Elgazzar *et al. Journal of Internet Services and Applications* 2013, **4**:17
http://www.jisajournal.com/content/4/1/17

Journal of Internet Services and Applications
a SpringerOpen Journal

RESEARCH — Open Access

AppaaS: offering mobile applications as a cloud service

Khalid Elgazzar*, Ali Ejaz and Hossam S Hassanein

Abstract

With the huge number of offerings in the mobile application market, the choice of mobile applications that best fit particular objectives is challenging. Therefore, there is a demand for a platform elevating the momentum of mobile applications that can adapt their behavior according to the user's context. This paper proposes *AppaaS*, a context-aware platform that provides mobile applications *as a service*. AppaaS uses several types of context information including location information, user profile, device profile, user ratings, and time to provision the best relevant mobile applications to such a context. AppaaS supports state preservation, where user-specific data and application status are stored for the user's future reference. Experimental validation demonstrates that AppaaS alleviates the burden on mobile users to find applications that work best for a particular situation. It also enables application providers to dynamically control access to the functionality of their applications. Performance evaluation results show that AppaaS can employ cloud elastic resource provisioning to offer flexible scalability, while satisfying certain QoS constraints. Experimental results also support a conclusion that with little overhead handling context information, AppaaS can bring remarkable benefits to provisioning mobile applications as a service.

Keywords: Mobile applications; Cloud computing; Location-based services; Context-aware; Mobile devices

1 Introduction

Mobile devices have become the most pervasive interface that enables access to information services anywhere, anytime. Nowadays, most business enterprises, governments, and public sectors rely on mobile applications to reach a wider range of customers at their most convenience anytime and anywhere. Recent years have witnessed increasing momentum for mobile applications that can adapt their behavior according to user context. Adaptation to context changes reshapes the application's behavior to support service personalization and offer better user experience. Such context includes location information, user profile, user ratings, device profile, and time.

At the time of writing this paper, *Google Play Store* has over than 1 million apps [1], while *Apple Store* is now the home of over than 900,000 apps [2]. Many of these applications are available from various businesses to benefit their customers and improve their experience. With the ever growing application market, a number of open questions related to context awareness arise. From a user perspective, how would a user know that there is an application relevant to his/her current location, and how can a user decide on which application to use from the many available choices. From a business perspective, how can businesses control access to their applications while maintaining a high level of mobility and flexibility to their employees or customers. The system proposed in this paper answers these questions.

We propose *AppaaS*, a context-aware platform for mobile application dissemination and control. AppaaS provisions mobile applications as a cloud service on demand with the appropriate access constraints. AppaaS provides a platform where mobile applications providers and users can match offerings with requests. The platform enables providers to customize their offerings and set proper access constraints according to the privileges of users and their context. It also enables users to conveniently find relevant applications to their situation and save the current application state for future reference. The platform handles several types of context information to ensure that selected applications best serve the user requesting the service. The proposed platform offers a user-friendly interface to facilitate the communication

*Correspondence: elgazzar@cs.queensu.ca
School of Computing, Queen's University, Kingston, Canada

between users and the backend server, where all context processing and data storage management occur.

The remainder of this paper is organized as follows. Section 2 presents motivating scenarios. Section 3 provides brief background and outlines related research. Section 4 describes the proposed architecture. In Section 5 we provide the implementation details of our prototype. The platform functionality is validated in Section 6 and a performance evaluation is presented in Section 7. Lastly, Section 8 draws some concluding remarks.

2 Motivating scenarios

To better understand the advantages and functionalities of such a system, consider the following scenarios.

- *Finding the Appropriate Application*: Suppose that Adam walks into a store with his smartphone and is looking for an application by which he can browse through different offers, product catalogues, or find information relative to a specific product of interest. Imagine how convenient this would be to Adam and how beneficial it would be to the store. AppaaS makes this scenario possible. As soon as Adam enters the store's physical space, the store-specific application gets downloaded onto Adam's smartphone, with his consent. As soon as Adam leaves the space, the application preserves its current state (browsed items, shopping cart, etc.) for future reference and gets uninstalled to free up the mobile resources it is holding. If Adam comes back to the same store (or another related store that shares the same application), the application launches back with the last preserved state that is relevant to Adam.
- *User-aware Applications*: Now, suppose Adam and his friend John enter a store where Adam holds a store membership and John does not. As soon as they walk in, Adam gets the application with the membership privileges, giving Adam exclusive access to membership offers, while John gets the application intended for non-member customers, which might have an open access scheme but is limited to the store's physical location. It is also possible that the same application may handle users with different access privileges. AppaaS facilitates such a model which offers great flexibility to business entities.
- *Controlling Application Behavior*: Now let's assume Adam is at work. Adam holds a position that gives him access to confidential data. As soon as Adam enters the workplace premises, AppaaS installs or activates the enterprise application functionality on his mobile device that allows him access to such data. It is possible that enterprises restrict access to their confidential data wherein coffee rooms or break lounges. Furthermore, enterprises might restrict their business-related mobile applications (or certain functionality) to specific physical places (such as enterprise premises, location of bidders' conference) with variant access privileges according to the user/employee position at workplace.

3 Background & related work

Mobile services that are capable of changing their behavior according to context changes are of particular interest to the provisioning of personalized behaviors [3]. To achieve the desired functionality, such context-aware services typically exploit a combination of several context information, such as location information, user profile, user satisfaction indicators, device features and capabilities, time, etc. [4,5].

Location-based services in ubiquitous computing environments is a rich domain of research [6-9], in which services make use of location information in particular to better provide relevant ubiquitous services. The application of location-based services is popular in many domains such as navigation and traveler services [10,11], shopping and entertainment services (e.g. finding nearby theater), emergency situations (e.g. nearest medical facility), information services [12], and many others [13]. Such services require access to the user's location, which compromises the user's privacy even if several location cloaking techniques are applied [14]. To tackle this challenge, Amoli et al. [14] propose a protocol to preserve privacy in location-based services while satisfying the requirement of accurate location. Similarly, Puttaswamy et at. [15] propose an approach to encrypt location data.

The ultimate objective of AppaaS is to relieve users from having to search and install relevant applications fitting their current situation, especially when there are a huge number of applications in the market. Sharing our objective, Quah et al. [16] propose a mobile application distribution system that retrieves relevant applications based on location information. The system uses the location as the main driver to download mobile applications. Which App? [17] also addresses the same issue in a different way. It proposes a mobile application recommender system that monitors the social interaction and behavior of users to recommend relevant applications accordingly. Toye et al. [18] employ personal information stored on smartphones to customize the behavior of site-specific applications to best fit a particular user. AppaaS shares the same interests of finding relevant applications to a particular context. However AppaaS offers more advantages in managing and controlling the behavior of such applications according to users' access privileges and time constraints. In addition, AppaaS offers the opportunity to preserve the user-specific application state for future reference.

