SOPHRA: A Mobile Web Services Hosting Infrastructure in mHealth

Richard K. Lomotey, Shomoyita Jamal and Ralph Deters
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
{richard.lomotey, s.jamal}@usask.ca, deters@cs.usask.ca

***Abstract*—The use of mobile devices such as smartphones and tablets, and other ICT tools to facilitate healthcare delivery in the medical landscape, known as mHealth, has witnessed a phenomenal rise recently. In most mHealth systems, mobile devices are employed as services and health information client consumers. Thus, healthcare professionals use these devices to consume services which are running on back-end platforms. However, in a research collaboration with the Geriatrics Ward of the City Hospital in Saskatoon, Canada, we have identified a huge potential in facilitating the mobile device as a service hosting node. Hence, we developed a physically distributed information infrastructure, called SOPHRA, which aids the healthcare professionals to securely access and share patients' medical information which are hosted on their mobile devices. Since mobile devices establish communication over wireless channels which can sometimes be unavailable, the proposed mobile hosting framework faces the challenge of reliable and real-time message propagation to the mobile participants.**

This paper presents the adopted methodologies employed in implementing SOPHRA to address the aforementioned challenges. A cloud-oriented middleware is implemented which enables the mobile participants to reliably communicate in soft real-time. Furthermore, the records of the patients are modeled as Web Services (WS) which aids medical information to be passed across the system components; and these WS are independently replicated on the middleware to ensure high information availability. Currently, SOPHRA supports both SOAP and RESTful Web Services protocols and facilitates information exchanges over Wi-Fi.

Keywords-REST; mobile cloud computing; middleware; healthcare information systems; resources state change

I. INTRODUCTION

The ever growing popularity of mobile devices such as smartphones and tablets as reported by Gibbs [1] is shaping the day to day businesses in enterprises. In less than a decade ago, these devices were just tools for exchanging data in the form of voice and simple SMS messages. Now, market reports suggest that there is a promising future for the mobile commerce space [2]. Furthermore, most of today's smartphones and tablet devices in production have good backings for varied networks and protocols. Thus, connectivity can be established between mobile participants via 4G, Wi-Fi, and Bluetooth [4].

As the commerce field is witnessing the mobile boom, other non-commercial enterprises such as mHealth is equally experiencing growth in the research on the use of mobile devices and other ICT tools for better healthcare delivery [20, 21]. Relentless efforts have been made by researchers to employ the potential of the evolving Web to deliver services on the mobile device for better healthcare [21]. However, most of these researches focus on the mobile devices as client consumers especially in cases where medical information are modeled as heterogeneous Web services.

In a current research in collaboration with the Geriatrics Ward of the City Hospital in Saskatoon, Canada, we have identified a huge potential in facilitating the mobile device as a host of medical information. Mobile hosting is a new research trend where the mobile device is modeled as a server or provider of Web services [4, 5, 7, 8]. Srirama et al. [5] noted that an advantage of mobile hosting is turning the mobile host into a multi-user node. From the view point of our medical partners, the mobile Web services hosting has improved on the management and accessibility of medical records of their "aged" patients.

However, mobile devices primarily communicate via wireless channels which can be unstable due to user mobility and bandwidth fluctuation. Thus, in the current research, the challenge is how to propagate messages reliably and in real-time to the healthcare mobile participants. Another challenge is the integration of our proposed infrastructure into the existing Healthcare Information System (HIS) at the Geriatrics Ward.

To address these challenges, we developed a mobile distributed infrastructure, called *SOPHRA*, which employs a cloud-centric middleware platform that aids the healthcare professionals to securely share and access their mobile hosted health records reliably and in real time. The health records are modeled as Web Services (WS) which can be propagated between the system components. These WS which are hosted on the mobile devices are also replicated independently on the middleware to ensure high data and services availability. The *SOPHRA* framework supports both SOAP and RESTful Web Services protocols and currently facilitates message exchanges over Wi-Fi using HTTP. The details of the architecture as illustrated in Fig. 1, are explained in the next section.

The remainder of this paper is organized as follows. Section II expounded on the *SOPHRA* architecture and in Section III, the implementation of *SOPHRA* is described. An evaluation is conducted on the performance of *SOPHRA* in Section IV. Section V investigates the related works on mobile hosting of Web services and the paper concludes in Section VI with the summary of our findings and future research.

