SOCA (2016) 10:55–70
DOI 10.1007/s11761-014-0164-8

ORIGINAL RESEARCH PAPER

Personal mobile services

Khalid Elgazzar · Patrick Martin · Hossam S. Hassanein

Received: 11 October 2013 / Revised: 18 August 2014 / Accepted: 19 August 2014 / Published online: 17 September 2014

Abstract Ubiquitous information access through mobile devices has become a typical practice in everyday life. The mobile service paradigm shifts the role of mobile devices from consumers to providers, opening up new opportunities for a multitude of collaborative services and applications ranging from sharing personal information to collaborative participatory sensing. Although many basic principles of the standard Web service approach continue to apply, the inherent constraints of mobile devices and broadband wireless access render the deployment of the standard architecture in mobile environments inefficient. This paper introduced *personal services*, a user-centric paradigm that enables service-oriented interactions among mobile devices that are controlled via user-specified authorization policies. Personal 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. Experimental validation demonstrates the ability of personal services to foster a new generation of collaborative mobile services. Performance evaluation results show that the publication and discovery through contact lists are efficient and that service announcements and discovery requests can reach a huge number of users in a few seconds. Results also support a conclusion that resources-constrained devices can collaborate to carry out functionalities beyond the ability of their resources limitations.

K. Elgazzar (✉) · P. Martin · H. S. Hassanein
School of Computing, Queen's University, Kingston, Canada
e-mail: elgazzar@cs.queensu.ca

P. Martin
e-mail: martin@cs.queensu.ca

H. S. Hassanein
e-mail: hossam@cs.queensu.ca

Keywords Mobile services · Mobile devices · Personal services · Service oriented

1 Introduction

Mobile service provisioning is intended to serve interoperable functionality from mobile devices over the network. Although many basic principles of the standard Web service architecture continue to apply, the intrinsic limitations of both mobile devices and wireless networks render the deployment of such architectures inefficient. For instance, resource constraints of mobile devices limit the options of applying advanced service discovery mechanisms (such as semantic approaches), and the intermittent connectivity of wireless networks poses challenges for synchronous service communications. Current mobile service provisioning architectures are borrowed from traditional approaches that were designed for fixed networks and resource-rich computing infrastructures. Several adaptations have been added to these architectures to accommodate the characteristics of dynamic mobile environments; however, these adaptations were unable to satisfy the requirements of efficient mobile service delivery.

Mobile devices have access to a variety of contextual information through a wide range of embedded sensors. Mobile devices also are typically associated with users who have personal information that they might be interested in sharing with family members, close friends or business partners. Users maintain a variety of contact lists ranging from phonebooks and mailing lists to social circles. The mobile service approach offers users who wish to share personal information with privacy preservation or want to participate in offering public information such as crowdsourcing the opportunity to do so. As such, mobile service architectures do not need to

be constrained to traditional service architectures. New architectures need to be developed to take advantage of the unique features of mobile environments while accommodating their various constraints.

The privacy and security of personal information have been deemed, until recently, at the lowest priority of businesses [1]. Lately, however, personal data and privacy preservation have become a global concern. 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 the general public.

This paper presents a reference architecture for *personal services* that enables user-centric personal data sharing and crowdsensing services, while preserving the provider's privacy. Personal services may be offered for a set of consumers that are explicitly authorized by the user providing the service (e.g., healthcare and behavior monitoring) or for public access such as community sensing (e.g., environment and transportation monitoring). In this architecture, mobile devices act as service providers, bringing together the convenience of mobile devices, the benefits of real-time access to a wide variety of contextual information. The architecture takes advantage of the provider's contact lists to announce and discover services.

Personal services open up opportunities for users who wish to share personal information and functionalities, yet maintain full control over their personal data. Such services capitalize on the ability of mobile devices to access a wide range of context information in real time. Potential domains include location-based mobile sensing applications (e.g., environmental and traffic monitoring, weather forecasting, air quality, noise level, etc.), personal healthcare monitoring, cooperative mobile learning, and personal social networking. For example, personalized group tweeting is an interesting application, where a clique of users can construct a personal distributed mobile tweeting circle. Users maintain their own tweets locally and share with subscribed users (from their contact lists). Users collaborate to disseminate and search for tweets. More detailed descriptions about potential applications can be found in [2]. To the best of our knowledge, no previous research has addressed personal services from this perspective.

The contributions of this paper are summarized as follows.

- Presenting a user-centric architecture for providing personal services from mobile devices.
- Introducing 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.
- Demonstrating the feasibility and utility of *personal services* through the development of a Smart contact List Management (SLiM) system, inspired by the proposed architecture.
- Conducting performance analysis of the proposed personal service architecture and determining the overhead that personal services incur on mobile terminals, whether as providers or collaborators that forward service communications.

The remainder of this paper is organized as follows. Section 2 outlines related work. Section 3 presents the definition, distinguishing characteristics, architecture, and design of personal services. Section 4 highlights the access control scheme used to protect access to personal services. Section 5 shows a functional prototype to demonstrate the utility of personal services. Performance evaluation and overhead analysis are discussed in Sect. 6. Lastly, Sect. 7 concludes the paper and provides future directions.

2 Related work

The ability of mobile devices to access a wide range of context information is mostly utilized to personalize service delivery to mobile users [3]. However, this ability also opens the doors to using mobile devices as a personal service provisioning platform, offering access to both personal information and public sensor data in real time. Personal services take different forms, in all of which the user is the key architectural component. Users carrying their mobile devices enable opportunistic real-time service provisioning for public access or for a specific group of common interest. In this section, we review research efforts that involve mobile devices offering data services.

Fednet [4] enables users to share personal resources and services in a P2P fashion. In this approach, each user maintains his own personal network (PN) that may encompass distributed personal devices and provide access to personal resources through a single end point. These PNs belonging to different users cooperate to provide access to personal services. Augusto et al. [5] use mobile devices as a central authority to manage the user identity and authorization. The authors present an Open Federated Environment for Leveraging Identity and Authorization (OFELIA)

to provide user-centric identity management to control distributed personal data and services that can be run by different providers.

Mobile sensing (or crowdsensing) is becoming a popular approach that offers personal-based or community-based sensor data [6]. This paradigm capitalizes on the increasing sensing capabilities of mobile devices and the growing interest in mobile sensing services. Mobile devices are flexible and programmable platforms that can efficiently offer customized sensor data as opposed to deploying specialized sensors of limited mobility and functionality. *MetroSence* [7] is a research project at Dartmouth that contains multiple projects aiming at leveraging smartphones to offer advanced personal and public mobile sensing platforms. The project addresses mobile sensing in terms of new applications, classification techniques, privacy approaches, and sensing paradigms. MOSDEN [8] is a distributed mobile sensing framework that enables smartphones to capture and share sensor data. The platform enables collaborative processing of sensor data and provides flexible opportunistic sensing services for applications and users.

Remote health monitoring is another domain that utilizes mobile devices as personal health monitoring platforms. The typical setting is that mobile devices gather vital signs through body sensors, perform pre-processing, and communicate this data to healthcare providers [9]. Mobile devices may also be used as a proactive interface to connect patients with their healthcare providers when certain vital signs exceed pre-specified thresholds [10]. Lomotey and Deters [11] present a cloud-assisted platform that enables mobile devices to provide access to the patient's medical record. The framework implements medical records as web services to facilitate interoperability with heterogeneous healthcare systems. The authors also propose an abstract policy-based access control scheme to maintain authorized access to medical data records and enforce privacy restrictions.

