Personal Web Services: Architecture and Design

Technical Report 2012-597

Khalid Elgazzar, Hossam S. Hassanein, and Patrick Martin

School of Computing, Queen's University, Canada
Kingston, Ontario, K7L 3N6
{elgazzar,hossam,martin}@cs.queensu.ca

Abstract. Mobile Web services promise to provide users with more personalized service provisioning and real-time access to context information but the constraints inherent in the mobile environment and the lack of a robust architecture have contributed to their slow realization. The smartphone culture and its pervasive use, however, highlight the inevitability of mobile services. This paper introduces *personal mobile Web services*, a new user-centric architecture that enables service-oriented interactions among mobile devices that are controlled via user-specified authorization policies. Personal mobile Web services exploit the user's contact list (ranging from phonebook to social lists) in order to publish and discover Web services while placing users in full control of their own personal data and privacy. We present a proof-of-concept implementation of an example personal mobile Web service to demonstrate the usefulness and feasibility of the concept.

Key words: personal mobile Web services, Web services, service provisioning, mobile devices, mobile computing, smartphone

1 Introduction

There is no doubt that mobile devices are an integral part of our daily activities. Smartphones, in particular, become more prevalent due to their convenience and ability to provide pervasive and ubiquitous information access [1].

The successful convergence of resource-rich mobile devices, high-capacity wireless technologies and service-oriented architecture has the potential to redefine the mobile computing experience. It can introduce a new range of mobile applications that promise advanced mobile computing paradigms and seamless data access. Mobile services, where the mobile devices act as the service providers, have not yet seen wide adoption. This primarily due to the constraints imposed by limited resources and power, and intermittent connectivity.

This paper introduces *personal mobile Web services* (personal services for short), which are a subclass of mobile Web services that are not hindered by the above constraints on resources, power and connectivity. Personal services are intended to offer a range of user-centric data services to limited set of consumers that are explicitly authorized by the user providing the service. Personal services

therefore do not require the same levels of power and resources as general mobile Web services and are not required to be always available.

The privacy and security of personal information have been deemed, until recently, at the lowest priority of businesses [2]. Lately, personal data and privacy preservation have become a global concern [3]. The user-centric nature of personal services means that privacy is a key issue. The paradigm enables each user to play a pivotal role in controlling their privacy and their personal data communications. For example, personal service providers may expose reports derived from raw data in contrast to allowing access to the data itself. In addition, providers may allow access to their personal information based on different levels of access privileges. A user may categorize certain kinds of personal data as private and restrict access to this data to close friends and family members. At the same time, other kinds of data may be classified as public and made accessible to business partners or co-workers. **The contributions of this paper are as follows:**

- We introduce *personal mobile Web services*, which is a novel architecture for provisioning of user-centric Web services using mobile devices.
- We define the concept of cooperative service publishing and discovery for resource-constrained environments. The architecture takes advantage of the user's contact list to advertise and discover services.
- We demonstrate the feasibility and utility of *personal mobile Web services* through the development of a **S**mart contact **Li**st **M**anagement (SLiM) system, inspired by the proposed novel architecture.

The remainder of this paper is organized as follows. Section 2 presents the definition, architecture, design, and distinguishing characteristics of personal mobile Web services. Potential application domains of personal services are presented in Section 3. Prototype implementation issues and validation scenarios are discussed in Section 4. Section 5 concludes the paper and outlines future research directions.

2 Personal Mobile Web Services

2.1 Definition

Personal mobile Web services are lightweight user-centric services hosted on resource-constrained mobile devices. They offer personal and contextual information to a limited subset of authorized users, in a given period of time, based on a user-defined access control policy. They manage personal and contextual information for a user give the user fine-grained control over access to that information.

Formally, a personal mobile Web service s exposes a set of methods (Web resources) M to a list of users $l \subseteq L$, where L is the user's contact list, $L = \{c_1, c_2, \ldots, c_n\}$ and c is one of the user's contacts. Each method (Web resource) $m \in M$ has a set of access rights A_m set by the mobile user (provider) P. Each

contact c_i has a set of credentials R_{c_i}. A contact c_i is granted access to a Web resource m_j if $R_{c_i} \overset{match}{\rightarrow} A_{m_j}$.

In this work, we lay the foundations for the general concept of personal mobile Web services. Thus, in the remainder of the discussion, we consider that personal services are offered for public access without any constraints. However, we point out different scenarios where access constraints might apply, mostly for future considerations.