978-0-7695-4754-1/12 $26.00 © 2012 IEEE
DOI 10.1109/MobServ.2012.14

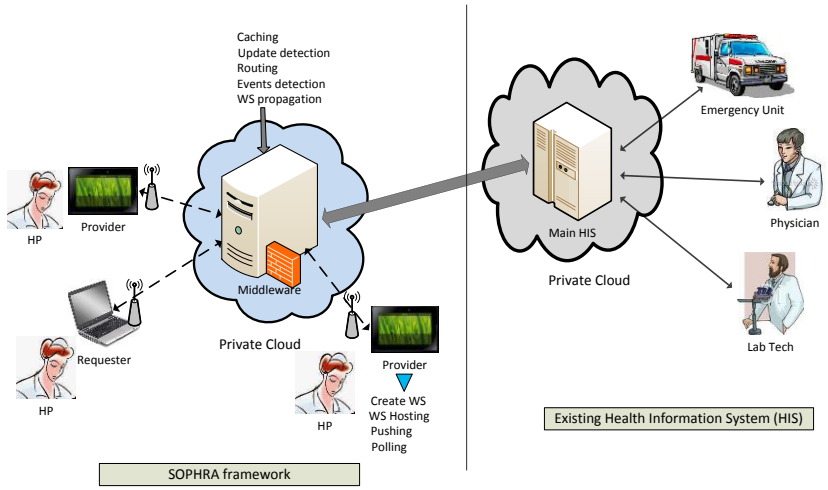

Figure 1. The SOPHRA Architecture being integrated into the existing HIS of the Saskatoon City Hospital.

II. ARCHITECTURAL DESIGN OF SOPHRA

In order to enable the mobile devices of the healthcare professionals in the Geriatrics Ward to host the patients' medical information, *SOPHRA* focuses on addressing the challenges of resources and services state change management. Issues concerning the reliability and freshness of data arise in mHealth systems due to the inherent transient connectivity in wireless networks. Also, data delivery in real-time to the healthcare professionals is a concern since *SOPHRA* is a mission critical infrastructure. To overcome the challenges, *SOPHRA* employs a middleware framework which is hosted on an internally owned private cloud platform. The middleware facilitates the mobile hosts to be reached via Wi-Fi by other mobile participants (healthcare professionals). The architectural design of *SOPHRA*, as depicted in Fig. 1, is classified into three tiers namely: mobile service requesters, middleware, and mobile service providers/hosts. The mobile devices connect to the middleware through 802.11g Wi-Fi 54Mbps connection. Also, the medical records such as patients' demographics, vitals, medical history etc. are modeled as Web Services [13] following both the SOAP [11] and REST [9] design paradigms.

The RESTful Web Services are modeled in the JSON format while the SOAP services follow the standard XML protocol. As a result of using these open standards to design the WS, *SOPHRA* is well integrated into the existing Health Information System (HIS) of the Saskatoon City Hospital.

As shown in Fig. 1, the middleware ensures the HTTP request-response messages routing between the healthcare professionals (HP). Each RESTful request from the mobile device of a HP is a tuple containing the elements as shown below:

{HTTP method, requester_id, timestamp, {mobile_host, resource_id}}

The HTTP method informs the middleware about the nature of the request; whether it is a write request (POST or PUT) or a read request (GET or HEAD). Since security and integrity is paramount in *SOPHRA*, all healthcare professionals issue requests with their uniquely assigned identification number which is the *requester_id*. The requester id informs the middleware and the mobile host about the credibility of the requester. Also, *timestamps* are put on all requests which aid the middleware to determine the global world view of subsequent requests. The use of timestamps facilitates the race condition management between a write and a read request. Thus, the first request is always treated with a higher priority. Furthermore, since the middleware stores states of the entire mobile participants, the requester has to explicitly specify the intended mobile host (*mobile_host)* so that the middleware can forward the request to that mobile host. Additionally, the identification number of the desired resource, *resource_id* that the requester wants to access has to be specified. Based on the tuple, the middleware formulates a URI that establishes a bi-directional communication channel to the intended mobile host.