3.1 Context management

Context is any information that can characterize a certain situation relevant to a user including the user itself [19]. Exploiting context information in developing mobile applications opens up new opportunities for a smart generation of applications that dynamically adapt to changing environments and look very personal to the user. Such applications promise a unique user experience. The various context information that AppaaS uses to provide mobile applications as a service are the following.

- Location: The location of mobile user influences what applications users may run on their mobile devices. A user within a certain location might not be aware of relevant application(s) to run in order to get the best service a location might provide. AppaaS employs location information to dispatch the most appropriate application to a user's context. The automatic collection of location information can be obtained in a variety of ways outdoors (e.g. GPS and mobile networks) [20,21], or indoors (e.g. Received Signal Strength techniques) [22,23].
- User Profile: Different users may have different access rights to same applications. For example, within an enterprise different users may have different roles and various job responsibilities, which influence their privilege to access certain information or enterprise-related confidential data. Furthermore, some users could have a variant level of access privilege to such data according to their location or current time. AppaaS exploits such user information in order to assign the user with the proper access rights to an application. Towards this end, AppaaS keeps records of user profiles and credentials.
- Device Profile: AppaaS exploits the device profile in order to identify the appropriate version of a relevant application that fits the device platform. AppaaS takes advantage of communication sessions to collect the device profile information [24].
- User ratings: Web 2.0 and open environments have enabled users to leave feedback and share their consumer experience. User ratings reflect the user perceived quality of service. AppaaS takes advantage of such a feature to choose between applications that provide similar services.
- Time: AppaaS uses time to apply time-based access control over mobile applications. Applications may restrict access to their functionality during certain periods of time. For example, enterprises may allow their employees to access confidential data only during working hours on the premises. AppaaS uses the server time as a reference while handling time-related constraints to avoid synchronization aspects. The server performs the appropriate time transformation according the user's time zone based on current location information.

3.2 State preservation

State preservation refers to how applications can save their latest status and user-specific data for future access. Mobile applications can manage their own state at different levels. For example, applications may use checkpointing techniques [25,26] to suspend and resume their execution for migration purposes. However, the energy and communication cost of migration is inevitably high for mobile applications. Since most of the mobile environments adopt the concept of the virtual machine (VM) [27] to isolate individual applications, the migration process transfers the whole VM and associated resources. To avoid high migration cost, Hung et al. [28] propose an energy-efficient approach that migrates applications between mobile devices and the cloud resulting in lower overhead. Their approach relies on pre-deployment of mobile applications on the cloud side and then transfer only the application state, by which an application resumes its executions.

Application-independent state preservation of user-specific data remains challenging. Current mobile platforms do not natively support state preservation at any level. However, platforms such as Android offer a generic framework for saving an application's state, in which the platform provides two interfaces *onSaveInstanceState* and *onRestoreInstanceState* for saving and retrieving an application state, respectively. Application developers though need to override these interfaces to handle the application state in a state *Bundle*. The *Bundle* [29] is an Android API for passing data, typically in key-value pairs, between various Android activities. Therefore, AppaaS requires application developers to implement their own algorithms and provide two proprietary APIs to save and restore the user-specific data that impacts the behaviour of their applications. AppaaS then uses these APIs to offer a persistent state preservation.

To better understand how to save an application state, we describe how applications are developed for Android. Android applications are composed of a set of activities. Each activity performs a certain task. Activities are managed by an activity stack, where recent activities reside on top. An Android activity may switch between four essential states [30]: *running*, *paused*, *stopped*, and *killed*. An activity is in the *running* state when it is active and running in the foreground of the user's screen. When the activity is out of focus but still alive, the Android platform puts the activity in the *paused* state. In this case the activity maintains all its current state information. When the activity is invisible to the user (put into the background), it means that this activity is in the *stopped* mode. The *onSaveInstanceState* method is called before placing the activity in a background state. Application developers use this method to capture any required user-specific data and pass it to the activity during the *onCreate* event through the *onRestoreInstanceState* interface. The Android system always kills stopped activities first when system memory is required by any other activity. Figure 1 shows the lifecycle of Android application activities and emphasises when the state preservation procedures occur. The rectangle boxes represent the different methods that an Android developer needs to implement to perform operations/functions while activities switch between various states.

4 AppaaS: system architecture

Figure 2 shows an overview of the AppaaS system architecture. The architecture encompasses three main entities, mobile user, space, and AppaaS server. We assume that the user has a smartphone that is capable of running mobile applications, the space is associated with particular mobile applications to better provide a specific service,

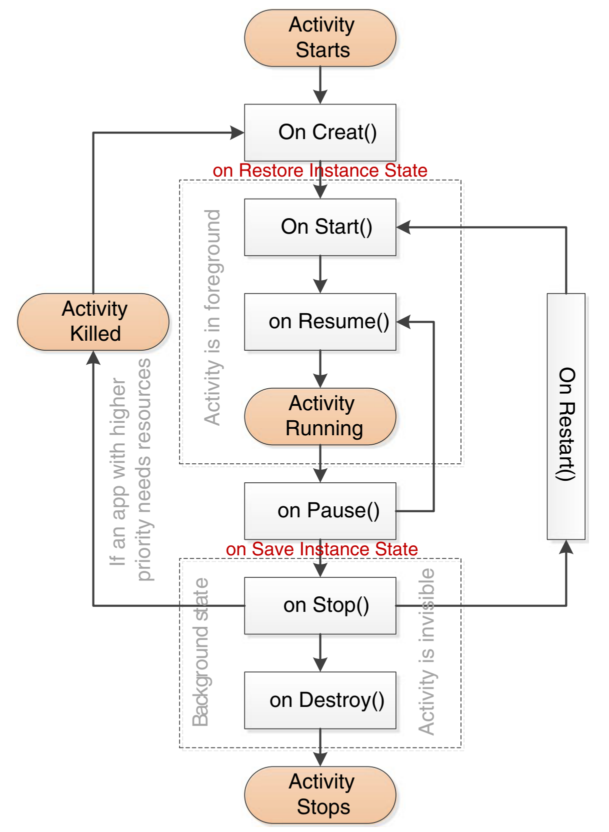


Figure 1 The lifecycle of Android application activities (re-produced from [30]).

and the AppaaS server is hosted on the cloud and is continuously available.