Services that are provided from mobile devices entail computation and energy consumption. Mobile users need incentives that encourage them to participate by either providing mobile services or offering proxy- and data-forwarding services. Mizouni et al. [12] propose a business model for mobile sensing as a service. This model presents a user identification mechanism and charging scheme. However, this model assumes that the mobile network provider takes responsibility for collecting and disseminating sensor data to interested entities. Privacy breaches are another factor that discourages mobile users from providing data services. Li et al. [13] propose two privacy-aware incentive schemes to promote user participation in mobile sensing. These schemes allow users to earn credits for sharing data, while ensuring that dishonest users cannot abuse the system. This credit can be used toward the user's benefit as per personal preferences, such as trading for other services, paying for 3G services, or converted to monetary value. Zhang et al. [14] propose three incentive schemes to motivate users to contribute their resources in participatory sensing services. These schemes are developed based on online reverse auction.

Mobile social network is an emerging domain that utilizes personal services to improve the user experience. This kind of social communication is attractive due to the ability to link people in certain locations together. For example, JAMPS [15] is a social network platform that allows users to share multimedia resources and messages on their mobile devices when they are in close proximity. The platform stores shared contents on mobile devices and enables access through software agents. However, users must maintain their registration in a zone-specific container that manages active software agents. CloudMoV [16] is a mobile social TV system that enables users to interact while sharing videos. Each user is represented by a surrogate on the cloud to synchronize and maintain the view progress of the shared video. It also stores the user's friend list. Communication between users are routed through surrogates. Greer and Ngo [17] developed a custom Facebook App to help people prepare for emergencies and receive support from family members and friends during disasters and aftermath. The application enables users to use their mobile devices to provide their location information to first responders and communicate with friends during emergency situations and disaster relief efforts.

The use of mobile devices to offer personal services is on the rise, supported by growing public interest. However, existing architectures for these services are domain specific, providing limited or no access control to personal data. In addition, such systems lack interoperability between heterogeneous platforms and possibly need further efforts to integrate with similar systems or to be deployed on different mobile platforms. Furthermore, none of the previous proposals address how personal services are published so that interested users are aware of them and know how to submit access requests. In contrast, we present a reference architecture for developing personal services that offer global or personal information access through mobile devices. We also present efficient publishing and discovery mechanisms that are aware of the resource limitations of mobile devices, yet announce and discover personal services in near-real time.

3 Personal services

3.1 Definition

Personal 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.

Formally, a personal 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, $L = \{c_1, c_2, \ldots, c_n\}$, n is the length of L, c_i is one of the user's contacts, and $1 \leq i \leq n$. Each Web resource $m_j \in M$ has a set of access constraints A_{m_j} 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 the appropriate access privileges to a Web resource m_j if $R_{c_i} \overset{satisfy}{\rightarrow} A_{m_j}$.

3.2 Distinguishing characteristics

Personal services are direct descendants of mobile Web services [18], and as such, they share both the advantages and constraints outlined in [19,20]. 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 resource constraints of mobile providers are less likely to impact the quality of service provisioning.
- The personal service paradigm places the service provider at the core of the service communications. Thus, the provider P manages access privileges of customers according to how their credentials R_{c_i} satisfy the access constraints A_{m_j} of a resource m_j.
- The aim of personal services was to enable access to personal data. Such services are offered at a best EFFORT level of service with no guarantee of continuous availability. As such, providers and consumers may communicate with each other whenever a reliable network connection is available. This avoids the major challenge stemming from the intermittent connectivity that characterizes wireless communications.
- The personal service paradigm takes advantage of the provider's contact list L to disseminate service announcements and requests with minimal resource consumption at the provider's side. Mobile users cooperate, using their own resources, to extend the reachability of service providers to advertise their services and expand the horizon for consumers to find better matches.

3.3 Personal service architecture

In this section, we present the architecture for personal services, which is shown in Fig. 1. Personal services are deployed on mobile devices, where an embedded lightweight Web server exists to provide the essential functionalities of HTTP-based service communications. The mobile device

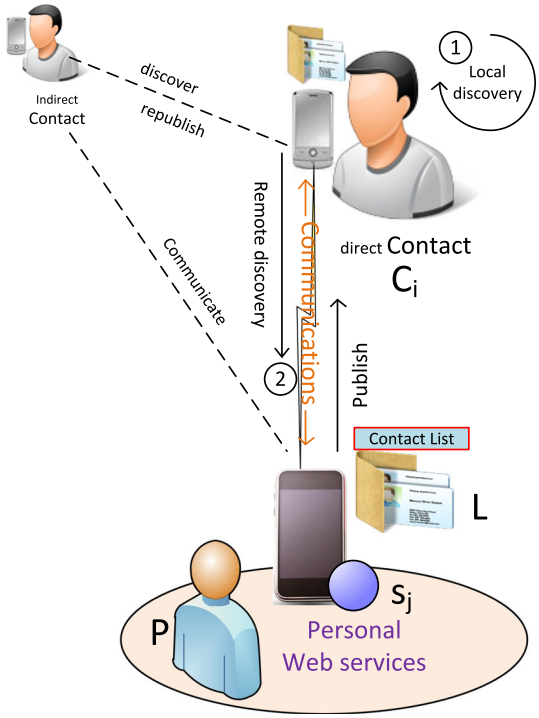

Fig. 1 The architecture of personal services

user represents the service provider, and service consumers are direct or indirect contacts of the provider. The service provider advertises personal services to the members of his/her contact list. The service advertisement determines whether recipients are allowed to forward it further to their local contact list. Similarly, the personal service discovery procedure sends discovery requests to the members of the requester's contact list, after checking its own local directory. The discovery procedure looks up the required service(s) in the recipient's local service directory first, if such a directory exists, (step 1). If no match is found, the discovery procedure forwards the request to people in the recipient's contact list (step 2), delegating the discovery task to them. Once a match is found, interactions between the provider and the consumer are performed over a direct link.

While the personal service architecture is optimally designed for mobile devices, there is no barrier preventing non-mobile based contacts from participating. The contact list concept is generic and it may integrate contacts from multiple sources (e.g., phonebook address, social circles, mailing lists), through which the provider can reach a wide range of customers.

3.4 Service directory

The standard Web service architecture uses a registry like UDDI [21] to support service publication and discovery. The personal service paradigm adopts a distributed service direc-

tory approach, where each mobile device manages its own offered services and maintains references to services it knows about. 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. Maintaining a service directory is optional for consumers or participants who collaborate to forward service announcements or discovery requests. However, the more service directories exist in an overlay network, the less time a discovery request may take to find matching services.

Listing 1 The structure of the personal service Metadata profile f_s

```
<?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>
  <TTL>time</TTL>
</WebService>
```

The service directory records Metadata about services, which provides a summary description of the service functionality and how consumers can reach the service resources. Listing 1 shows the XML structure of the personal service Metadata profile f_s. The *sID* represents a unique service ID, the *publicationDepth* determines how far the service announcement can reach, the *title* is a string that holds the service name and typically projects its core functionality, the *description* is a plain text that explains the offered functionality, and *endPoint* is the service base URI. The *title* and *description* attributes are used by the discovery process to match a user objective. Therefore, the name and description 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 the following generic format: http://www.device-URI/service-root/metaProfile. Although some broadband network providers offer a fixed IP address for mobile devices though which the device can be addressed, we remark that Internet addressing is beyond the scope of this thesis. Nevertheless, there is a number of research efforts on how to get a personal Uniform Resource Identifier (URI) for mobile devices, viz. RFC6116 [22], which proposes a translation of a telephone number into a URI using a special DNS record type.