2.2 Distinguishing Characteristics

Personal mobile Web services are direct descendants of mobile Web services [4] and as such, they share both their advantages and constraints outlined in [5, 6]. There are, however,the following unique characteristics that give personal services advantages over mobile Web services:

- Personal services are primarily offered to people in contact list L which implicitly indicates that these services are accessed by a limited number of consumers. Therefore, the impact of resource constraints of mobile providers are less-likely affect the quality of service provisioning.
- Personal services place the owner (service provider) at the core of the service communications. Thus, the service owner P is in full control of how their data is being accessed by others by determining the access rights A for each method m and the privileges R for each customer c.
- Personal services are envisioned to typically include services such as personal profile management, contact list update, and photo/video sharing. Therefore, provisioning such services does not require continuous availability (which copes the impact of *intermittent connectivity*). Service providers and consumers may communicate with each other whenever a reliable network connection is available.
- Since personal services are hosted on mobile devices that are usually attached to users P with a personal contact list L, the idea is to exploit contacts in L to announce the existence of their services s, while consumers discover services provided by those in their contact list. Contacts cooperate, using their own resources, to extend the reachability of service providers when they announce their service existence or service requesters when the look-up a service.

2.3 Architecture of personal mobile Web services

In this section we present the conceptual architecture for personal services.

Figure 1 depicts the proposed architecture of personal services. The Web service is deployed on the mobile device, where an embedded lightweight Web server exists to provide the essential functionalities of HTTP-based service communications. In this architecture, the mobile device user/owner is the service provider and the service consumer is a direct or indirect contact of the provider. The mobile service provider advertises Web services to the members of his/her contact list. According to the settings of the service advertisement and provider

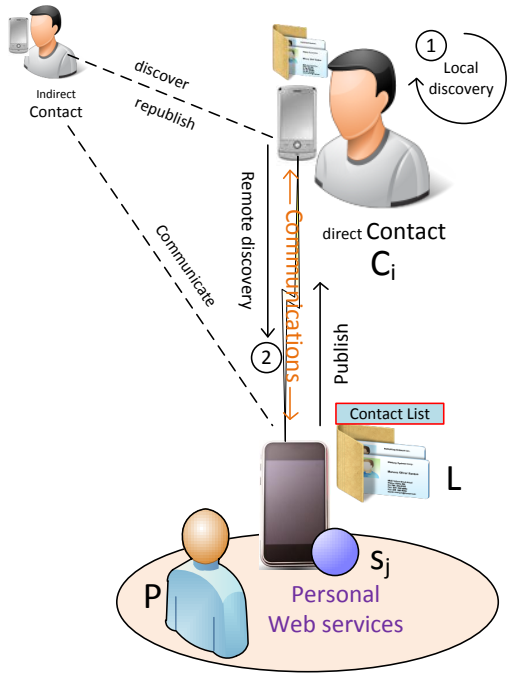


Fig. 1: A conceptual architecture of personal services

preferences, contacts who receive a service advertisement may be able to in turn distribute the service announcement to their own contact list. Personal service discovery looks up required services in the local service directory, if such a directory exists, (step 1). If no match is found, the discovery mechanism sends a service discovery request to people in the contact list (step 2) and may delegate to them the discovery task. The service communications (service request and service response) are performed through a direct link between the provider and a customer.

It is worth mentioning that, while personal services architecture is optimally designed for mobile devices, there is no barrier for a non-mobile based contact to participate, as it can be reached through either a mailing list or a social list.

2.4 Service directory

The standard Web service architecture uses a registry like UDDI (Universal Description, Discovery and Integration)[7] to support service publication and discovery. UDDI is a centralized service repository that manages and maintains Web services entries and keeps a reference for their description file.A discussion

of UDDI and a comprehensive comparison between different service registration and discovery techniques is given by Elgazzar et al [8].

For personal services, we adopt a distributed service directory approach, where each mobile device manages its own offered Web services and maintains references to services it knows about. Having a service directory is optional, for participants who only participate as service customers. From this perspective, there are two categories of services: *local services* and *remote services*. *Local services* are hosted and provided by the local system, whereas *remote services* are "active and running" services hosted on other mobile devices. Table 1 shows a possible table structure for the personal service directory.