AppaaS maintains a database that stores information about each of the system main entities, namely, users (U), spaces (S), i.e. physical locations, and mobile applications (M). A user $u \in U$ has a set of credentials R_u and an application $m \in M$ has a set of access rights A_m. A user u is granted access to an application m if $R_u \overset{satisfy}{\rightarrow} A_m$. Variations of access rights could be granted to a user according to how the user's credentials satisfy an application's access constraints. Users are identified by their user ID (`user_id`), while applications are identified by a system-generated (`app_id`). Applications are associated with a URL that refers to the application package on their respective market store. Spaces are identified by a numeric location ID (`location_id`) representing a physical location $s \in S$.

Each space $s \in S$ is associated with one or more mobile applications $m \in M$. Once a registered user enters a designated space that is associated with mobile applications, the user's smartphone detects the location and sends the location information to AppaaS context manager for manipulation. AppaaS collects the device profile during communication sessions and retrieves respective context (such as user profile) from the database. AppaaS then checks these various pieces of context information and dispatches the appropriate application to the user's smartphone. AppaaS sends a link to the application of interest, to the user's smartphone, which is then downloaded and installed. If the user has previously used this application, it starts with the last state that the user had left the application with when it was suspended or removed last time. It is worth mentioning that it is possible, according to the provided context, AppaaS returns a response of *"no relevant application fits current context"*, despite that the space has an associated mobile application. This might be because one or more context does not satisfy the access constraints of such an application.

Upon leaving the designated space, the user's smartphone reports that the user is currently outside the space. The AppaaS server reassesses the current context and invokes the *OnExit* procedure to apply prespecified actions set by application providers. These actions include uninstalling or deactivating certain applications. In all cases, the *OnExit* procedure captures the user-specific data and saves it on the server, so that the application can resume with the last user state in the future. Figure 2b illustrates the interactions between the various system entities.

5 Implementation details

We have developed a prototype to test the fundamental functionality of AppaaS. The prototype implements the context and data management on the server side

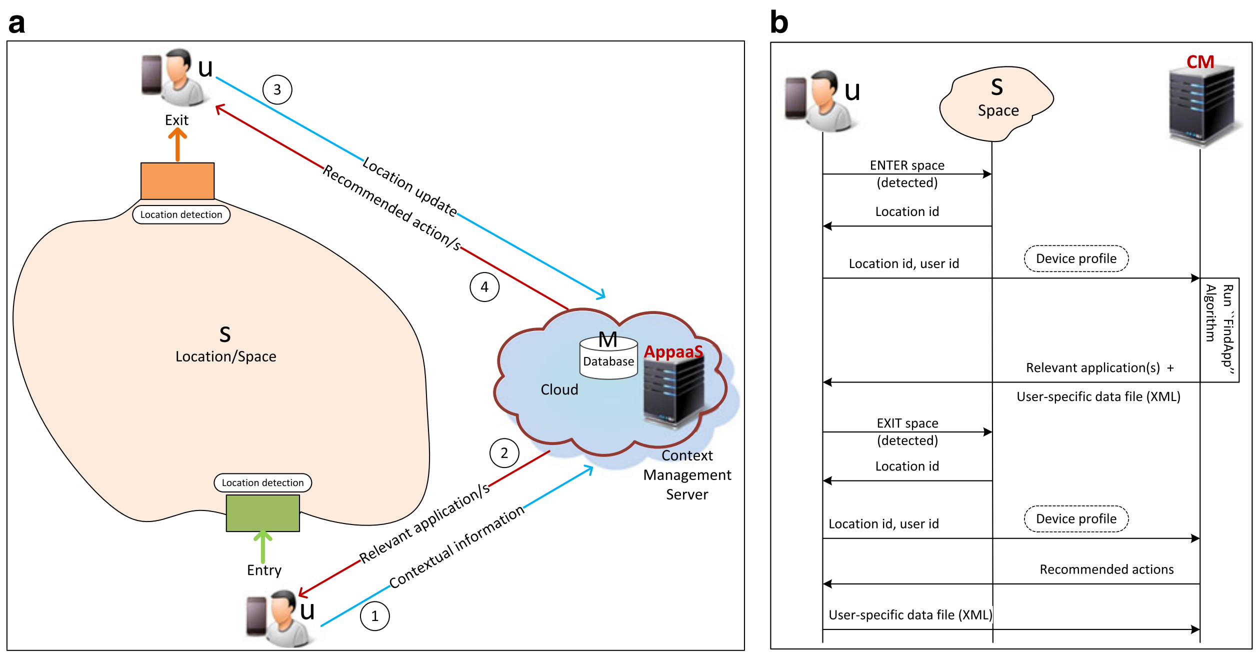


Figure 2 AppaaS Architecture: (a) spatial view of system architecture illustrating different composing entities, (b) logical interactions between various system entities.

while the mobile user interface resides on the mobile device. The user interface is implemented using the latest Android SDK [31]. The server components are responsible for managing the database, handling context information, and dispatching relevant applications along with recommended actions.

5.1 User interface

A user must register with AppaaS to use its functionality. The registration requires the user "`name`", "`email address`", and "`password`". The email address is used as a unique user ID. Once the user is successfully authenticated, AppaaS starts the main process which coordinates between other activities. Figure 3 shows the flow of the main system processes and how they exchange information with one another.

The current version of this prototype identifies location using two Near Field Communication (NFC) RFID tags, one at the space entrance and the other one at the space exit. The entrance tag holds "`ENTER:location_id`" while the exit tag holds "`EXIT:location_id`", where `location_id` represents a numeric value of the location ID. We store the location information on tags in plain text. The user taps the mobile application to scan the location ID. The *NFC scanner* reads the NFC Data Exchange Format (NDEF) message from the RFID tag and reports it to the *Context Manager*, which extracts the locations ID and the user status to query the database for relevant applications and replies with the recommended actions. If an application is found relevant, the *Downloader* downloads and installs it onto the user's smartphone along with the proper access privileges or function constraints. We remark that the use of RFID to identify space is only an instance and other techniques, including use of beacons are also possible.