The SOAP requests follow the same description as the RESTful requests but with a slight variation. Since SOAP accepts only the HTTP POST method for both read and write operations, the tuple in SOAP is modified as shown:

{HTTP POST, cache_request, requester_id, timestamp, {mobile_host, resource_id}}

The use of HTTP POST in SOAP to pass XML data for all requests can be confusing especially in a system like *SOPHRA* where middleware caching is employed. Hence the tuple in SOAP contains an additional element defined as *cache_request*, which can be a text say "cacheable" or "non-cacheable", that informs the middleware whether the request should be cached or not respectively.

Due to the intermittent loss in connectivity in a Wi-Fi network, *SOPHRA* uses the asynchronous messaging mechanism for all communications. Hence, healthcare professionals (both requesters and providers) are able to send

requests without waiting for an acknowledgement before making other requests. Also, the middleware employs event notifications following the *publish/subscribe* model to inform registered mobile participants of message updates. All the mobile participants are registered by the middleware for particular services from preferred providers. Thus, if a provider's resources or services are updated, the middleware propagates (publishes) the update message to all the other healthcare professionals who have subscribed for services from that provider.

As well, *SOPHRA* employs resources and services replication technique on the middleware. The replication approach is important for ensuring high availability of the data and information since the connectivity of the mobile providers cannot be guaranteed in the Wi-Fi network. In case a particular mobile provider is unreachable, the requester can access the providers' replica resources that are stored in the middleware cache. As a result, the middleware acts as a back-up for the mobile providers.

In order to keep resources state updated for real-time delivery of service to the requester, the mobile providers use long-polling to fetch updates from the middleware and pushing techniques to notify the middleware of updates.

A. Update Management

The *SOPHRA* infrastructure uses *read-your-write* consistency based on the report by Monash [17] that this consistency technique works best for NoSQL back-end systems. Thus, all updates applied to resources or services on a mobile host are visible to both the provider and other healthcare professionals on the next read request. For instance, if a provider sends resources state change message to another provider/requester, the latter will update its former state with the new state and only the new state will be seen subsequently on every read request. In addition, the identification of which mobile service hosts have the latest service updates can be traced from the middleware's cache. Thus, the mobile participants that must be updated can make read requests (e.g. HTTP GET) to the mobile provider who has the latest update of a resource or service.

The use of timestamps also ensures reliability of the messages, hence, when a requester receives a response, the requester sees when the last changes were made to the resource or service. Also, resources of disconnected providers that are served from the middleware are time stamped which aids the requester to determine how old the data is and how long the intended provider has been unavailable. The middleware also notifies the requester that the intended provider is temporary unavailable. Furthermore, when updates are applied to resources state, the new states are pushed to all subscribers within an inconsistency window so that eventually, all the healthcare professionals will receive the update. The next section discusses the middleware and how all the functionalities are achieved.

B. Middleware Platform/Framework

The cloud-based middleware serves as a router for all the asynchronous messaging between the mobile service requesters and the mobile service providers. All the information of each mobile host is logged in the middleware storage for callbacks; as a result, the middleware uses pushing to send updates to the mobile participants that have subscribed for services in soft real-time. Furthermore, the middleware can be centralized or distributed but this paper presents a centralized system to minimize the challenges of data inconsistency.

The internal framework of the middleware is shown in Fig. 2. When an HTTP request is invoked by a healthcare professional, the request first identifies the *HTTP Requester Interface* of the middleware through a specified port. All requests are then forwarded to the *Transaction Manager (TM)* which determines the nature of the request (e.g. read or write request). The *TM* then communicates with the *Propagation Controller (PC)* to notify the appropriate mobile host/provider of the request through the *HTTP Provider Interface*. The response of the provider is sent back to the requester through the *TM*. All successful responses are also stored in the cache by the *Persistence Manager (PM)*. In case the *PC* responds that the specified mobile host is unavailable, the *PM* searches through the cache, where the mobile hosts' resource replicas are stored, and return the result. In addition, anytime the mobile host's resources are updated, the updates are pushed to the middleware through the *HTTP Provider Interface*. The updates are then forwarded by the *Propagation Controller* to the *Persistence Manager* for it to update the cache. Furthermore, all events are determined by the *Event Handler* for appropriate notification (e.g. resource state change, client connected, and client disconnected).