Column	Type	Description
contactID	int (PK)	A system generated reference to the provider (contact) that services belong to
contactEndPoint	String	The provider's base Internet address
sID	int	Service id
title	String	Service title
description	String	Service description
endPoint	String	A reference to the service description file
status	int	0="active" (default), 1="inactive"

Table 1: A structure for the personal service directory

The *contactEndPoint* identifies the base Internet address at which the service provider (contact) can be reached. It sets the default for two other registry-related attributes: `contactEndpoint/registry/publish`, through which the service publication mechanism gets access to the service directory and registers services, and `contactEndPoint/registry/discover`, to which service discovery requests are sent.

The functions used in the service directory management to handle the basic directory operations as shown in Table 2, where *"object"* could be a service object or a list of services. It's worth noting that publishing is a system function and does not belong to the responsibilities of service directory.

Functionality	Return	Purpose
add(contactID, sID, title, description, endPoint)	int	adds a new service
update(contactID, sID, title, description, endPoint)	int	updates an existing service
delete(contactID, sID)	int	deletes an existing service
get(contactID, sID)	object	retrieves a service information
getall(contactID)	object	gets all services belong to a provider
status(contactID, sID)	int	queries the status of a service
search(capabilities[])	object	searches for services that match a list of capabilities

Table 2: The essential service directory functionalities

When a service is suspended or becomes invalid (not offered anymore), the provider sends out an update message with the service *ID* to either deactivate or delete the service. A service can be deactivated when the provider wants to temporarily suspend the service with a high possibility that the service will reactivated again. In order to avoid false discovery and maintain consistency, the provider, in all cases, must use the same service publication depth used with the service announcement to ensure that the message reaches everyone that holds a reference to the service.

2.5 Personal service publication

Service discovery is a crucial component in a Web service architecture, especially in heterogeneous mobile environments. Failure to find the services relevant to a user's objective renders the Web service approach useless. Limited resources on mobile devices present unique challenges for service discovery [8]. The standard Web service approach to discovery, which employs a registry like UDDI, has not been widely adopted [9]. Therefore, providers usually resort to other methods to publicize their services, for example, on their own websites.

In contrast, personal services take advantage of an individual's contact list to announce their existence. Such an approach enables the service provider to selectively assign access rights to Web service resources. Hence, the service might be offered to only a subset of the contact list, perhaps with different access privileges.

To facilitate personal service publishing and discovery using the contact list, we assume that each participating contact is reachable via an Internet address. To achieve this we propose to add an entry *contactEndPoint* for each contact to indicates how this contact can be reached over the Internet. This entry provides a unique HTTP address for the contact. If *contactEndPoint* is empty, then the contact cannot participate in service publishing or discovery.

The process of personal service publication using contact lists is shown in Figure 1. Once a personal service is deployed on the mobile device, a publication procedure announces the service's existence by sending a message to the contactEndPoint of each (authorized) contact in the provider's contact list.

The service's *sID*, *publicationDepth*, *title*, *description*, and *endPoint* are sufficient to define the service and how to obtain the service description and specification file. The *title* and *description* attributes are used by the discovery process to match a user objective. Hence, the *description*, in particular, should contain sufficient detail about the service functionalities to permit successful service discovery. The *endPoint* attribute is used to retrieve the service specification details. The *endPoint* address is in the form of a Web service resource with a generic format such as, `http://device-URI/service-root/specifications`. Although some broadband network providers offer fixed (static) ip address for mobile devices, perhaps for a charge, we remark that Internet addressing for mobile devices is out of this paper's focus. Nevertheless, there is a number of research efforts on how to get a personal Uniform Resource Identifier (URI) for mobile devices, viz. RFC6116 [10], which proposes a translation of a telephone

number into a URI using a special DNS record types. Hence, we assume that mobile devices are reachable online.

If we assume that a contact's personal information can be sent over the Internet in an XML file, then a service summary information (service announcement) could be part of such a file, or it could be sent in a separate message. The choice of whether to send all the contact information (including all offered services by this contact) or just to send the service portion separately depends on how frequently the service is updated and whether any other contact information is updated or not. Listing 1 shows an example of a separate service XML summary document fs_i for service s_i. Upon receiving a service summary message, the recipient of the announcement checks to see if the service exists in the service directory, and if so, updates the information. If the message is for a new service, the service and related information is added to the service registry. If the service provider is already an existing contact in the recipient's contact list, then a link between the service and contact is established. If not, the recipient would have the option to pull the provider's contact information, using the announced *contactEndPoint* associated with the service advertisement, and add the service provider as a new contact to the recipient's contact list.

