S. N. Srirama, M. Jarke, and W. Prinz, "Mobile web service provisioning," in *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*. IEEE, 2006, pp. 120–120.

[4] K. Mohamed and D. Wijesekera, "A lightweight framework for web services implementations on mobile devices," in *Mobile Services (MS), 2012 IEEE First International Conference on*, 2012, pp. 64–71.

[5] R. Tergujeff, J. Haajanen, J. Leppanen, and S. Toivonen, "Mobile soa: service orientation on lightweight mobile devices," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, 2007, pp. 1224–1225.

[6] L. Li and W. Chou, "Cofocus-compact and expanded restful services for mobile environments." in *WEBIST*, 2011, pp. 51–60.

[7] F. AlShahwan and K. Moessner, "Providing soap web services and restful web services from mobile hosts," in *Internet and Web Applications and Services (ICIW), 2010 Fifth International Conference on*. IEEE, 2010, pp. 174–179.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[9] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*. Springer, 2001, pp. 329–350.

[10] A. Zisman, G. Spanoudakis, J. Dooley, and I. Siveroni, "Proactive and reactive runtime service discovery: a framework and its evaluation," *Software Engineering, IEEE Transactions on*, vol. 39, no. 7, pp. 954–974, 2013.

[11] The World Wide Web Consortium (W3C), "W3c standards," [Last Accessed: October 30, 2014]. [Online]. Available: http://www.w3.org/standards/

[12] Oasis, "Uddi version 3.0.2 spec technical committee draft [last accessed january 30, 2014]," http://uddi.org/pubs/uddi-v3.0.2-20041019.pdf, 2004.

[13] S. Dustdar and M. Treiber, "A view based analysis on web service registries," *Distributed and Parallel Databases*, vol. 18, no. 2, pp. 147–171, 2005.

[14] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, "Web service modeling ontology," *Applied ontology*, vol. 1, no. 1, pp. 77–106, 2005.

[15] M. Sabou, C. Wroe, C. Goble, and H. Stuckenschmidt, "Learning domain ontologies for semantic web service descriptions," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 3, no. 4, pp. 340–365, 2005.

[16] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne *et al.*, "Daml-s: Web service description for the semantic web," in *The Semantic WebISWC 2002*. Springer, 2002, pp. 348–363.

[17] P. Saint-Andre, "Rfc 3920: Extensible messaging and presence protocol (xmpp): Core," Internet Engineering Task Force (IETF) proposed standard, Tech. Rep., 2004.

[18] E. P. Saint-Andrew, "Rfc 3921: Extensible messaging and presence protocol (xmpp): Instant messaging and presence, october 2004," Internet Engineering Task Force (IETF) proposed standard, Tech. Rep., 2004.

[19] R. Verma and A. Srivastava, "A novel web service directory framework for mobile environments," in *Web Services (ICWS), 2014 IEEE 21st Internation Conference on*. IEEE, 2014.

[20] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services: Concepts, Architectures and Applications*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[21] P. Hancke and D. Cridland, "Bidirectional server-to-server connections," 2012.

[22] A. Michlmayr, F. Rosenberg, C. Platzer, M. Treiber, and S. Dustdar, "Towards recovering the broken soa triangle: a software engineering perspective," in *2nd international workshop on Service oriented software engineering: in conjunction with the 6th ESEC/FSE joint meeting*. ACM, 2007, pp. 22–28.

[23] P. Millard, P. Saint-Andre, and R. Meijer, "Xep-0060: Publish-subscribe," *Jabber Software Foundation*, 2006.

[24] Android - Google Inc., "Android connectivitymanager," [Last Accessed: October 30, 2014]. [Online]. Available: http://developer.android.com/reference/android/net/ ConnectivityManager.html

[25] BlackBerry, "The network api," [Last Accessed: October 30, 2014]. [Online]. Available: http://docs.blackberry.com/en/developers/deliverables/34480/Network_API_Overview_1251781_11.jsp

[26] The Apache Software Foundation, "Apache vysper," [Last Accessed: October 30, 2014]. [Online]. Available: http://mina.apache.org/vysper-project/

[27] The Apache Software Foundation, "Apache juddi," [Last Accessed: October 30, 2014]. [Online]. Available: https://juddi.apache.org/

[28] SmartBear Software, "Soapui," [Last Accessed: October 30, 2014]. [Online]. Available: http://www.soapui.org/

[29] GSam Labs, "Gsam battery monitor," [Last Accessed: October 30, 2014]. [Online]. Available: https://play.google.com/store/apps/details?id=com.gsamlabs.bbm

[30] A. B. Mnaouer, A. Shekhar, and Z. Y. Liang, "A generic framework for rapid application development of mobile web services with dynamic workflow management," in *Services Computing, 2004.(SCC 2004). Proceedings. 2004 IEEE International Conference on*. IEEE, 2004, pp. 165–171.