Table 1 shows the table structure for the personal service directory. The *status* determines whether a service is up and running or temporarily suspended. The *contactEndPoint* attribute identifies the base Internet address at which the service provider can be reached. This attribute also sets the URI address of two APIs: contactEndpoint/registry/publish and contactEndPoint/registry/discover to register and look up services, respectively. Table 2 shows a full list of interfaces that the service directory exposes to handle the basic directory operations.

Each personal service announcement is associated with a Time To Live (TTL) parameter. The TTL determines the age of the announcement. Personal services must be re-announced before their respective TTL expires to remain valid. Providers define the TTL for each service. The service directory removes services whose TTL is expired. The personal service paradigm uses the TTL concept to control the validity of the service announcement and maintain the consistency of distributed service directories. Personal service providers may temporarily suspend their services that are likely to be re-offered in future.

To avoid temporal inconsistency of service directories, nodes that rejoin from a disconnected state or switch ON from an OFF state must query service providers whether their services are still valid using the *"status"* method offered by the directory manager. These nodes then must update their local directory and contacts accordingly. If service providers do not respond to the status query, their hosted services are marked *inactive* until a new re-announcement is received.

Table 1 The structure of 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
TTL	int	Time to live
type	int	0 = local, 1 = remote
status	int	0 = active (default), 1 = inactive

Table 2 The essential service directory functionalities

Functionality	Return	Purpose
add(contactID, sID, title, description, endPoint)	int	Adds a new 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

3.5 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 [23]. The standard Web service discovery approach has not been widely adopted [24] due to the limitations and lack of robustness of the UDDI [23]. Therefore, providers usually resort to other methods to publicize their services, for example, on their own websites.

The personal service paradigm enables providers to selectively advertise their services based on preauthorization. Figure 1 illustrates the publication procedure using contact lists. Once a personal service is deployed on the mobile device, the publication procedure registers it with the local directory and sends a message containing the Metadata profile of the service to members of the contact list, if applicable. The provider controls the parameter d, which represents the publication depth for a particular service. If d is set to "0", the service is only registered with the local directory. If $d > 0$, the recipient of the announcement is allowed to forward it to others. The depth controls how far a service announcement can reach. Recipients of service announcements discard the message if the service already exists in the local directory, to avoid duplicate announcements coming from multiple passes (contacts), and add a new entry if the service is new. Service providers are not allowed to republish an existing service before its TTL expires to avoid possible denial of service (DoS) attacks.

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 $L(P)$ or a subset of it $l \subset L(P)$. A contact c_i receives the service advertisement, reduces d by 1 and republishes the service to its local contact list L_{c_i}, excluding the direct source of the message to avoid open loops that can occur if the source and the delegate are reciprocal contacts. The publication stops when d reaches "0".

Algorithm 1: Personal service publication.

```
   Input: service Metadata profile f_s
   Output: Boolean (0/1)
 1 Function Publish(f_s)
 2 Parse f_s
 3 Set the provider P
 4 Set the publication depth d
   // check if service exists in the local
      directory
 5 if s exists in sdir then
       // discard duplicate announcement
 6     return 0
 7 end
 8 else
 9     add s
       // check if forwarding is allowed
10     if d > 0 then
11         d = d − 1
12         update d in f_s
           /* set L to local contact list,
              excluding publishing peer        */
13         source ← publishing_peer
14         L = L_local / source // or subset l ⊂ L
           // delegate publication to contacts
15         foreach contact c_i in L do
               // distributed recursive call
16             Call Publish(f_s) // at the contact side
17         end
18     end
19     return 1
20 end
```

3.6 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 [25] to identify the required functionalities (RF) from a user request (SR) and to extract service capabilities (SC) from the service description field $description$ in the service Metadata profile f_s. The personal service paradigm uses keyword-based matching to reduce the resource consumption of mobile