Listing 1: Sample illustration of a personal service XML summary document fs_i.

```
<?xml version="1.0" encoding="utf-8"?>
<WebService>
  <contactID>id</contactID>
  <contactEndPoint>HTTP-address</contactEndPoint>
  <sID>sid</sID>
  <publicationDepth>d</publicationDepth>
  <title>title-str</title>
  <description>desc-str</description>
  <endPoint>HTTP-address</endPoint>
</WebService>
```

Algorithm 1 outlines the proposed publication mechanism for personal services. The publication process is distributed and recursive in that providers can allow contacts to propagate the publication of the service on their behalf using their own resources. The publication depth d indicates how far the service provider wants the advertisements to reach. The contact list L is initially set to the providers' contact list Lp. A contact c_i receives the service advertisement, reduces d by 1 and republishes the service to its contact list L_{c_i}, according to a prespecified pattern of access rights for indirect contacts. These access rights are set by providers via appropriate mechanisms. The publication stops when d reaches "0". If the service provider sets d to "0" for a particular service s, it means that s is only offered to the provider's direct contacts and the recipient of the announcement is not allowed to republish it.

Algorithm 1: Personal service publication.

```
Input: service XML summary document f_{s_i}
Output: null
Function Publish(f_{s_i})
Parse f_{s_i}
Set the provider P
Set the publication depth d
// start with the local contact list
L = L_{local}
foreach contact c_i in L do
    // check if service already exists in the service directory
    if s exists in sdir then
        update s
    end
    else
        add s
    end
    // check if forwarding is allowed
    if d > 0 then
        d = d − 1
        update d in f_{s_i}
        //delegate publication to the contact
        //distributed recursive call
        Call Publish(f_{s_i}) //at the contact side
    end
    else
        return null
    end
end
```

2.6 Personal service discovery

Personal service discovery begins with a service request (SR). The service request describes the required functionalities that fulfill a particular user objective in plain text. A simple feature extraction approach can be applied [11] to identify the required functionalities (RF) from a user request (SR) and to extract service capabilities (SC) from the service description field *desription* in the service header message f_s. Since Web service discovery is, by default, a resource-intensive process [12], the matching technique uses a keyword-based search to reduce the resource consumption. More sophisticated service discovery is unnecessary in this environment since it can be assumed that because services are being shared with a provider's contacts, it is likely that the contacts are familiar with the types of services that might be offered.

The discovery mechanism applies Algorithm 2 to find Web services relevant to a user's request by matching the required functionalities with capabilities offered by Web services. The algorithm is also distributed and recursive like the

publication algorithm. It ranks the retrieved relevant services ($RelS$) according to the similarity between the required functionalities by a user request and the offered capabilities by a Web service.

Algorithm 2: Personal service discovery.

```
   Input: Web service request SR, discovery depth d
   Output: set of relevant Web services RelS
 1 Function Search(SR,d)
 2 extract functionalities RF from SR
 3 // search local service directory first
 4 foreach s in sdir do
 5 |   extract capabilities SC from desription_s
 6 |   rank=match(RF,SC)
 7 |   add s to RelS indexed by rank
 8 end
 9 if RelS is null then
10 |   // check if deep search is allowed
11 |   if d > 0 then
12 |   |   d = d − 1
13 |   |   //delegate discovery to contacts
14 |   |   // using their own resources
15 |   |   L = L_local
16 |   |   foreach contact c_i in L do
17 |   |   |   //distributed recursive call
18 |   |   |   Call Search(SR, d) // at the contact side
19 |   |   end
20 |   end
21 end
22 return RelS
```

The *"match"* function applies the formula in Equation 1 to match the required functionalities with the offered capabilities.

$$match(RF, SC) = \frac{M(rf_i, sc_j)}{L_{RF}} \tag{1}$$

where $rf \in RF$ and $sc \in SC$, $M(rf, sc)$ is the number of distinct matched pairs between request functionalities RF and Web service capabilities SC, and L_{RF} is the total numbers of extracted functionalities from request SR.

The discovery algorithm assumes that services are publicly accessible. However, if services are provider-protected and some access constraints apply, then the algorithm would need to be augmented to limit service discovery to only those who have proper access rights.