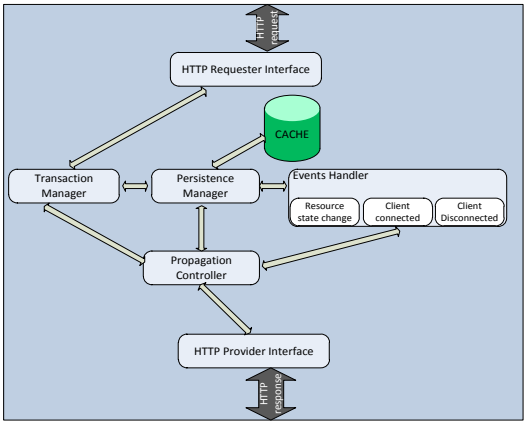

Figure 2. Middleware features.

The main functionalities of the middleware are discussed below.

1) Read/Write Request : The middleware controls the race condition between a read request and a write request that are issued at the same time. This is achieved by introducing timestamps on each request. The transaction

manager determines which request comes first and processes that request with a higher priority.

2) Caching : Since REST affords stateless communication between system components, RESTful resources of the mobile providers are replicated independently in the middleware cache. The cache has a RESTful interface that supports the CRUD methods. This means that the content of the cache can be created, read, updated, and deleted. Also, based on the SOAP message passing system in *SOPHRA*, SOAP services are cached as well. Additionally, the cache stores states of all the mobile providers that have subscribed for services.

3) Routing : All communications are routed through the middleware since some mobile participants can disconnect due to network instability. In such cases, requesters who want to access the resources and services from disconnected mobile hosts can access it from the middleware.

4) Events : The Events Handler is responsible for actions such as resources and services updates, and connection and disconnection of a mobile host. When updates arrive in the cache, the new states of resources override the older versions in the cache. The Events Handler then informs the Propagation Controller to propagate the changes to all healthcare professionals who have subscribed via HTTP. Also, the *Events Handler* notifies the propagation controller of all connected clients so that updates will be sent to them in soft real-time. When write requests are sent to disconnected providers, the middleware creates the services and stores them in the cache. The updates are pushed from the middleware to disconnected mobile nodes when they reconnect. Also, a mobile service requester is notified if a host is disconnected.

C. *Mobile Framework*

The mobile framework consists of the mobile service requesters and the mobile service providers/hosts. The requesters are the users plus the devices that send HTTP requests to the middleware and the provider either for a resource to be created or to fetch an existing resource or service. The mobile service hosts are the users plus devices that serves the resources and services. Additionally, a mobile service host can act as a service requester. In our architecture, the mobile service hosts behave as *Web servers*.

Recently, more developers are looking towards HTML5 mobile applications as a solution for targeting multiple devices and platforms. This is because the Web browser is becoming the default platform and its de facto standards are HTML5 and JavaScript [18]. Smartphones and tablets support the development of native applications which are platform specific. However, these devices also have an embedded browser which makes it possible to deploy mobile web applications. Thus, with the advancement of HTML5, as evidenced in the HTML5 stack in Fig. 3, a hybrid app methodology is adopted in the design of the mobile architecture to build the mobile web app that looks and functions as a native app.

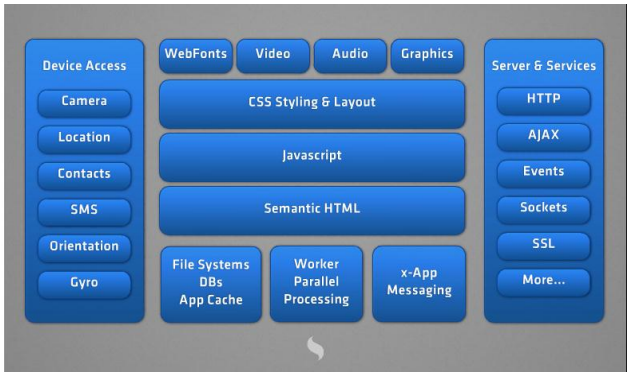

Figure 3. HTML5 stack on the mobile platform [18]

As illustrated in the HTML5 stack (Fig. 3), the native functionalities that have access to the device can be imported into the mobile Web app. HTML5 also supports server and services which makes it affordable to consume Web services. Hence, we are able to embed a Web server into the mobile device in order to support the HTTP requests. Also, Web services can be invoked in the embedded browser regardless of the mobile device platform. Embedded browsers provide rich graphical user interfaces and support multiple Web based languages such as HTML, JavaScript, CSS, and so on.