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
   // search local service directory first
3  foreach s in sdir do
4  |  extract capabilities SC from descriptions
5  |  rank=match(RF,SC)
6  |  add s to RelS indexed by rank
7  end
8  if RelS is null then
   |  // check if deep search is allowed
9  |  if d > 0 then
10 |  |  d = d − 1
   |  |  // delegate discovery to contacts
11 |  |  source ← searching_peer
12 |  |  L = L_local / source
13 |  |  foreach contact c_i in L do
   |  |  |  // distributed recursive call
14 |  |  |  Call Search(SR, d) // at the contact side
15 |  |  end
16 |  end
17 end
18 return RelS
```

devices. Although semantic discovery approaches yield better results, they are computationally intensive and negatively impact the performance of resource-constrained environments. Generally, according to the potential applications of personal services and their associated lifetime parameter, the number of expected active services in one's local directory is not expected to be large. This means that a simple discovery process that incurs low burden on local resources is effective. However, in cases where the number of active services in the local directory is large, offline clustering techniques [25] may be used to limit the search in only a subset of services that is relevant to the request.

The discovery procedure applies Algorithm 2 to find services relevant to a discovery request by matching the required functionalities with capabilities offered by personal services. The discovery algorithm is distributed and recursive, where each node initiates its own instance on local resources once it receives a discovery request. Relevant services ($RelS$) are collected at the request originator and ranked according to their similarity factor.

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

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

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.

4 Service access control

Personal service providers need to control access to their offered services. Access management schemes allow providers to selectively assign access privileges to services based on user credentials and provider-defined access policies. Therefore, interacting parties need to establish a certain level of trust before serious interactions occur. Trust is considered a primary security mechanism to make access decisions for digital contents [26,27]. During trust establishment, providers may request to exchange information with the service requester to make informed access decisions. However, providers offering personal services need to take extra precautions when granting access to their sensitive personal data due to the fact that such data could be cached for future reuse without the owner's consent. The personal service paradigm builds its access control mechanism based on the centralized access control scheme proposed by Machulak et al. [28]. Although a sophisticated access control mechanism is beyond the scope of this research, our proposed approach provides the required fundamental functionality, aiming to ensure that only users with the proper access privileges can access service functionalities. However, our current access control mechanism is not intended to limit service publication or discovery to only authorized users.

Figure 2 illustrates the architecture of the proposed access control mechanism. Four entities interact together to establish control over access: a service provider (P), a mobile device (H), an authentication manager (AM), and a user(U). The user represents the service requester who is interested in the offered services. The user could be a person, or an application acting on its behalf. The service provider represents the service owner who sets and administers access policies. The

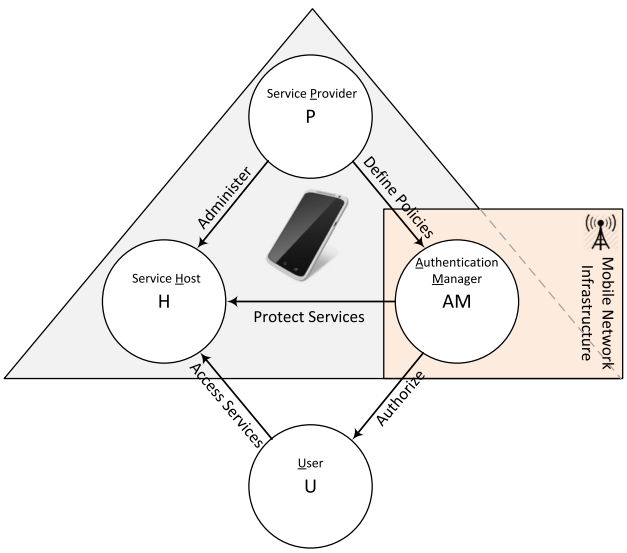

Fig. 2 The architecture of the proposed access control mechanism

mobile device offers the hosting environment to protected services and enforces access control decisions. The authentication manager manages the access control and acts on behalf of the service provider. It evaluates the incoming requests against relevant access policies and makes access decisions whether to grant or deny access to protected resources.

The AM could be provided by a third party (e.g., mobile network providers) or hosted on the platform of the personal service provider, depending on the type of service, how vulnerable it is to security attacks, and how capable is the personal service provider to support security mechanisms. Despite that there are lightweight security techniques that can be applied on resource-constrained platforms, a trade-off between performance and required level of security must be made. Our design suggests that the AM component is better to be deployed on the network provider's infrastructure for the following reasons: (1) mobile network providers have their own established security provisioning system; (2) communications with the personal service provider go through the mobile network infrastructure; hence, the network provider is a natural candidate to offer subscribers efficient security services; (3) the computing infrastructure of the mobile network provider is capable of offering better performance and deploying advanced security techniques that can detect and protect subscribers from malicious attacks.

4.1 Access control functionality

The access control scheme provides necessary functionality to protect personal services from unauthorized access. To gain access to protected services, the user needs to associate the request with a valid access token. A request without a valid access token results in a response with an unauthorized access message. This response includes Meta information about the AM such as its URL where the user may seek a valid access token to the service of interest. In what follows, we describe the basic functionalities offered by an AM.

4.1.1 Defining access control policies

The service providers define access policies for their offered resources. Providers may use simple rules or sophisticated policy structures to govern access to their online resources. Policies could be generic or tightly coupled with services [29]. Policies may require the user to reveal identity or verify the possession of certain attributes. For example, physicians who wish to access the health record of their patients either submit their registration ID or are able to verify that they are the patient's healthcare provider. An access policy encompasses a set of rules and claims. A rule represents a category of users and their associated access privileges, where a claim represents an attribute or a property that an authorized user must satisfy. The provider links services with related access policies.

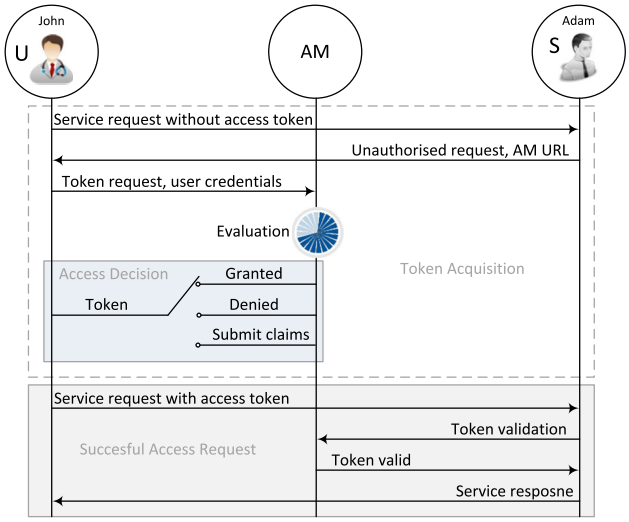

Fig. 3 Typical interactions of a successful service request

4.1.2 Requesting access tokens

Figure 3 illustrates the token acquisition procedure. Users send access requests to personal services. Each request must be associated with a relevant access token to the requested service. If this token is missing or invalid, the service host dispatches an access denial message with a URL referring to the AM. The user communicates with the AM to obtain a valid access token. The user registers with the AM and provides the necessary information or credentials. The AM evaluates the user's credentials against access polices and, if appropriate, provides the user with a valid access token. This token could be valid for only the current request, or for a certain period of time t, defined by the service provider per service session. The AM may require the user to submit further claims before access is granted, such as adhering to certain terms and conditions. The AM may request the consent of the user to collect and verify user credentials in real time.

4.1.3 Making access decisions