The *Context Manager* applies Algorithm 1 to find relevant application to current user's context. The *"match"* function applies Equation 1 to match the user's credentials with the application's access requirements.

$$match(R_u, A_m) = \frac{\sum_{i,j} F(r_{u_i}, a_{m_j})}{i * j} \qquad (1)$$

where $r_u \in R_u$ denotes a single user credential and $a_m \in A_m$, denotes a single application's access constraints, i and j are the number of credentials and access constraints, respectively. $F(r_u, a_m)$ represents how much r_u satisfies a_m and it produces values {0,1}. A value of 0 means that if the respective a_m is a constraint at the application level, the application will not be available for the user. If a_m is a constraint at the function level, this functionality will be disabled for the user. Otherwise access is granted. Each relevant application has two scores, relevancy score $(match_m)$ and rating score $(rating_m)$. For simplicity, we consider the two scores are equally important.

Algorithm 1: Find relevant application(s) to a user's context

```
Input: user_id, location_id, device_profile,time
Output: download url of relevant application
1  Function FindApp(user_id,location_id,device_profile,time)
2  Initialize Apps=[] //App collector
3  if location_id has application(s)(M) then
4      foreach $m \in M$ do
5          Initialize $match_m$=0 //application relevancy score
6          Initialize $rating_m$=0 //application rating score
7          // retrieve application access constraints $A_m$
8          if m has no absolute time constraints then
9              if m has open-access then
10                 // m gets max. score
11                 $match_m$ =1
12             end
13             else
14                 // retrieve user profile and credentials $R_u$
15                 Initialize $match_m$=0
16                 foreach $r_u in R_u$ and $a_m \in A_m$ do
17                     // check how much $r_u$ satisfy $a_m$
18                     $match_m = match_m + F(r_u, a_m)$
19                 end
20             end
21         end
22         $m = 0.5 * match_m + 0.5 * rating_m$
23         // add m to relevant Apps
24         Apps=Apps + m
25     end
26 end
27 if Apps is not null then
28     //choose the application with the highest score
29 end
30 return url //if null means no applicaitons
```

5.2 State preservation

Figure 4 illustrates an abstract view of how AppaaS handles the state preservation of mobile applications. As we noted earlier, the current implementation of AppaaS assumes that applications provide two proprietary APIs for the sake of state preservation. The *saveState()* API saves the application's current state, where all user-specific data is saved in an XML file format. The *loadState()* API uploads an application state from an XML file when the application launches. Figure 5 shows the lifecycle of AppaaS state preservation. AppaaS invokes the *saveState()* whenever the application stops running (in Android operating environments, this implements the *onStop()* method). The system generates a state bundle object to save the user-specific parameters. User-specific data is added in a key-value pairs format to a bundle object. This object then is written in an XML file. Each parameter record holds the parameter name, data type, and latest value in memory. The generated XML file is transferred to the server and the context manager updates the database tables or inserts a new record that associates the application ID, user ID, and location ID with a state file name. The state file name is composed of the application ID, user ID and location ID. Listing 1 shows an abstract structure of the state preservation file.

Listing 1 An abstract structure of the application state preservation file in XML format

```xml
<?xml version="1.0" encoding="UTF-8"?>
<application>
        <name> app_name</name>
        <ID> application_id </ID>
</applicaton>

<context>
        <user>
                <ID>user_id </ID>
        </user>
        <location>
                <ID>location_id</ID
        </location>
</context>

<data>
        <parameter>
          <name>paramter_name</name>
          <type>data_type</type>
          <value>parameter_value</value>
        </paramter>

        <parameter>
          <name>paramter_name</name>
          <type>data_type</type>
          <value>parameter_value</value>
        </paramter>
        .
        .
        .
</data>
```

When a user launches a location-specific application, AppaaS checks if the user has a previous saved state of this particular application at this location. If a match is found in the database, the system retrieves the state file to the user side and uploads it into a state bundle. This bundle is used by `onRestoreInstanceState()` callback

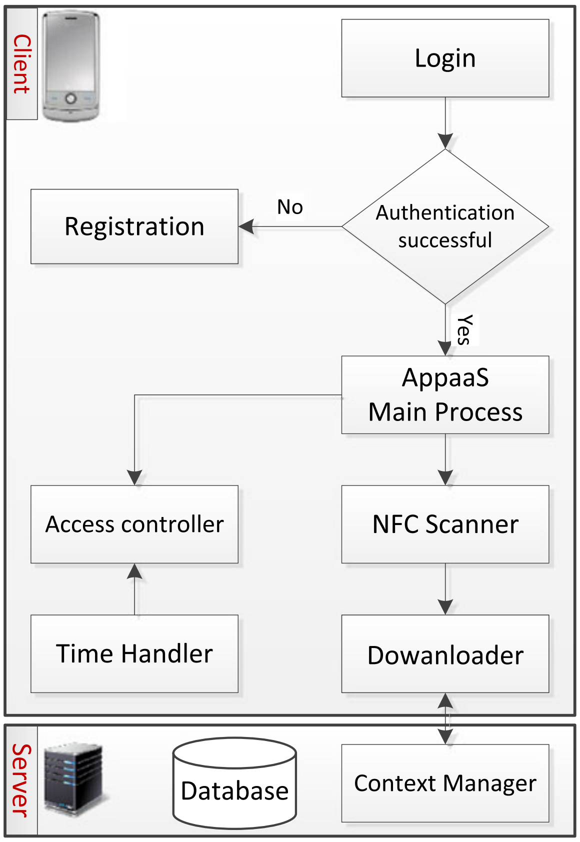


Figure 3 AppaaS components and process flow.

methods during the `onCreate()` to resume the application from where it stopped last time. Although the state file structure is generic, the state information and parameters are application-specific. The current implementation only preserves the last application state and does not keep track of all previous states. It is worth mentioning that AppaaS provides no state preservation support for applications that do not provide appropriate handling of their internal status.

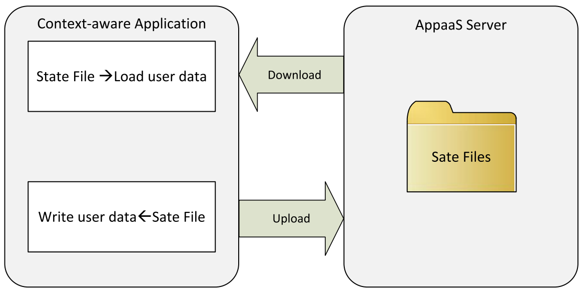

Context-aware Application
State File →Load user data
Write user data←Sate File
Download
Upload
AppaaS Server
Sate Files

Figure 4 An abstract view of state preservation handling in AppaaS.