3 Potential Application Domains for Personal services

Personal services open up opportunities for users who wish to share personal information and functionalities, yet still maintain full control over their personal data. The utility of personal services may extend to domains that are deemed beyond the capabilities of mobile devices. Such domains include:

Location-based applications. Most users carry their mobile phones with them for the majority of the time. While users move, they may offer access (with various privileges) to real-time context, such as location, air pollution levels, luminosity, and noise levels. Location-based applications are expected to benefit the most from personal services' ability to guarantee privacy under the provider's control.

Healthcare monitoring. Mobile devices may provide low-cost mobile and efficient remote health monitoring by utilizing the personal services approach. Mobile devices can collect a patient's vital signs using the sensors embedded in the mobile device, where applicable, or via communication with a body sensor network without interfering with the activities of the patient [13]. Using personal services, real-time data could be sent to caregivers or made available upon request to those identified within the patient's contact list. Personal services could also provide instant access to continuously changing context attributes, such as a patient's location.

Personal publishing. An author writing articles or blog posts may request feedback from contacts using personal services. While giving a lecture or presentation, a speaker could also receive questions and comments from the audience via personal services.

Mobile learning. In mobile learning (m-learning) scenarios [14], learners can share resources such as videos, audio, documents and comments. Participants in m-learning may also assume one or more roles including learners, mentors and peer-tutors. With personal services, participants can manage their own learning profile, including their progress and expertise, and share their experience with friends or learning partners while maintaining their privacy.

Personal information. Personal profile, photo/video sharing, and schedules are amongst the applications that can potentially benefit from the concept of personal services.

Personal social networking. A clique of friends may dynamically form a private social network while keeping all their personal information, social status, posts, and updates on their mobile devices [15, 16]. Personal services would enable users to maintain their social profile and allow friends to access their social information without sharing data with a third party.

4 Smart Contact List Management (SLiM): A proof-of-concept application

We demonstrate the feasibility and usefulness of personal services with an implementation of an application for automated contact list management called SLiM for Smart Contact List Management. Implementing this application with personal services benefits both mobile users and their contacts. A user's contact information is automatically kept up-to-date, as long as the contacts are reachable online. This saves users both time and effort from having to manually maintain their contact information.

The architecture of SLiM is illustrated in Figure 2. The basic idea is to let users each maintain their own record of contact information on their smartphone. Users then grant access to their contacts to perform requests such as pulling information from the contact list or updating their records in the user's contact list. For example, as shown in Figure 2, user Adam maintains his contact list and grants permission to his contacts John and Mike to search for and retrieve information from his contact list as well as update their records in his list.

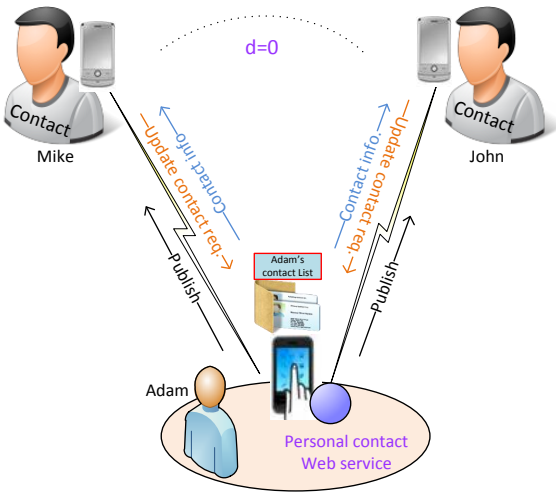


Fig. 2: A high level overview of the architecture of SLiM.

4.1 Contact List

Each mobile device user maintains a contact list of friends, family members, business partners, or others on his/her phone. A typical contact record has two main categories of information: contact-related information and user-related information. The former category includes information about the contact person

(or entity), such as a name and photo, and how the contact can be reached, such as a phone number, email, and address. The latter category describes the contact from the perspective of the mobile user based on personal preferences, such as the group to which this person belongs (eg. close friend, relative, business contact, etc), the user-assigned ringtone for this person, etc. Both categorizes are currently entered and maintained manually by the phone user.

SLiM aims to automate the maintenance of the first category while allowing the contact person (to whom this information belongs) full control of granting access rights, perhaps with fine-grained control levels. This automatic management of contact information might also open new opportunities to expand the contact information with new entries or parameters that enable taking smart actions. For example, adding time attributes to the *phone* entry would indicate that the person desires to be reached at that phone number only during the associated time slots.