The AM makes access decisions by asserting the user's credentials against access policies of the requested service. It performs such evaluation using a simple access control matching matrix or a policy engine that supports advanced policy handling and composition mechanisms. According to the matching results, the AM may permit access, deny access, or verify users.

4.1.4 Enforcing access decisions

Users present access tokens when submitting service requests. The service host (mobile device) verifies with the

Table 3 Personal service exposing a patient's eHealth record

Web Resource	Description	Access Policies
`/serviceRoot/vitalSigns`	Fresh reading of vital signs (ECG, SPO2, etc.)	(1) Dr. Barber, (2) Members of Queen's Family Health Team, (3) Paramedic
`/serviceRoot/imagingReport/[list]`	Display imaging report by selecting form available list	(1), (2), (4) General Hospital, (5) Imaging Center, (6){Student of Queen's Medical School+ Claim_1 (I understand that this information is confidential and made available for educational purposes. I hereby undertake not to disclose all or part of it to any unauthorized person.)}
`/serviceRoot/summarySatus`	A summary status about the current health issues	(1), (2), (7) Wife, (8) Family memeber
`/serviceRoot/onMedications`	List of current medications	(1), (2), (3), (7)
`/serviceRoot/prescriptions/[list]`	Display prescription items, show a list of available prescriptions on a chronological order	(1), (7), (9) myPharmacy

AM whether or not the access token is valid. The request is granted or denied access based on the validation result.

4.2 Scenario

This scenario elaborates on the possible interactions between different entities during a service access request. Adam (provider) maintains his own eHealth record on his mobile device (host). Adam offers various Web service methods that provide access to his eHealth record with various details as shown in Table 3. Adam protects such personal information using an authentication manager running on his mobile device. John (user) is a resident family physician working with the Queen's Family Health Team, which provides healthcare services to Adam. John sends a request to access the latest status of Adam's vital signs. John's request is missing the appropriate access token, resulting in the following access denial message: *Access denied, please consult the AM at* http://www.adam-phone-domain-or-ip/am. John presents to the AM a password with which the AM is able to verify that John belongs to the Queen's Family Health Team. The AM evaluates John's credentials against the relevant access policies and provides a valid access token. John uses this token to resubmit the unsuccessful service request. Adam's mobile device verifies the request with the AM and responds with Adam's current vital signs. Interactions between the various entities communicating in this scenario are shown in Fig. 3.

5 SLiM: Smart contact List Management

We demonstrate the feasibility and usefulness of personal services by implementing a SLiM service, a functional prototype that automates the management of personal contact lists. This prototype shows mutual benefits for 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.

Figure 4 illustrates the architecture of SLiM. For illustrative purposes, we show a provider, Adam, who maintains his own profile and contact information on his smartphone. Adam grants his contacts access to his profile and allows them to update their records with his current information. If Adam's friends offer a similar service, Adam also can update his contact list. Adam authorizes John to search his contact list for contact information of common friends.

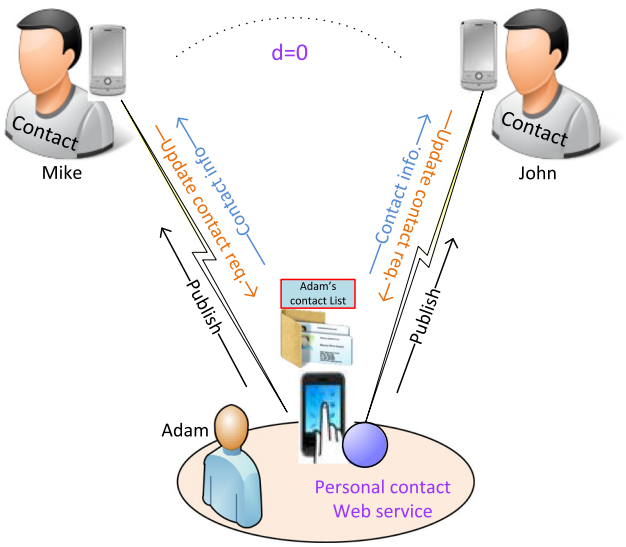

Fig. 4 A high level overview of the SLiM scenario

Table 4 Methods exposed by the personal service

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

5.1 Contact list

Each mobile device user maintains a contact list of friends, family members, business partners, etc. A typical contact record has two main categories of information: contact-related information and user-related information. The contact-related 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 user-related category describes the contact from the perspective of the mobile user based on personal preferences, such as the group to which this person belongs, the user-assigned ringtone for this person. Both categories are currently entered and maintained manually by phone users.

SLiM aims to automate the maintenance of the contact-related information while allowing the contact person (to whom this information belongs) full control of granting access privileges, 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. Such a feature renders automatic contact list management even more exciting.

5.2 Implementation details

A personal service, `contactInfo`, is developed using the Python programming language in compliance with the "RESTful" principals to provide the core functionalities of SLiM. 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 [30] is used to handle the low-level details of Web service development such as protocols, sockets, and process management.

The `contactInfo` is deployed on Adam's Samsung Galaxy II I9100 smartphone (Dual-core 1.2 GHz Cortex-A9, 1 GB RAM). The phone runs a rooted Android 4.0.4 platform [31] and is connected to a WiFi network. Table 4 shows the basic methods and functionalities exposed by `contactInfo`, pertaining to Adam's contact details. Each method represents a Web resource that can be accessed through a unique identifier (URI) by authorized users. The generic format of the resource identifier is http://www.root-address/service-name/resource-name/[parameters].
The `root-address` is Adam's phone IP address or domain name (*contactEndPoint*), `service-name` is the given name of the personal service, and the `resource-name` refers to a specific service resource/functionality.

The current version of SLiM focuses on personal service provisioning, excluding discovery aspects. However, the search functionality that SLiM presents is very similar to the process of personal service discovery. SLiM uses the contact list attribute `Internet call`, which is an existing entry in Android's contact information dedicated to hold the contact SIP address for Internet call, to hold the `contactEndPoint`. We envisage that the `contactEndPoint` could be added as a dedicated entry to the contact information in mainstream releases of future mobile platforms.

SLiM sets the publication depth d to "1" to limit the service announcement to the members of Adam's contact list. The publication procedure registers the service with Adam's local directory through the API http://www.172.1.6.36:8080/registry/publish and informs all Adam's contacts of the existence of `contactInfo`.

In our implementation, the Web service offers different representations of the same service response, namely, *XML*, *JSON* and *HTML* using `mimerender` [32], 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 directory is implemented using SQLite [33], which is the Android default Database engine. The built-in package `android.database` handles general database operations, while *android.database.sqlite* contains classes specific to SQLite [33]. A user interface (UI) for SLiM

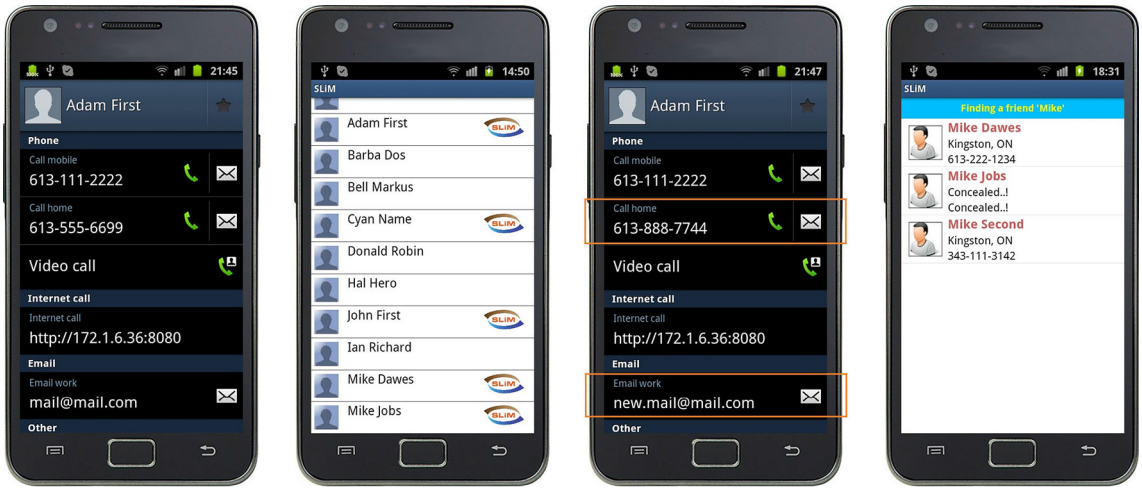

Fig. 5 SLiM test case scenarios

is developed on an Android platform using the Android SDK [34]. Members of the contact list are SliM contacts if their `contactEndPoint` attribute is active (i.e., set with a value), whether the contact is offering a personal service or just taking advantage of receiving automatic updates from contacts about changes in their information.

Figure 5b shows a screenshot of the UI, in which contacts with a SLiM icon (to their right) are active participants. The Android contacts API `android.provider.Contacts Contract` [35] 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 whether 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.