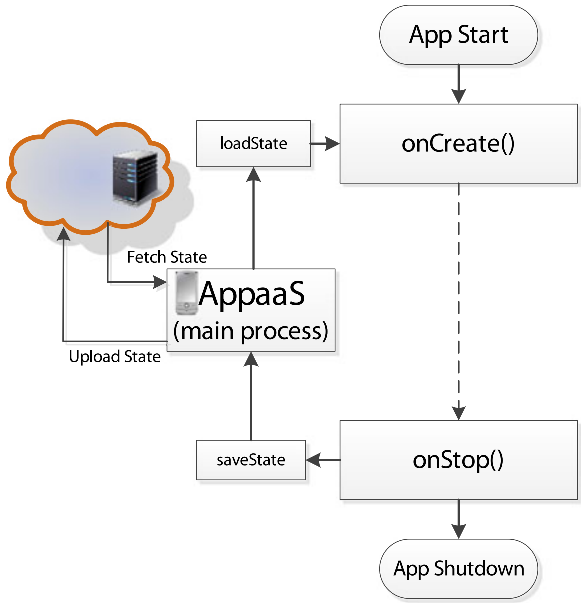


Figure 5 AppaaS state preservation lifecycle.

5.3 AppaaS server-side components

AppaaS system has two components that run on the server: *Database Server* and *Context Manager*. The latter handles context information sent by the user to see if any mobile applications fit this context. The former component (Database server) maintains information about various system entities, users, applications, and a list of registered locations. The database is implemented using MySQL Database Server [32]. It maintains various tables that hold information about each entity such as user profile and credentials, application profile and related access constraints, and location detail which holds information about physical boundaries, associated type of business, working hours, etc. In addition, there are also various database tables that hold the relations between different entities which, for example, relate a certain location to a specific application and users to both locations and applications.

6 Experimental validation

The prototype aims to highlight three aspects of AppaaS functionalities. The first aspect is how AppaaS finds mobile applications which are relevant to a specific context information, in particular user, time, and location. The second aspect demonstrates the ability of AppaaS to customize the application behavior in order to fit certain access rights and constraints. The third aspect shows how AppaaS preserves the application state relevant to a user for future reference.

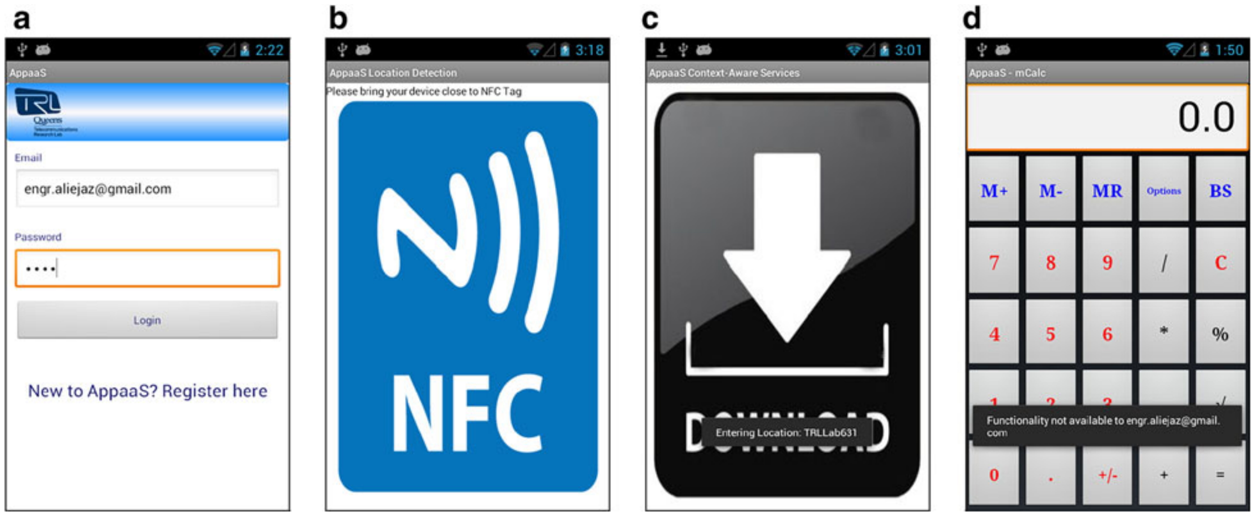
To demonstrate the first functionality, we define in our prototype some locations and associated these locations with mobile applications. The mobile application packages (.apk) are stored on the AppaaS server with each package associated with a *url* for download purposes. The *id* of one of these registered locations is stored on two RFID tags, one represents the location entrance while the other one holds the NDEF message that denotes the location's exit. To start with the system, a user logs into the AppaaS as shown in Figure 6a. The user then taps the RFID tag that represents the location entrance as shown in Figure 6b. The user's context along with the location information is sent to the context manager for processing. The context manager finds the relevant application that fits this context and sends the download *url* back to the user's device. The *Downloader* at the user's device downloads and installs the application as exhibited in Figure 6c.

AppaaS may also apply access limitations to certain functionalities according to the user's access privileges. To validate the second aspect, we developed *mCalc* to demonstrate how AppaaS implements control over access and preserves the user-specific application sate. The *mCalc* is a proof-of-concept mobile application that implements the basic functionality of a calculator with access restrictions. We have restricted the access to some functions to particular users. Therefore, if an unauthorized user tries to perform these restricted functions, an access denial message appears as illustrated in Figure 6d. At this stage, AppaaS supports control over access for mobile applications that implement embedded policy-control for their functions.

The third aspect that our prototype highlights is the state preservation. We use *mCalc* to test this functionality. We set the application *mCalc* to be accessible by a specific user during two specific time periods, 10:00–11:00 AM and 11:15–11:30 AM. This user logs into the system during the first time period and performs calculations. When first period ends, AppaaS initiates the *saveState* procedure and stops the application *mCalc*. The generated state file is sent to be stored at the context manager. Then, AppaaS invokes the uninstall procedure of *mCalc*. Currently, AppaaS always uninstalls the application whenever the user's authorized time elapses. The same user logs into the system again during the second period of time. When the application downloads onto the user's smartphone, the context server finds that there is a previous state preserved for the user for this application. The context manager sends this state along with the application download *url* to start the application with. In this case the user gets a message during the application launching to choose either to resume the application with the previous state or to initiate a clean start.