Though the mobile providers have the same functionalities as the requesters, they have additional functionalities. The idea is that, the mobile provider will act as a Web server as well as a consumer.

The WS created are hosted on the mobile hosts as resources and services. All newly created resource or service is given globally unique identifier (id) which enables other mobile participants to select or search for that resource or service. The mobile hosts' resources are accessed using URIs that is provided by the middleware.

Also, the responses from the mobile hosts make use of the HTTP status codes. This notifies the middleware on the state of a particular resource or service. The requester is informed whether there is an error in a request (with a 400 status code) or the request is successful (with a 200 status code).

Furthermore, the mobile host uses the combination of polling and pushing to synchronize data on the middleware. There is the tendency of having outdated information on a mobile host especially in cases of network communication loss. Thus, new updates on the middleware from other healthcare professionals might not reflect at the providers' side at the same time. To address the problem of keeping outdated information, the mobile data validation is done using HTTP HEAD and GET requests. Since network connectivity is crucial in mobile technology, a periodic HTTP HEAD request is sent to the middleware for only the *Etag* – which is a unique attribute of a Web resource. The mobile nodes compare the Etag to determine whether the local copy of the resource is the same as the incoming resource or service from the middleware. A change in Etag is an indication of a possible update of an existing resource. The screenshots of the application on an Android tablet is shown in Fig. 4.

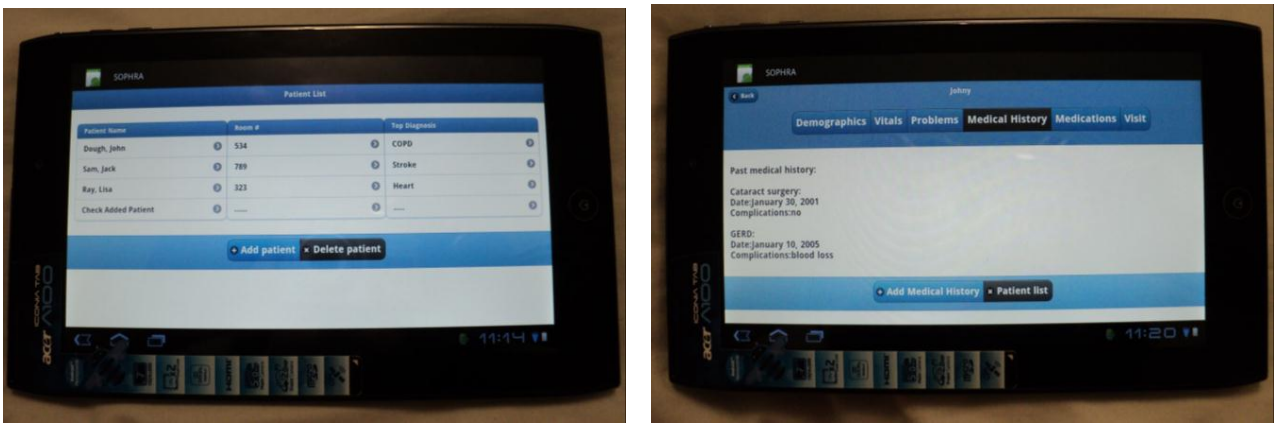

Figure 4. SOPHRA UI on an Android tablet (showing dummy patient records due to privacy reasons)

III. IMPLEMENTATION OF SOPHRA

A. The Mobile Implementation

The mobile platforms that are considered in the implementation of *SOPHRA* are the BlackBerry Playbook, Android tablet devices, and Apple iOS. Since the mobile platforms are heterogeneous in terms of their underlying operating systems, building a native app will mean that multiple versions of the same application have to be built in different programming languages. To avoid this situation, we employed the embedded browser pattern to write a single code base that is deployed on multiple platforms.

Though the embedded browser design approach facilitates the deployment of cross-platform mobile applications, there are some challenges encountered during the user interface (UI) design. Due to the mobile WebKit diversity, the UI is rendered differently on different mobile platforms. However, as explained by Pearce [18], mobile Web technology frameworks can be employed for: building enthralling user interfaces, easy browser interoperability, narrowing the gap between native and mobile web designs, and creating consistent application architectures. Thus, we adopted a mobile Web technology framework called jQuerymobile [18] which we embedded in HTML5 to overcome the challenges faced with WebKit diversity.