5.3 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. Adam sets access policies so that family members are granted full access, while business partners can access only public information, such as email address. John, a caregiver, opens up his contact list and taps the SLiM icon right after Adam's name. Adam's contact information is automatically updated on John's end, given that John fulfills access requirements or has been preauthorized by Adam, as shown in Fig. 5c.

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 for friends in common. John is looking for Mike's contact information. John sends Adam a Web service request to look up Mike's contact information. The service request has the form http://www.172.1.6.36:8080/contactInfo/search/Mike, 1, where `search` is the method name, `Mike` is the search term, and "`1`" is the search depth (d). The depth being set to 1 means that John does not authorize Adam to forward this request any further. Utilizing the comma-separated approach in passing parameters to the HTTP request is merely an implementation issue.

Figure 5d 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 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.

6 Performance analysis

This section studies the performance of personal services in general. The publication and discovery algorithms are implemented in Python. We run these experiments on two mobile devices whose configurations are mentioned earlier, one represents Adam and the other represents John. The mobile system consumes a power unit $p_c = 0.9$ W/s for computing and $p_t = 1.3$ W/s for sending or receiving data over a wireless

WiFi link. At each device, we generated a pool of contacts containing 500 records. Each contact is a structure record that contains basic contact information, such as `name`, `phone`, `email`, and most importantly `contactEndPoint`, by which the contact can be reached. We set the contactEndPoint parameter to point to a unique IP address; however, two records register the real IP addresses of the two test devices. This is set out to study the impact of the depth d on publication and discovery procedures. When a mobile device receives a service announcement or a discovery request, it picks up a random set of contacts ranging from 50 to 500 contacts to create an ad hoc random contact list. This setup means that whenever $d > 0$, the two devices keeps forwarding the current request to each other until $d = 0$. Each device maintains a local service directory containing 50 service entries. The two devices communicate over a WiFi link with a Round Trip Time (RTT) = 17 ms and a data bandwidth $B = 12.4$ MB/S. We run each experiment 10 times and take the average readings of execution time, overall response time, CPU usage, and length of contact lists. We remark that confidence intervals are found to be negligible and hence are not reported.

6.1 Single-level publication

This experiment investigates the time that the publication procedure takes to announce a service to local resources. The provider first registers the service with the local directory, which takes on average 225 ,ms to insert a new record into the directory database. Then, the provider traverses through the contact list and sends a message with the service(s) announcement to each participating contact. This message contains the service Metadata profile with an average length of 1.5 KB per service. Although we announce a single service, the message may contain multiple service announcements, say X, which would increase the size by X multiples. Figure 6 shows the execution time of the publication procedure on local mobile resources. The publication execution time is dependent on the length of the contact list. The more contacts in the list, the more time the publication procedure takes to finish. However, such a time is <1.5 s for a list of 50 contacts. The procedure does not verify whether the contact successfully received the announcement, which is left to the underlying transport protocol. This explains why the publication local execution time is independent of the network latency between communicating parties.

6.2 Multi level publication

This experiment evaluates the total execution time of multi level service publication, beginning when a provider originates a service advertisement until all contacts at a certain d level are informed. We measure the elapsed publication time versus a varying depth $0 \leq d \leq 10$. Whenever a device receives a service advertisement, it picks up a random set of contacts ranging from 50 to 500 contacts to create an ad hoc random contact list, ensuring that at least one contact referring to the other device is picked up. This setup means that a device services a single request at a time. The mobile device only registers the service into the local directory when $d = 0$. When $d > 0$, it also forwards the announcement to all participating contacts. Our setup of contacts makes the two devices alternate searching at each level further of d and processes only a single request at each round. To calculate the number of informed contacts, we add up all contacts in the contact tree whose root begins at the source of the service announcement, assuming that contacts within the same round (or d level) disseminate the announcement in parallel using their own resources.

Figure 7 time, the earliest advertisement time, and the number of informed contacts versus a varying number of depth levels d. The elapsed time represents the overall accumulated time until all contacts in a respective level are informed. The earliest advertisement time represents the ear-

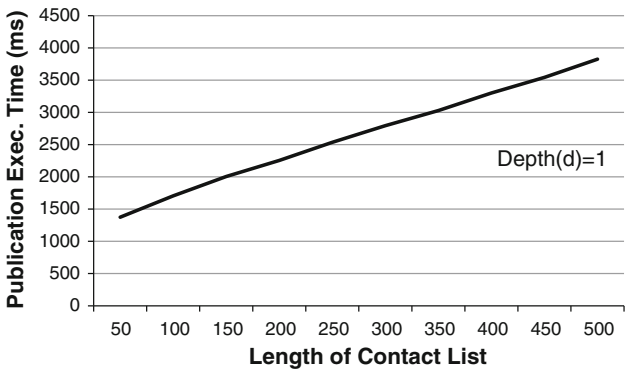

Fig. 6 The execution time of the publication procedure on local resources

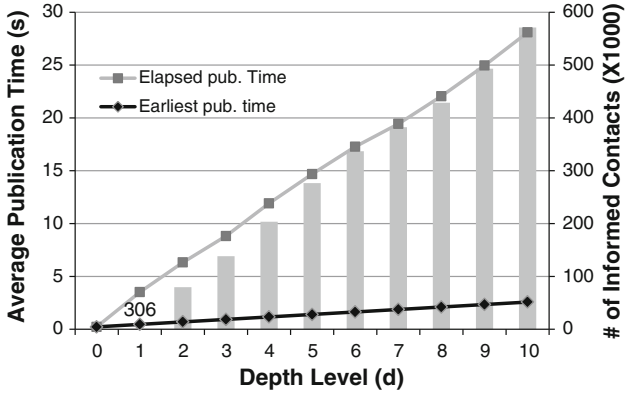

Fig. 7 Total publication time versus a varying number of depth levels d

Table 5 Detailed results of personal service publication procedure

Deth (d)	0	1	2	3	4	5	6	7	8	9	10
Elap. Pub. time	0.23	3.51	6.325	8.82	11.915	14.69	17.275	19.43	22.055	24.96	28.095
Earliest Ad. time	0.23	0.46	0.695	0.93	1.165	1.4	1.635	1.87	2.105	2.34	2.575
Pub. time/level	0.19	3.29	2.815	2.495	3.095	2.775	2.585	2.155	2.625	2.905	3.135
Length of contact list	1	306	259	227	287	255	236	193	240	268	291
# of informed contacts	1	307	79,561	138,354	203,503	276,688	336,868	382,416	428,736	493,056	571,044

Table 6 Detailed results of investigating the total discovery time

Deth (d)	0	1	2	3	4	5	6	7	8	9	10
Elap. Disc. time	2.5	5.91	10.68	15.8	20.8	25.59	30.83	35.94	41.44	46.65	51.72
Min. Resp. time	2.5	3.06	5.62	8.18	10.74	13.3	15.86	18.42	20.98	23.54	26.1
Exec. time/level	2.5	5.41	4.77	5.12	5	4.79	5.24	5.11	5.5	5.21	5.07
Length of contact list	1	286	222	257	245	224	269	256	295	266	252
# of inspected contacts	1	287	63,779	120,833	183,798	238,678	298,934	367,798	443,318	521,788	588,820

liest time at which a contact from the respective level can be informed. The total number of informed contacts indicates the aggregated number of contacts that are informed by the end of each level. A mobile device needs around 0.225 s to register an announcement into the local directory. In this experiment, we send a separate announcement message for each contact. It takes 3.51 s to serve a contact list of 300 members on average. It also takes around 28 s to inform more than half million participants in 10 levels. Despite the fact that this number is not strictly accurate in practice, as we assume a fixed length of contact list at each level and some contacts might be duplicate, the numbers give a deep insight about the significant number a request can reach as the depth progresses further. Table 5 provides more details about this experiment.

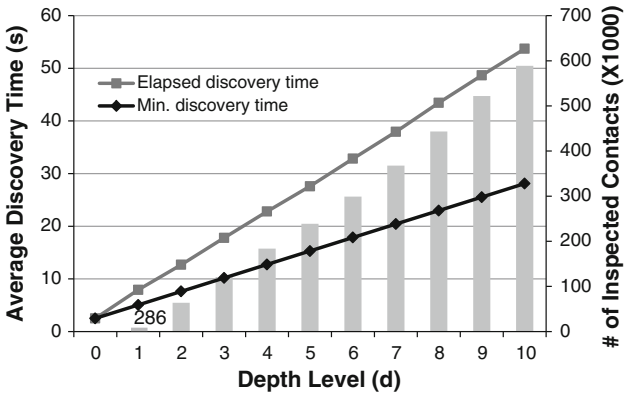

Fig. 8 Total discovery time versus a varying number of depth levels d

6.3 Multi level discovery

This experiment is similar to the previous one except that it investigates the total discovery time and how many contacts are searched with each level of depth allowed by d. The setup is also the same. However, the random generation of contact lists yields lists of different lengths. The mobile device searches only the local directory when $d = 0$ and forwards the discovery request when $d > 0$. At each level, participants search in parallel their local directories. The service matchmaking algorithm is keyword-based, matching the keywords of the request with the keywords of service description.