7 Performance evaluation

Several experiments are conducted to evaluate the performance of AppaaS including the overall response time, system overhead and footprint, and system scalability. The first experiment investigates the response time of the various system activities such as the user authentication, location identification, and the *OnExit* procedure, in which the system applies certain actions on the current application upon leaving its designated space. The second experiment studies the overhead of state preservation and access constraints. The third experiment explores the system scalability. Each experiment is repeated 10 times and the average of the respective performance parameters is calculated. We remark that confidence intervals were found to be small and hence not reported.


a
AppaaS
Email
Password
Login
New to AppaaS? Register here
b
AppaaS Location Detection
NFC
c
AppaaS Context-Aware Services
d
AppaaS - mCalc
0.0
M+
M-
MR
Options
BS
7
8
9
/
C
4
5
6
*
%
0
.
+/-
+
=

engr.aliejaz@gmail.com

Please bring your device close to NFC Tag

Entering Location: TRLLab631

Functionality not available to engr.aliejaz@gmail.com

Figure 6 Prototype screenshots: (a) user login screen, (b) location identification using NFC technology, (c) application download to a user's smartphone, (d) mCalc shows access denial message for restricted functionalities.

The AppaaS user interface is installed on a 3G-enabled Samsung I9100 Galaxy II (Dual-core 1.2 GHz Cortex-A9, 1 GB RAM, Super AMOLED Plus 480 x 800 pixels display, 4.3 inches) with a rooted Android 4.0.4 platform. Although we conducted most of our experiments while the smartphone is connected to a WiFi network, the response time of the various activities of AppaaS user interface is evaluated against both 3G and WiFi connectivity for comparison. The server component of AppaaS implements the context manager and storage and is deployed on the Amazon EC2 cloud. We created a pool of instances of virtual machines of the type *'m1.large'* [33] with an EC2 pre-configured image (AMI) of *'Ubuntu Server 12.04 LTS, 64 bit'*. The server maintains the system database, which includes locations and their associated applications. We use two RFID tags to identify the location boundaries, one tag represents the location entrance and the other tag represents the exit.

7.1 Response time

This experiment measures the end-to-end response time, including both communications and processing time, of the various system activities involving communications between the user interface and the back-end server. A user logs into the system with a valid login ID and password. The user authentication module sends a query to the server to validate the user's credentials and checks whether the user has access to the system. When the user taps the location entrance tag, the location identification module reads the location code and sends a query to the system with the location ID and the user status, including whether the user is entering or leaving the location. The server checks if there are applications associated with this location and responds with a list of download links of relevant applications, if found. When the user exits the location, the *OnExit* module sends the location ID, user ID and an exit status to consult the context manager on the appropriate actions to apply on the user's mobile devices, whether to uninstall or deactivate the associated application. The default action of the current prototype upon exiting a location is uninstalling the location-specific application and saving the user current state on the server side. The main reason behind choosing this default action is to prevent the accumulation of outdated or unused applications. However, disabling the application or allowing the user to make the decision are possible options, if the application has an open access.

Figure 7 shows the response time of the three main components of the user interface: the *User Authentication*, *Location Identification*, and *OnExit Action*. This response time is measured for both WiFi and 3G connectivity with a bandwidth of 17.3 MB/sec and 2.5 MB/sec, respectively. Each of these activities sends a query to the server with an average size less than 0.5 KB. The average query time on the server side in our case is 145 ms. This time basically depends on the size of the database (10 KB in our case) and the number of concurrent requests. While the processing component of the response time is the same in both cases, the communication component varies significantly. The WiFi connectivity yields a 3-fold better response time than the 3G. This gap between the WiFi and 3G increases with higher data transfer requirements due to communication speed.

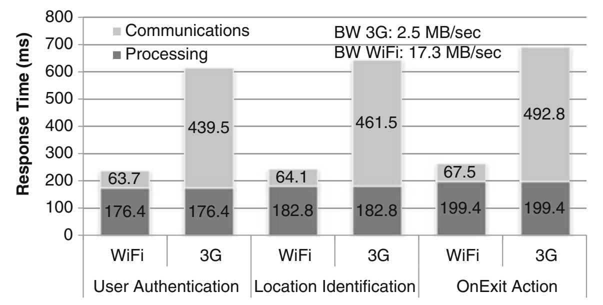


Figure 7 **The response time of various AppaaS activities.**

The overall download time of selected applications depends on the network conditions and the size of the application package. For example, an application package with a size of 1.9 MB takes about 2.5 Sec on WiFi and 19 Sec on 3G. This time is much higher than all other activities. Frequent application download comes at a high cost on mobile resources, specifically battery power and bandwidth. This leads to a debate on which is better, uninstalling unused applications in order to free up the occupied space and prevent unauthorized access, or deactivating their functionalities and avoid frequent downloads of same applications if used regularly. The tradeoff between overhead and convenience is an issue for future consideration.

7.2 Performance overhead analysis

This experiment investigates the overhead that AppaaS incurs on mobile applications should they implement

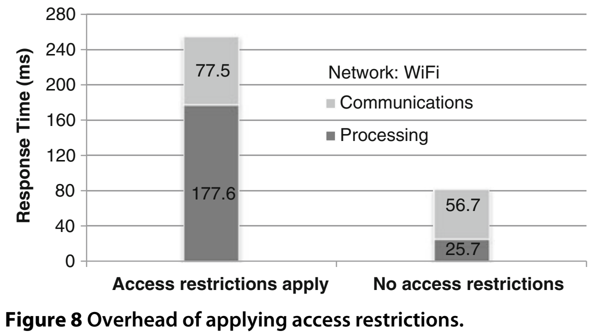


Figure 8 **Overhead of applying access restrictions.**

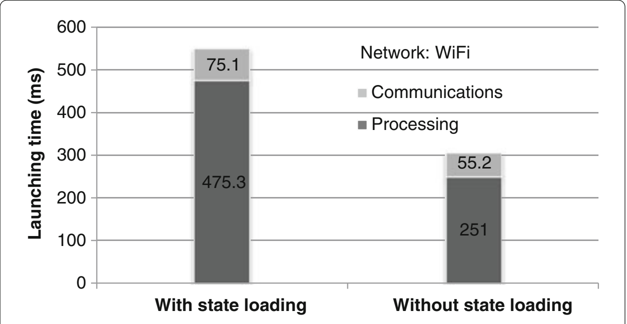


Figure 9 Overhead of state preservation.

access restrictions or launch with a previous state. We employ the *mCalc* to further investigate these aspects since it provides AppaaS with two APIs to capture and restore its internal user state. Figure 8 shows the cost of applying user access control of the mCalc operations. When a user tries to access a certain function of mCalc, AppaaS checks if the user is allowed to perform such an operation or not. AppaaS queries the context manager with the user ID and context. If the user has access to the requested operation, AppaaS sends back a confirmation to the application that the user can perform the operation. Otherwise, the application dispatches a message to the user interface showing that access to the requested functionality is denied. Applying access constraints is performed at the expense of application performance. Figure 9 illustrates another overhead caused by application state preservation. This overhead comes from loading a previous user state, in contrast with starting the same application with no previous state. The size of state file is relatively small in the range of 1 KB. Therefore, fetching this state object from the server side over a reliable network link takes a short time. Such overhead comes with the two benefits: providing users with the appropriate application of a particular location according to the user's context, and gaining more control over the application behavior according to the user's access rights.