The BlackBerry version of the application is built as a BlackBerry WebWorks [3] project, which is invoked in the embedded browser. The same WebWorks code is reused to compile an Android WebView [3] project. The WebView is a framework that supports the deployment of mobile Web applications on the Android devices. In addition, the same code is exported to the iOS platform and compiled as an Xcode [3] project.

However, the Internet Protocol (IP) address of a mobile device changes as the user moves from one hotspot to another. In addition, registering the device name within a network domain is not ideal because the user can move from one network domain to a different domain. In view of these challenges, the mobile embedded Web server is implemented to establish a dual communication channel between the mobile device and the middleware. Hence, the mobile requester is able to send an HTTP request to the provider using the IP address or the registered computer name of the middleware – which is running on the private cloud.

B. Middleware Implementation

The middleware is built using Erlang since the Erlang programming platform supports concurrency and database distributions. The middleware is built on the Generic Server Behaviour (gen_server) process. The gen_server is a module that facilitates the implementation of a server for the request-response interaction between the mobile participants and the middleware. Also, the DETS storage facility of Erlang is used to build the cache which stores the replicas of the providers' resources. The Erlang code showing read and write commands in the DETS storage for the middleware implementation is shown in Fig. 5.

```
insert(Key, Value) ->                    %% Write request
    case cache:search(Key) of            %% Check if key is already present
        {ok, Pid} ->
          element:update(Pid, Value);    %% Update an existing WS
        {error, _} ->
          {ok, Pid} = element:create(Value),   %% Create a new WS
            cache:insert(Key, Pid)             % % Store the newly created WS
        end.

search(Key) ->                                 %% Read request
      try
        { ok, Pid} = cache:search(Key),        %% Fetch process id for key
        { ok, Value} = element:read(Pid),
        { ok, Value}
    catch
        _Class:_Exception ->
        {error, not_found}      %% Return this line if the WS is not in cache
      end.
```

Figure 5. Snippet of Erlang code showing how the WS replicas are stored and fetched from the DETS table.

IV. EVALUATION

The BlackBerry Playbook simulator and Android tablet emulator are the mobile platforms that are put forward for the testing in the experiments. All the experiments are simulated in order to enable us to have more control over the testing environment. The BlackBerry Playbook simulator which is running as the requester is deployed on a computer with the following specifications in the laboratory: *Processor: Intel Core i5, CPU 650 @ 3.66GHz 3.56GHz, RAM: 4GB (2.99GB usable), System 32-bit operating system.* The Android 4.0 tablet emulator which is running as the mobile host is configured on a computer with the following specifications in the laboratory: *Processor: Intel Core i5, CPU 650 @ 3.22GHz 3.22GHz, RAM: 8GB, System 32-bit operating system.* The middleware is hosted on a privately owned cloud platform with the following specifications: *Windows 7 Enterprise, Service Pack 1, Processor: Intel(R) Xeon(R), CPU E5140 @ 2.33GHz 2.33GHz (2 processors), Installed memory (RAM): 16.00GB, System type: 64-bit Operating System.* Furthermore, the mobile clients are configured to connect through the University of Saskatchewan secure Wi-Fi network using 802.11g.

The first experimental setup that is deployed focused on the round-trip time between a mobile requester and a host. The mobile host was initially populated with 600 WS representing patients' demographic information; and each file size is approximately 11kb. The mobile concurrent requesters execute 30 HTTP GET requests and the result is illustrated graphically in Fig. 6 and in tabular form in Table I for both REST and SOAP Web services.

TABLE I. COMPARISON OF RESPONSE TIMES OF THE WS

Web Service	Average response time for sending 30 requests (ms)	Standard deviation	Maximum request rate (request/s)
REST	584.47	174.94	51.33
SOAP	642.39	200.79	46.70

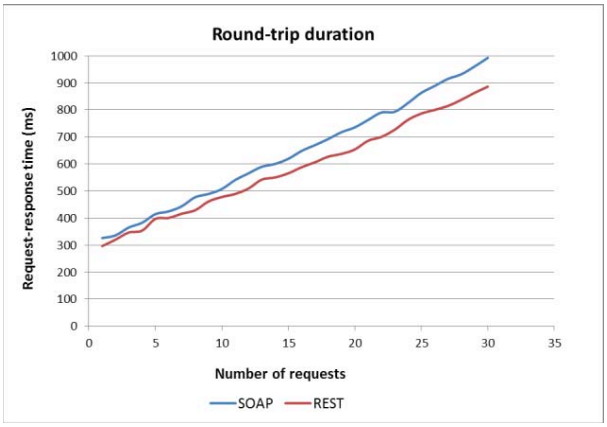

Figure 6. Mobile requester-host interaction through the middleware.