[31] S.-T. Cheng, J.-P. Liu, J.-L. Kao, and C.-M. Chen, "A new framework for mobile web services," in *Applications and the Internet (SAINT) Workshops, 2002. Proceedings. 2002 Symposium on*. IEEE, 2002, pp. 218–222.

[32] M. Adacal and A. B. Bener, "Mobile web services: A new agent-based framework," *Internet Computing, IEEE*, vol. 10, no. 3, pp. 58–65, 2006.

[33] S. Berger, S. McFaddin, C. Narayanaswami, and M. Raghunath, "Web services on mobile devices-implementation and experience," in *Mobile Computing Systems and Applications, 2003. Proceedings. Fifth IEEE Workshop on*. IEEE, 2003, pp. 100–109.

[34] C. Riva and M. Laitkorpi, "Designing web-based mobile services with rest," in *Service-Oriented Computing-ICSOC 2007 Workshops*. Springer, 2009, pp. 439–450.

[35] E. Guttman, "Service location protocol: Automatic discovery of ip network services," *Internet Computing, IEEE*, vol. 3, no. 4, pp. 71–80, 1999.
[36] J. Waldo, "The jini architecture for network-centric computing," *Communications of the ACM*, vol. 42, no. 7, pp. 76–82, 1999.
[37] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM, 1999, pp. 24–35.
[38] B. A. Miller, T. Nixon, C. Tai, and M. D. Wood, "Home networking with universal plug and play," *Communications Magazine, IEEE*, vol. 39, no. 12, pp. 104–109, 2001.
[39] B. Hofreiter, C. Huemer, and W. Klas, "ebxml: status, research issues, and obstacles," in *Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems, 2002. RIDE-2EC 2002. Proceedings. Twelfth International Workshop on*. IEEE, 2002, pp. 7–16.
[40] K. Elgazzar, P. Martin, and H. S. Hassanein, "Personalized mobile web service discovery," in *Services (SERVICES), 203 IEEE Ninth World Congress on*. IEEE, 2013, pp. 170–174.
[41] E. Al-Masri and Q. H. Mahmoud, "Mobieureka: an approach for enhancing the discovery of mobile web services," *Personal and Ubiquitous Computing*, vol. 14, no. 7, pp. 609–620, 2010.
[42] B. Sapkota, D. Roman, S. R. Kruk, and D. Fensel, "Distributed web service discovery architecture," in *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*. IEEE, 2006, pp. 136–136.
[43] S. H. Robinson and R. C. Knauerhase, "Dynamic service registry," Patent 20 040 128 345, July, 2004. [Online]. Available: http://www.freepatentsonline.com/y2004/0128345.html
[44] K. Dziembowski, "Dynamic service discovery," *W3C Workshop on Web Services for Enterprise Computing*, 2007. [Online]. Available: http://www.w3.org/2007/01/wos-papers/gestalt
[45] P. Saint-Andre, "Xep-0174: Serverless messaging," 2008.
[46] J. Hildebrand, P. Millard, R. Eatmon, and P. Saint-Andre, "Xep-0030: service discovery," 2008.
[47] M. Miller, "Xep-0050: Ad-hoc commands," 2005.
[48] F. Forno and P. Saint-Andre, "Xep-0072: Soap over xmpp," 2005.
[49] D. Bernstein and D. Vij, "Intercloud directory and exchange protocol detail using xmpp and rdf," in *Services (SERVICES-1), 2010 6th World Congress on*. IEEE, 2010, pp. 431–438.

Rohit Verma is a PhD student at Indian Institute of Technology, Indore, India. He has received the B.Eng. and M.Tech. degrees in computer science in 2010 and 2012, respectively. His research interests include service-oriented systems, workflow management, mobile based and P2P web services, information security and crowdsourcing systems.

Abhishek Srivastava received the B.Eng. and M.Eng. degrees in Power-Electronics (Electrical) Engineering and Software Engineering in 2000 and 2002, respectively, and the PhD degree from University of Alberta, Edmonton, Alberta, Canada,in 2011. He is an assistant professor at Indian Institute of Technology, Indore, India. His research interests are in the area of service computing, crowdsourcing over mobile devices, Nature inspired approach towards web-service composition, Economic aspects of cloud computing, Demand-oriented architectures.