Figure 8 shows the elapsed discovery time, the minimum response time, and the number of inspected contacts versus a varying number of depth levels d. The elapsed time represents the overall accumulated time until all contacts in a respective level are inspected. The minimum response time represents the minimum time at which a response can be received at each level, i.e., a match found at the first contacted participant. The total number of inspected contacts indicates the aggregated number of searched contacts by the end of each level. The relative difference between the elapsed time and the minimum response time represents the time window in which a response can be received at each level. We observe at the zero level ($d = 0$), where the requester searches only the local directory of 50 entries in our case, the overall discovery time is almost 2.5 s. Whereas at the first level ($d = 1$), the requester investigates 286 contacts, each searches its own local directory, within around 8 s, noting that the request travels from the requester to each contact in 17 ms, on average. We also observe that the number of inspected contacts increases exponentially with each additional level. For example, at level 5, the total number of searched contacts exceeds 183,000. Table 6 shows the fine grain details of the results of this experiment.

6.4 Overhead on local resources

This experiment investigates the overhead that the personal service functionalities incur on mobile nodes, whether forwarding or processing requests. We record the CPU usage and the overall execution time for both publication and discovery, under a varying stress load, ranging from 1 to 200 simultaneous requests. We calculate the energy consumption as a function of the execution time and data transfer as the following:

$$E = T \times p_c + \frac{D}{BW} \times p_t \quad (2)$$

where E is the total energy consumption of a mobile node, T is the total CPU time of the node, p_c is the power consumption unit of computation, p_t is the power consumption unit of data transfer, D is the total amount of data transfer, and BW is the link bandwidth. In this experiment, we change the contactEndPoint attribute for all contacts in our pool to point to one of the two devices. Each device picks up contacts with a contactEndPoint referring to the other device. This setup creates a synthesized load of concurrent requests, in this case originating from the same source. Therefore, each device receives multiple concurrent requests, whether for discovery or advertisements.

Figure 9 shows the overall execution time for discovery and publication requests versus a varying number of concurrent requests. We observe that publication is typically faster than discovery. This difference is due to the matchmaking process that each node performs against its local service directory to fulfill the discovery request. This process in itself is computational-intensive, especially if the search corpus is large. We also observe that as the load increases, the rate at which the discovery time increases is greater than the rate of increase in publication time. The figure shows that the time to handle 200 concurrent advertisement requests is almost 3.8 times the time to process only one request, whereas it takes around 4.7 times in a similar discovery scenario. It is worth spending this time, if necessary, to communicate request to more than 600,000 nodes.

Figure 10 shows the percentage of CPU usage versus a varying load. The CPU usage in general is lower in publication in contrast to similar discovery scenarios. The extra overhead again is attributed to searching the local directory for services that match the current discovery request. The experimental results reveal that a mobile device may dedicate almost 40 % and 17 % of its CPU capacity to handle 200 concurrent discovery and advertisements requests, respectively. While 17 % CPU usage could be tolerable in the case of publication, on a non-continuous basis, we believe that mobile users may not be able to afford offering half of their computational resources to process and forward discovery requests on behalf of others.

Figure 11 illustrates the energy consumption profile of the publication and discovery procedures of personal services. The results reveal a major challenge for devices in meeting the energy needs of this processing. Although the energy consumption of serving discovery requests is higher

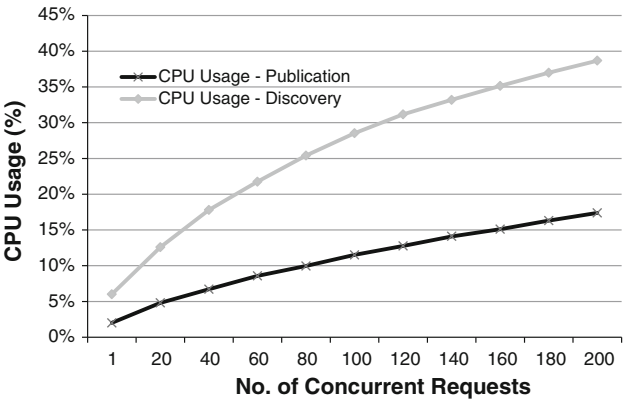

Fig. 10 CPU usage versus a varying load

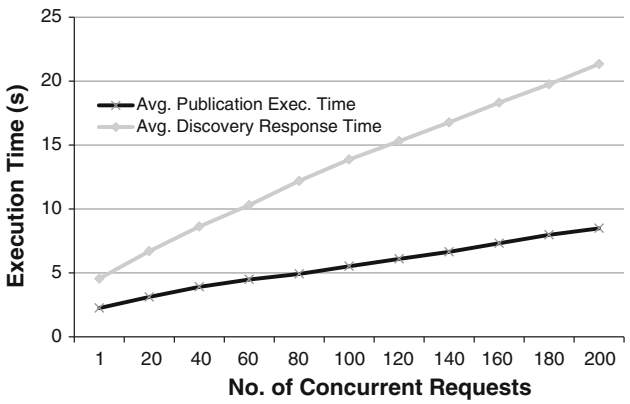

Fig. 9 Execution time for discovery and publication requests versus a varying number of concurrent requests

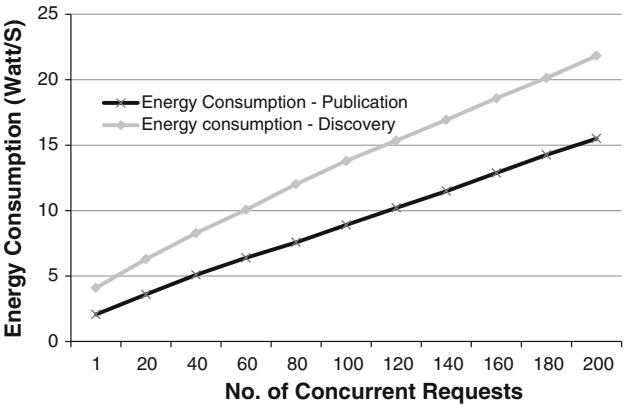

Fig. 11 Energy consumption profile of the publication and discovery procedures

than the requirements of similar publication requests, both energy requirements are significantly high. This might drain the mobile device battery very quickly, which may cause a negative impact on other functionality. The core problem stems from sending multiple messages with the same content to a group of recipients on an individual basis. This means that each contact receives a unicast message with the respective request. This problem could be approached by sending a single message to a group of different recipients using multicasting techniques. However, this is a direction for future research.

According to the overhead that personal services may place on mobile nodes, users may configure certain thresholds and constraints, based on which the system can make participation decisions. For example, when resources are low, users could opt-out from participation, or when the consumption of certain resource exceeds a certain threshold, all upcoming requests are rejected. We also anticipate that there would be some sort of incentives or business models that encourage users to participate in such collaborative computing paradigms. For example, users can be credited for their participation with a point-collection system, managed by the network provider. The more the user participates, the more the user can collect points. The user may redeem these points at any time to pay for his/her service or cash them in a monetary value.

7 Conclusion

This paper introduces *personal services*, a user-centric Web service architecture hosted on mobile devices. The personal services architecture takes advantage of the provider's contact list to announce service existence and discovery requests. 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.

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, but it provides some insights into the major functionality. The opportunities made possible by the development of such a prototype, and validation scenarios, have made mobile devices an increasingly attractive platform for everyday life tasks. We carried out extensive performance evaluation for the main activities of the personal service paradigm as well as evaluated the overhead it incurs on local resources of participating nodes. Experimental results show that requests can reach a huge number of contacts within a relatively short time. Experiments also show that the energy requirements of such a paradigm could be discouraging for users with low battery level.

We believe that the personal service paradigm lays the foundation of a new service architecture for mobile devices that will expand the horizon of mobile applications and their domains. It enables users to share personal information, while maintaining full control over privacy. This paradigm will reshape the way many applications, such as social networking and public participatory sensing, are currently offered.

We plan to extend the architecture to enable providers to advertise services that are bound to a specific physical location and consumers to discover services that are offered within their close proximity.