The mobile phone-based contact list information contains basic information about contacts. Perhaps the reason, by design, is to ease the input burden on mobile users when they add a new contact given the typical input limitations of a mobile device. Table 3 shows the basic contact information on a typical mobile device address book, where S/M denotes whether the entry allows **S**ingle instance or **M**ultiple instances.

Entry	S/M	Options
Photo	S	Album, Take Photo
First Name	S	
Last Name	S	
Phone	M	Mobile, Home, Work, ...
Internet call	S	sip uri
Email	M	Gmail, Yahoo!, Hotmail, ...
Instant Messenger(IM)	M	Skype, Google Talk, Yahoo!, ...
Postal Address	M	Home, Work, ...
Organization	M	Work, Other

Table 3: Typical contact information on a phone-based contact list

Some smartphone platforms, such as Android, offer a consolidated contact list. It integrates contacts from multiple accounts, and from various underlying data sources (such as the phone address book, social network lists, and emailing lists) in a single place, the contact list. It also allows for combining different records that belong to the same contact into a single, aggregate record. These features render automatic contact list management even more beneficial.

4.2 Implementation details

As mentioned earlier, the main objective of the proof-of-concept prototype is to demonstrate the concept of personal services. A Web service that provides

the core functionalities of SLiM is developed in compliance with the "RESTful" principals; using the Python programming language. It is based on the guidelines of provisioning Web services from resource constrained devices proposed by Elgazzar [5]. The choice of Python is motivated by the fact that the standard Python library comes with a lightweight Web server that can provide essential HTTP functionalities. The Python-based REST Framework `Web.py` [17] is used to handle low-level details of Web service developments such as protocols, sockets, and process management.

The personal service is deployed on a Samsung Galaxy II smartphone with a rooted Android Gingerbread OS [18], connected to a WiFi network. On this mobile device, the user maintains a record of his/her own contact information, which the personal service exposes to the user's contact list. Table 4 shows the basic methods and functionalities, exposed by the Web service, pertaining to the user's contact details. Each method represents a Web resource that can be accessed using the HTTP-based resource identifier by an authorized user. The generic format of the resource address is `http://root-address/service-name/resource-name/[parameters]`. The `root-address` is the server (mobile) address, *contactEndPoint*, `service-name` is the name of the personal service, and the `resource-name` represents a service method name.

Resource	Relative resource address	Purpose
Service Description	/contactInfo/description	Get the service description and specifications
Contact Info	/contactInfo/details	Get the complete contact info
Phone	/contactInfo/details/phone	Get the phone portion of the user contact
Email	/contactInfo/details/email	Get the email portion of the contact info
Search	/contactInfo/search/[contactName,depth]	Search for contacts

Table 4: Functionalities and methods exposed by the personal service

The version of SLiM implemented here focuses on personal service provisioning and does not include the personal service discovery aspects described earlier. However, the search functionality that SLiM presents is very similar to the process of personal service discovery in terms of depth search and matching, given that participants are reachable online. For reachability purposes, SLiM employs an entry called *Internet Call*, which is an existing entry in Android's contact information that is supposedly dedicated to hold the contact SIP address information for Internet call, to hold the `contactEndPoint`. Eventually the entry `contactEndPoint` can be proposed as an expansion to the contact information.

SLiM employs an individuals' contact list to announce the existence of the personal service. It sets the publications depth d to "0" to limit the service announcement to members of the provider's contact list. Then, it initiates the publishing process for contacts with their *Internet Call* (contanctEndPoint) set. The default publication URI path that provides access to the local service direc-

tory for each contact is configured to be `contactEndPoint/registry/publish` as per the service directory settings.

In our implementation, the Web service offers different representations of the same service response, namely, *XML*, *JSON* and *HTML* using `mimerender` [19], which is a Python library for RESTful resource representation using MIME Media-Types. We set the *XML* format as the default representation when no HTTP "Accept" header is identified. Therefore, when a Web service resource is called, an XML-formatted response is dispatched to the HTTP request handler's result. Other formats (HTML, JSON) are created for testing purposes such that the Web service can be invoked via a mobile or a standard web browser (i.e. customer application) and the response is dispatched in HTML format.