7.3 Scalability

We conduct two experiments to evaluate the scalability of AppaaS. The first experiment tests the scalability when a fixed hardware setting is dedicated to serve incoming requests. In this case, we use only one Amazon EC2 instance of the type *m1.large*. The second experiment shows how AppaaS takes advantage of the cloud computing elastic resource provisioning to accommodate increasing number of users while satisfying certain quality of service (QoS) constraints. In this case, the AppaaS system is served by a pool of cloud server instances. In both cases, we apply a varying stress load to evaluate the performance of AppaaS context manager. We use the WAPT load stress tool [34] to apply various loads (number of users) while measuring the system response time. Figure 10 shows the results of the first experiment. We observe that the response time rapidly increases as the number of incoming requests increase in a nonlinear relation. Although AppaaS context manager can serve a relatively high number of concurrent requests, mobile users might not afford long delays. In such cases, the system administrator may set a threshold value after which the server rejects any upcoming requests to maintain a certain level stability and responsiveness.

In the second experiment, we set a response time threshold of 2 seconds as a QoS measure. This means that when a violation occurs, our cloud setup launches another instance of the type *m1.large* from our pool of VMs. In this case no matter how large the number of requests becomes, the response time does not exceed the prespecified threshold value. Figure 11 shows how AppaaS scales up to accommodate increasing requests, while maintaining the response time below the desired threshold. We observe that the new instance takes up to 60–90 seconds to become active and ready to handle requests. This

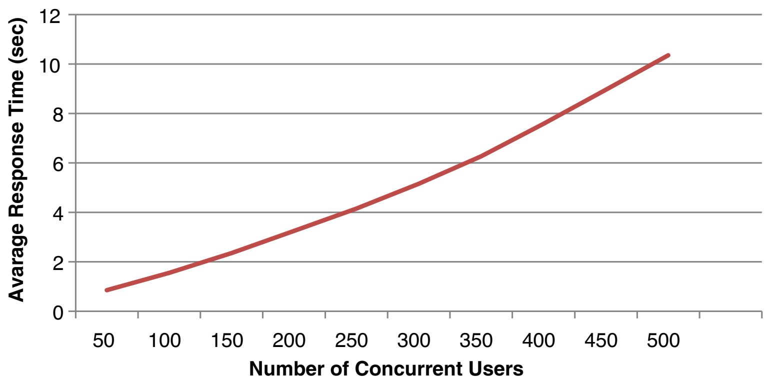


Figure 10 The response time versus varying numbers of concurrent users.

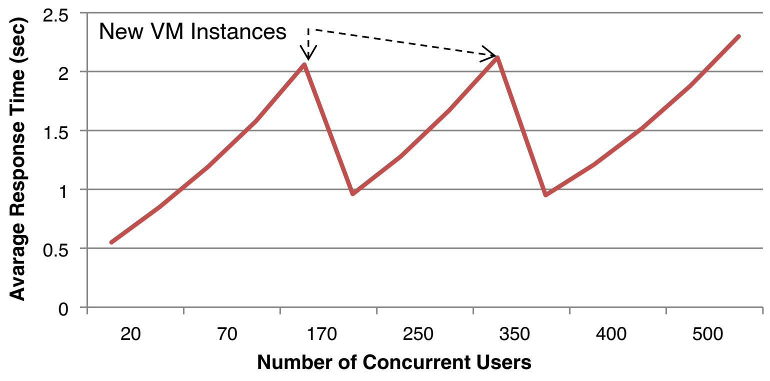


Figure 11 AppaaS uses the cloud *autoscale* feature to improve scalability.

explains why the response time reaches beyond 2 seconds. The EC2 load balancer (ELB) then splits the load between participating instances. Figure 11 also shows that there is a non-negligible overhead incurred by the scheduling of the load balancer, which explains why the response time of $2X$ requests with 2 instances is slightly more than the response time of X requests with a single instance. However, the scheduling overhead is constant and does not increase with further addition of instances.

7.4 AppaaS footprint

Since mobile devices are resource-constrained, mobile applications and services must be highly efficient in resource consumption. The mobile resources of specific concern are CPU cycles, memory, storage, and battery power. These resources indicate how efficient an application is with respect to system resource consumption. In this experiment, we measure the system footprint in terms of memory usage, storage required, and average CPU cycles that are used in cases when AppaaS runs in idle mode (i.e. no activities are performed), and when AppaaS is in active mode (i.e. at least one activity is running). We are concerned with the system footprint at the user side. Table 1 shows the AppaaS system footprint in terms of CPU usage, memory and user application size in both idle and active modes. The energy consumption of the mobile user interface varies according to the running activity and the required data transfer between the interface and the backend server. The lower the data transfer requirements, the higher the energy efficiency of the system while in active mode. However, AppaaS has a low energy consumption profile in the idle mode, as it consumes on average 350 mj/minute.

Table 1 AppaaS system footprint in terms of CPU usage, memory, and the size of the user application in both idle mode and active mode

AppaaS (user side)	Avg. CPU usage (%)	Memory usage (MB)	Storage space (KB)
Idle mode	0.00	35.20	96.00
Active mode	6.65	45.96	96.00

8 Conclusion

This paper presents *AppaaS*, a system for provisioning context-aware mobile applications *as a service*. AppaaS uses various context information including location, user information, user ratings, device profile, and time to provide the appropriate mobile applications to such a context. AppaaS also supports access control over application functionality according to users' privileges and access rights. Currently, AppaaS supports this capability only for applications that implement embedded policy control. However, part of our future extension of this research is to provide an application-wrapping capability to enable this feature for applications that lack support for embedded policy control. A prototype is developed to highlight three features of AppaaS, specifically finding the appropriate application to a certain context, controlling access over an application's functions, and preserving the application's state that is relevant to a particular user for future use purposes. Performance evaluation shows that AppaaS can scale up to accommodate increasing number of users while maintaining desired QoS levels. Experimental validation demonstrates that with little overhead, AppaaS can bring benefits to both users and providers of mobile applications with robust and flexible provisioning models.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
KE proposed the main idea, provided implementation guidance, put the layout of experimental validation and performance evaluation, and drafted the manuscript. AE implemented the prototype and carried out the performance evaluation. HH suggested some modifications on the architecture and recommended the implementation of state preservation. All authors read and approved the final manuscript.