References

1. Kirkham T, Winfield S, Ravet S, Kellomaki S (2011) A personal data store for an internet of subjects. In: The international conference on information society (i-Society), pp 92–97
2. Elgazzar K (2013) Discovery, personalization and resource provisioning of mobile services. PhD thesis, School of Computing, Queen's University, Kingston, ON K7L 2N8, August 2013
3. O'Sullivan M, Grigoras D (2013) The cloud personal assistant for providing services to mobile clients. In: The IEEE 7th international symposium on service oriented system engineering (SOSE), pp 478–485
4. Ibrohimovna M, de Groot S (2009) Policy-based hybrid approach to service provisioning in federations of personal networks. In: Third international conference on mobile ubiquitous computing, systems, services and technologies, pp 311–317
5. Augusto, AB, Correia M (2012) Ofelia a secure mobile attribute aggregation infrastructure for user-centric identity management. In: Proceedings of the IFIP international information security and privacy conference, pp 61–74
6. Campbell A, Eisenman S, Lane N, Miluzzo E, Peterson R, Lu H, Zheng X, Musolesi M, Fodor K, Ahn G-S (July 2008) The rise of people-centric sensing. Internet Comput IEEE 12:12–21
7. MetroSence (2014) http://www.metrosense.cs.dartmouth.edu/. Accessed 1 Mar 2014
8. Jayaraman P, Perera C, Georgakopoulos D, Zaslavsky A (2013) Efficient opportunistic sensing using mobile collaborative platform mosden. In: The 9th international conference on collaborative computing: networking, applications and worksharing (Collaboratecom), pp 77–86
9. Kozlovszky M, Bartalis L, Jokai B, Ferenczi J, Bogdanov P, MeixnerZ, Nemeth L, Karoczkai K (2013) Personal health monitoring with android based mobile devices. In: The 36th international convention on information communication technology electronics microelectronics (MIPRO), pp 326–330
10. Elgazzar K, Aboelfotoh M, Martin P, Hassanein HS (2012) Ubiquitous health monitoring using mobile web services. In: The 3rd international conference on ambient systems, networks and technologies, August 2012
11. Lomotey, R, Deters R (2013) Using a cloud-centric middleware to enable mobile hosting of web services: mhealth use case. In: Personal and ubiquitous computing, pp 1–14
12. Mizouni, R, El Barachi M (2013) Mobile phone sensing as a service: business model and use cases. In: The seventh international

conference on next generation mobile apps, services and technologies (NGMAST), pp 116–121
13. Li Q, Cao G (2013) Providing privacy-aware incentives for mobile sensing. In: 2013 IEEE international conference on pervasive computing and communications (PerCom), pp 76–84
14. Zhang X, Yang Z, Zhou Z, Cai H, Chen L, Li X (2014) Free market of crowdsourcing: incentive mechanism design for mobile sensing. IEEE Trans Parallel Distrib Syst 99:1–11
15. Fortino G, Palau C (2012) An agent-based mobile social network. In: The international conference on multimedia computing and systems (ICMCS), pp 961–967
16. Wu Y, Zhang Z, Wu C, Li Z, Lau F (June 2013) Cloudmov: cloud-based mobile social tv. IEEE Trans Multimed 15:821–832
17. Greer M, Ngo J (2012) Personal emergency preparedness plan (pepp) facebook app: using cloud computing, mobile technology, and social networking services to decompress traditional channels of communication during emergencies and disasters. In: The IEEE ninth international conference on services computing (SCC), pp 494–498
18. Srirama SN, Jarke M, Prinz W (2006) Mobile web service provisioning. In: The international conference on internet and web applications and services, pp 120–126
19. Elgazzar K, Martin P, Hassanein H (2011) A framework for efficient web services provisioning in mobile environments. In: The 3rd international conference on mobile computing, applications, and services, Springer's LNICST, October 2011
20. Mizouni R, Serhani M, Dssouli R, Benharref A, Taleb I (2011) Performance evaluation of mobile web services. In: The 9th IEEE European conference on web services, pp 184–191
21. Clement L, Hately A, von Riegen C, Rogers T (2004) Uddi version 3.0.2, January 19 2004. http://www.uddi.org/pubs/uddi-v3.0.2-20041019.htm. Accessed 9 July 2013
22. The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM). http://www.tools.ietf.org/html/rfc6116. Accessed 9 July 2013
23. Elgazzar K, Hassanein H, Martin P (2011) Effective web service discovery in mobile environments. In: P2MNETS, the 36th IEEE conference on local computer networks (LCN), pp 697–705
24. Legner C (2009) Is there a market for web services? In: Service-oriented computing - ICSOC 2007 workshops, pp 29–42
25. Elgazzar K, Hassan AE, Martin P (2010) Clustering wsdl documents to bootstrap the discovery of web services. In: Proceedings of the 2010 IEEE international conference on web services, pp 147–154
26. Xu F, He J, Xu J, Zhang Y (2013) Toward trust-based privacy protection in consumer communication. Int J Secur Appl 7(3):85–98
27. Gao F, He J, Ma S (2012) Trust based privacy protection method in pervasive computing. J Netw 7(2):322–328
28. Machulak MP, Maler EL, Catalano D, van Moorsel A (2010) User-managed access to web resources. In: Proceedings of the 6th ACM workshop on digital identity management, pp 35–44
29. UMA Scenarios and Use Cases (2013) http://www.kantarainitiative.org/confluence/display/uma/UMA+Scenarios+and+Use+Cases. Accessed 30 June 2013
30. Web Framework for Python (2013) http://www.webpy.org/. Accessed 9 July 2013
31. Android 4.0 Platform (2013) http://www.android.com/about/ice-cream-sandwich/. Accessed 9 July 2013
32. Mimerender Python Module (2013) http://www.code.google.com/p/mimerender/. Accessed 9 July 2013
33. Android SQLite (2013) http://www.developer.android.com/reference/android/database/sqlite/package-summary.html. Accessed 9 July 2013
34. The Android SDK (2013) http://www.developer.android.com/sdk/index.html. Accessed 9 July 2013
35. Android: using the contacts API (2013) http://www.developer.android.com/resources/articles/contacts.html. Accessed 9 July 2013

Khalid Elgazzar is a post-doctoral research fellow in the School of Computing at Queen's University. Dr. Elgazzar received his PhD from Queen's University in 2013. He has 8+ years of industrial experience in software design and development and 9+ years of experience in interdisciplinary academic teaching and research. His research interests span the areas of cloud computing, mobile and ubiquitous computing, context-aware systems, mobile services and applications, and elastic networking paradigms. Dr. Elgazzar has received several recognitions and best paper awards at top international conferences. He leads a team on novel ubiquitous cloud paradigms.

Patrick Martin is a professor at the School of Computing at Queen's University. He holds a BSc from the University of Toronto, MSc from Queen's University and a PhD from the University of Toronto. He joined Queen's University in 1984. He is also a visiting scientist in IBM's Centre for Advanced Studies. His research interests include database system performance, web services, autonomic computing systems and cloud computing.

Hossam S. Hassanein is a leading authority in the areas of broadband, wireless and mobile networks architecture, protocols, control and performance evaluation. His record spans more than 400 publications in journals, conferences and book chapters, in addition to numerous keynotes and plenary talks in flagship venues. Dr. Hassanein has received several recognition and best papers awards at top international conferences. He is also the founder and director of the Telecommunications Research (TR) Lab at Queen's University School of Computing, with extensive international academic and industrial collaborations. Dr. Hassanein is a senior member of the IEEE and is a former chair of the IEEE Communication Society Technical Committee on Ad hoc and Sensor Networks (TC AHSN). Dr. Hassanein is an IEEE Communications Society Distinguished Speaker (Distinguished Lecturer 2008-2010).