In the experiment, it is observed that there was no loss or corruption in requested data. The results show that the REST-WS are consumed efficiently at higher response time since the percentage increase in maximum request rate is 9.91% in favor of REST. The low performance of SOAP WS could be attributed to the verbosity of the XML data that is exchanged in each request-response interaction as observed by Aijaz et al. [19]. However, the performance of *SOPHRA* regarding SOAP Web services is encouraging considering the fact that it has higher concurrency requesters compared to the results in [8]. Also, the healthcare professionals using *SOPHRA* are comfortable with the response time.

A second setup was deployed to determine the average response time of the middleware in cases involving an unavailable mobile host. The middleware cache is populated with 10000 WS representing patients' demographic information; and each file size is approximately 11kb. Fault injection technique is used to turn off the mobile host while we observed the response time of the middleware. The mobile requesters issue 70 concurrent read requests for both REST and SOAP Web services and the result is presented in Table II and graphed in Fig. 7.

TABLE II. RESULT OF THE MIDDLEWARE RESPONSE

Web Service	Average response time for sending 70 requests (ms)	Standard deviation	Maximum request rate (request/s)
REST	60.10	33.70	1164.73
SOAP	381.30	192.71	183.58

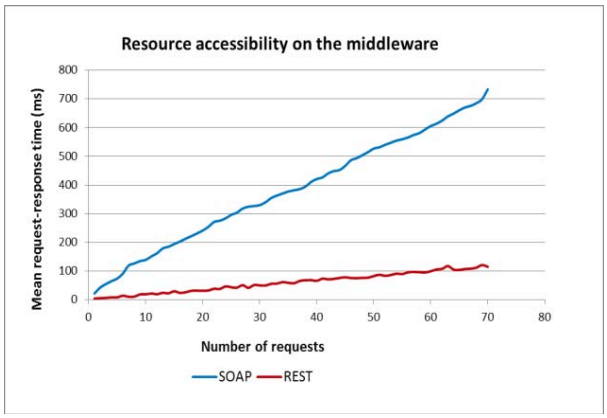

Figure 7. Middleware response rate in cases of unreachable mobile hosts

The result in the second setup shows that the middleware responds efficiently in cases involving unavailable mobile hosts. This result is encouraging since it shows that in case a particular mobile host is unreachable, the services of the disconnected host can be accessed on the middleware by other healthcare professionals in real time.

Another setup was deployed to measure the scalability of the middleware by simulating the concurrent activities of 600 mobile participants; who send requests ranging from 600 to 40000. This is to determine the performance of the middleware when the users as well as the users' requests increase. A load generator is configured to send concurrent HTTP requests to the middleware for the replicated Web services. The size of each resource is 11kb and the load generator sends requests at a rate of 1 request per 10 seconds following the exponential distribution of mean, 0.1

requests/second. The outcome of the scalability testing is presented in Table III and graphed in Fig. 8.

TABLE III. OUTCOME OF THE MIDDLEWARE SCALABILITY TEST

Mean Throughput (req/s)	Maximum Throughput (req/s)	Minimum Throughput (req/s)
331.40	387.77	284.67

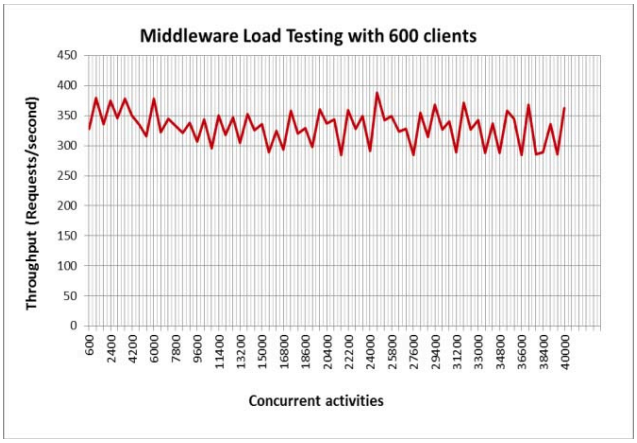

Figure 8. Load performance testing of the middleware

During the scalability testing, the error rate is observed to be zero percent for the concurrent HTTP requests ranging from 600 to 40000. This shows that the cloud environment for hosting the middleware is highly available. The performance of the middleware in the test environment shows high throughput and support for higher number of users. The estimated number of healthcare professionals in the Geriatrics Ward who will be using *SOPHRA* is 200.