Acknowledgments
This research is funded by a grant from the Ontario Ministry of Economic Development and Innovation under the Ontario Research Fund-Research Excellence (ORF-RE) program.

Received: 16 April 2013 Accepted: 25 October 2013
Published: 4 November 2013

References

1. Google Play Store. https://play.google.com/store?hl=en. Accessed: February, 2013
2. Apple Store. http://store.apple.com/ca. Accessed: February, 2013
3. Toutain F, Bouabdallah A, Zemek R, Daloz C (2011) Interpersonal context-aware communication services. Commun Mag IEEE 49(1): 68–74
4. Elgazzar K, Ejaz A, Hassanein HS (2013) Appaas: Provisioning of context-aware mobile applications as a service In: The IEEE International Conference on Communications (ICC)
5. 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
6. Ahn C, Nah Y (2010) Design of location-based web service framework for context-aware applications in ubiquitous environments In: IEEE international conference on sensor networks, ubiquitous, and trustworthy computing (SUTC), pp 426–433
7. Chatterjee L, Mukherjee S, Chattopadhyay M (2011) A personalized mobile application using location based service In: Advances in computer science and education applications, pp 413–419
8. Junglas IA, Watson RT (2008) Location-based services. Commun ACM 51(3): 65–69
9. Mokbel MF, Levandoski JJ (2009) Toward context and preference-aware location-based services In: Proceedings of the 8th ACM international workshop on data engineering for wireless and mobile access, pp 25–32
10. Husain W, Dih LY (2012) A framework of a personalized location-based traveler recommendation system in mobile application. Int J Multimedia Ubiquitous Eng 7(3): 11–18
11. Shi X, Sun T, Shen Y, Li K, Qu W (2010) Tour-guide: providing location-based tourist information on mobile phones In: The IEEE 10th international conference on computer and information technology, pp 2397–2401
12. Stuedi P, Mohomed I, Terry D (2010) Wherestore: location-based data storage for mobile devices interacting with the cloud In: The 1st ACM workshop on mobile cloud computing and services: social networks and beyond, MCS'10, Co-located with ACM MobiSys, pp 1–8
13. Chow CY, Mokbel MF, Liu X (2011) Spatial cloaking for anonymous location-based services in mobile peer-to-peer environments. Geoinformatica 15(2): 351–380
14. Amoli A, Kharrazi M, Jalili R (2010) 2ploc: preserving privacy in location-based services In: The IEEE 2nd international conference on Social Computing (SocialCom), pp 707–712
15. Puttaswamy KPN, Zhao BY (2010) Preserving privacy in location-based mobile social applications In: HotMobile: The 11th workshop on mobile computing systems and applications, pp 1–6
16. Quah JTS, Lim LR (2011) Location based application distribution for android mobile devices In: Proceedings of the IASTED international conference on wireless communications, pp 118–123
17. Costa-Montenegro E, Barragáns-Martínez AB, Rey-López M, Mikic-Fonte F, Peleteiro-Ramallo A (2011) Which app? A recommender system of applications in markets by monitoring users' interaction In: Proceedings of the IEEE International Conference on Consumer Electronics (ICCE), pp 353–354
18. Toye E, Sharp R, Madhavapeddy A, Scott D (2005) Using smart phones to access site-specific services. IEEE Pervasive Comput 4(2): 60–66
19. Dey AK (2001) Understanding and using context. Pers Ubiquitous Comput 5: 4–7
20. Ahn J, Heo J, Lim S, Kim W (2008) A study on the application of patient location data for ubiquitous healthcare system based on lbs In: 10th international conference on advanced communication technology, vol. 3, pp 2140–2143
21. Xin C (2009) Location based service application in mobile phone serious game In: International joint conference on artificial intelligence, pp 50–52
22. Altintas B, Serif T (2012) Indoor location detection with a rss-based short term memory technique (knn-stm) In: IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops), pp 794–798
23. Diaz J, de A Maues R, Soares R, Nakamura E, Figueiredo C (2010) Bluepass: an indoor bluetooth-based localization system for mobile applications In: IEEE Symposium on Computers and Communications (ISCC), pp 778–83
24. Al-Masri E, Mahmoud QH (2010) Mobieureka: an approach for enhancing the discovery of mobile web services. Pers Ubiquitous Comput 14: 609–620
25. Biswas S, Neogy S, 2010 A mobility-based checkpointing protocol for mobile computing system. Int J Comput Sci Inf Technol 2(1): 135–151
26. Tuli R, Kumar P (2011) Analysis of recent checkpointing techniques for mobile computing systems. Int J Comput Sci Eng Surv 2(3): 133–141
27. Sevinç PE, Strasser M, Basin D (2007) Securing the distribution and storage of secrets with trusted platform modules In: Proceedings of the 1st IFIP TC6 /WG8.8 /WG11.2 international conference on Information security theory and practices: smart cards, mobile and ubiquitous computing systems. Springer-Verlag, Berlin, Heidelberg, pp 53–66
28. Hung SH, Shih CS, Shieh JP, Lee CP, Huang YH (2012) Executing mobile applications on the cloud: framework and issues. Comput Math Appl 63(2): 573–587
29. Android APIs (Bundle). http://developer.android.com/reference/android/os/Bundle.html. Accessed: August, 2013
30. Android Activity Lifecycle. http://developer.android.com/reference/android/app/Activity.html. Accessed: February, 2013
31. Android SDK. http://developer.android.com/sdk/index.html. Accessed: February, 2013
32. MySQL Database Server. http://www.mysql.com/. Accessed: February, 2013
33. Amazon EC2 Instance Types. http://aws.amazon.com/ec2/instance-types/. Accessed: August, 2013
34. Web Application Testing (WAPT). http://www.loadtestingtool.com. Accessed: February, 2013

doi:10.1186/1869-0238-4-17
Cite this article as: Elgazzar *et al.*: AppaaS: offering mobile applications as a cloud service. *Journal of Internet Services and Applications* 2013 **4**:17.