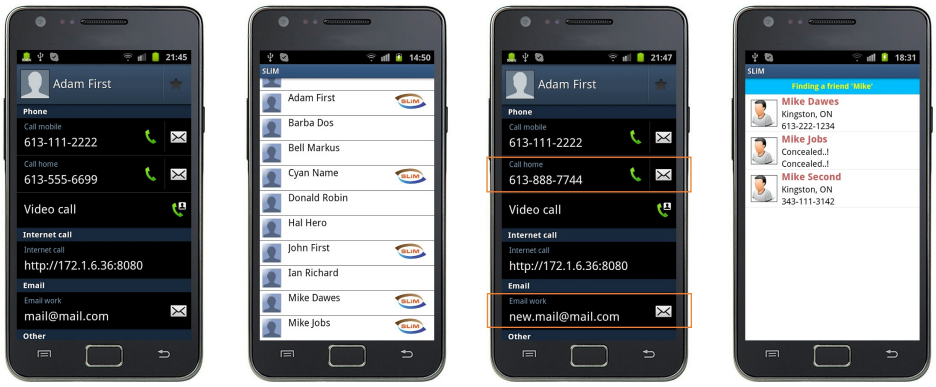
The service registry is implemented using SQLite [20] (the Android default Database engine). The built-in package `android.database` handles general database operations, while `android.database.sqlite` contains classes specific to SQLite [20].

A user interface (UI) for SLiM is developed on an Android platform using the Android SDK [21]. As mentioned earlier, we employ the existing contact entry *Internet Call* to act as the proposed extension *contactEndPoint*. Once this entry is filled in with the proper HTTP-based address, the associated contact is considered a participant in SLiM, whether offering personal contact information as a service, or receiving automatic updates from contacts about changes in their information. Figure 3b shows a screenshot of the UI, in which the contacts marked with a SLiM icon (to their right), have a valid and active `contactEndPoint` and are enabled for participation in automatic contact update. The new contacts API defined in `android.provider.ContactsContract` [22] is used to handle basic contact operations, *insert*, *update*, *delete*, and *Query*. When a new contact is to be inserted, the underlying Android system handles the insertion and checks to see if there is an existing contact representing the same person (or entity). If a match is found, then the system gets the contact's `CONTACT ID` and adds the new contact information. Otherwise, a new contact record is created.

4.3 Prototype Validation

We have carried out a number of experiments to validate the operation of SLiM in order to ensure that it functions as expected. In the first test scenario, *Adam* changes his contact information (in particular, his phone number and email address) on his smartphone. When *John*, one of *Adam*'s contacts taps the SLiM icon right after *Adam's* name, his contact information is automatically updated, as shown in Figure 3c.

The second test case examines the operation of the search function offered by SLiM. *Adam* exposes a functionality that enables his contacts to search his contact list. *John* is looking for *Mike's* contact information. *John* sends *Adam* a Web service request to look up *Mike*'s contact information (if found). The service request has the form `http://172.1.6.36:8080/contactInfo/search/Mike,0`, where `http://172.1.6.36:8080/` is *Adam*'s contactEndPoint, `contactInfo` is the personal service name offered by *Adam*, `search` is the method name, `Mike`



(a) Adam's contact information before update (seen on John's smartphone).

(b) SLiM presents contacts.

(c) Adam's contact information after update.

(d) Searching for contacts with SLiM.

Fig. 3: SLiM test case scenarios.

is the search term, and "0" is the search depth, which indicates that *John* only wishes to search his direct contacts (i.e. whoever offers the service). Utilizing the comma-separated approach in passing parameters to the HTTP request is merely an implementation issue.

Figure 3d shows the service response presenting the search results obtained from *Adam*'s contact list. The service request is sent only to *Adam*. Our current implementation to the search function presents the results (if found) with a contact name, photo, city, and phone, if the person shares his/her information. Otherwise, The system only presents the contact name and conceals all other information. Each row of the presented results is linked to the corresponding `contactEndPoint` that the requester can use to retrieve the contact information. When the requester clicks on the contact name, it sends a service request to that person to retrieve the contact information. We realize that not all contacts might be reachable online, but we have left that for future expansions.

5 Conclusion

This paper introduces *personal mobile Web services*, a new user-centric Web service architecture hosted on mobile devices (in particular smartphones). The personal services architecture takes advantage of the provider's contact list to announce service existence and discovery. Contact list members cooperate to disseminate service advertisement and discovery requests, if needed, using their own resources. The motivation of proposing such an architecture is twofold: overcoming the barriers of mobile service provisioning on resource-limited mobile devices and placing users at the core of controlling their personal data. Moreover, personal services, intrinsically, inherit the flexibility and dynamicity that mobile

services offer. We have discussed the domains and applications which can potentially benefit from such an architecture, both as currently applicable and in the future.

A prototype is developed to demonstrate the usability of the personal service architecture, and to depict how different tasks of personal service provisioning can be performed. The implementation of the prototype does not cover all the proposed aspects of the personal service architecture. The opportunities made possible by the development of such a prototype, and validation scenarios, have made mobile devices an increasingly attractive platform for every-day life tasks.

In this paper we have laid the foundation of a new Web service architecture for mobile devices that would expand the horizon of mobile applications and their domains. However, we believe that significant research is still needed to achieve the full potential of personal services. We plan to expand on many of the issues we have identified here in our future research. In particular, we plan to investigate the incorporation of access control aspects, ranging from service advertisement to limiting service access to those with sufficient access privileges.

References

1. Roussos, G., Marsh, A.J., Maglavera, S.: Enabling pervasive computing with smart phones. IEEE Pervasive Computing **4**(2) (2005) 20 – 27
2. Kirkham, T., Winfield, S., Ravet, S., Kellomaki, S.: A personal data store for an internet of subjects. In: The International Conference on Information Society (i-Society). (June 2011) 92–97
3. : European Union, The Stockholm Programme: An open and secure Europe serving and protecting citizens, Official Journal of the European Union, 2010, [Online]. Available: http://europa.eu/legislation_summaries/human_rights/fundamental_rights_within_european_union/jI0034_en.htm, [Accessed: Feb 25, 2012].
4. Srirama, S.N., Jarke, M., Prinz, W.: Mobile web service provisioning. In: The International Conference on Internet and Web Applications and Services (ICIW'06). (Febraury 2006) 120–126
5. Elgazzar, K., Martin, P., Hassanein, H.: A framework for efficient web services provisioning in mobile environments. In: The 3rd International Conference on Mobile Computing, Applications, and Services(MobiCASE'11), Springer's LNICST (October 2011)
6. Mizouni, R., Serhani, M., Dssouli, R., Benharref, A., Taleb, I.: Performance evaluation of mobile web services. In: The 9th IEEE European Conference on Web Services (ECOWS. (September 2011) 184 –191
7. Clement, L., Hately, A., von Riegen, C., Rogers, T.: Uddi version 3.0.2 (January 19 2004) http://uddi.org/pubs/uddi-v3.0.2-20041019.htm.
8. Elgazzar, K., Hassanein, H., Martin, P.: Effective web service discovery in mobile environments. In: P2MNETS, The 36th IEEE Conference onLocal Computer Networks (LCN). (October 2011) 697–705
9. Legner, C.: Is there a market for web services? In: Service-Oriented Computing - ICSOC 2007 Workshops. (2009) 29–42
10. : The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM), http://tools.ietf.org/html/rfc6116.

11. Elgazzar, K., Hassan, A.E., Martin, P.: Clustering wsdl documents to bootstrap the discovery of web services. In: The 8th IEEE International Conference on Web Services (ICWS'10). (July 2010) 147–154
12. Steller, L.A.: Light-Weight and Adaptive Reasoning for Mobile Web Services. PhD thesis, Monash University, Australia (May 2010)
13. Elgazzar, K., Aboelfotoh, M., Martin, P., Hassanein, H.S.: Ubiquitous health monitoring using mobile web services. In: The 3rd International Conference on Ambient Systems, Networks and Technologies. (August 2012)
14. Wang, C.S., Wang, Y.H.: Design of an soa-based ubiquitous learning environment. In: The IEEE International Conference on Granular Computing (GrC). (November 2011) 697–702
15. Church, K., Pujol, J.M., Smyth, B., Contractor, N.: Mobilehci'10 workshop summary: social mobile web. In: The 12th international conference on Human computer interaction with mobile devices and services. MobileHCI '10, New York, NY, USA, ACM (2010) 509–512
16. Brooker, D., Carey, T., Warren, I.: Middleware for social networking on mobile devices, Auckland, New zealand (2010) 202–211
17. : Web Framework for Python, http://webpy.org/.
18. : Android 2.3.4 Platform, http://developer.android.com/sdk/android-2.3.4.html.
19. : Mimerender Python Module, http://code.google.com/p/mimerender/.
20. : Android SQLite, http://developer.android.com/reference/android/database/sqlite/package-summary.html.
21. : The Android SDK, http://developer.android.com/sdk/index.html.
22. : Android: Using the Contacts API, http://developer.android.com/resources /articles/contacts.html.