We used other metrics as well such as the read and write mixed requests to determine the inconsistency window of the system. This we found that it takes approximately 0.078s to see 600 newly created RESTful Web resources on all mobile participants and approximately 0.11s for 600 newly created SOAP services to be visible on all connected mobile participants.

V. RELATED WORK

A. WS* and REST

Web Services (WS) [7, 13] are network oriented applications that serve information based on standards such as SOA and REST. Furthermore, Web services can be used to deploy business services and applications on many platforms since WS are mostly XML based and use HTTP as a communication protocol [11, 15]. Recent studies show that deploying web services in a cloud computing environment can guarantee scalability and good system performance [6, 10, 15]. Pautasso et al. [13] compared WS*, which they described as "Big" Web Services (or SOA framework), with RESTful Web Services. Their paper concluded that the best design architecture depends on the needs of the developer since the two standards have different architectural design principles. In another finding, Beal [14] in his article, "Understanding Web Services" reports that Web services have shaped the paradigm of business communication between client nodes and servers. The Service-Oriented Architecture (SOA) framework provides support to various components of Web services to interoperate. According to Wicks et al. [11], SOA focuses on reusability of software and integration. Additionally, SOA supports packaging, which makes changing of older versions of software very fast and at minimal cost [11].

Apart from the SOA approach, other researches have explored the REST approach. *REpresentational State Transfer* (REST) is an architectural principle that uses the Web platform for distributed computing [13, 15]. REST is better understood in the context of identifying everything as a resource, representation, and state. The design follows certain technological principles as described in [9]:

1) Identification of resources through URI: The key resources should be given Universal Resource Identifiers (URIs) which will facilitate interactions within the system.

2) Uniform interface: Resources can be manipulated through representations using HTTP methods such as GET, HEAD, POST, PUT, DELETE, OPTION, TRACE, PATCH and CONNECT.

3) Self-descriptive messages: Since resources are decoupled from their representation, it makes content accessibility very simple regardless of the format of the resource content [13].

4) Stateless interactions: While resources have states, their interactions should be kept stateless.

5) Hypermedia as the engine of application state (HATEOAS): In order to navigate between resources, URIs such as hypertext can be used in a resource representation. HATEOAS aids the client to know the next steps to take since the returned URI contains links to available options.

B. Mobile Provisioning of Web Services

It is surprising that as at now there are only a handful of studies on mobile hosting and Web services provisioning. Srirama et al. [7] are one of the innovative researchers on facilitating the mobile device as a provider of Web services. In their initial studies, they demonstrated the feasibility of hosting Web services on the mobile device by adopting the SOAP messaging approach. The paper resolved the IP addressing challenges in mobile peer-to-peer communications by adopting two approaches: High-Speed Circuit Switched Data (HSCSD) dial-up connection and General Packet Radio Services (GPRS) environments. Later, the authors extended their work in [5] to facilitate the mobile provider to handle SOAP requests over HTTP. In addition, the mobile provider was enabled to handle concurrent requests, and support runtime services deployment.

Also, Meads et al. [4] employ the cloud based middleware technique to provide a communication interface for ubiquitous devices to communicate with mobile providers in heterogeneous networks. The paper concludes that mobile providers can be reached via Bluetooth or Wi-Fi, an approach that gives requesters the flexibility to explore

different communication channels. Furthermore, Hassan et al [8] researched on managing the limited resources of the mobile provider and proposed a mobile web service partitioning scheme. As a result, complex business processes can be executed by the mobile provider with a backend super computer doing most of the high demanding computations. Additionally, Aijaz et al. [19] demonstrated the high performance of RESTful mobile Web services provisioning compared to SOAP Web services.

VI. CONCLUSION

This paper presents *SOPHRA*, a mobile hosting infrastructure for the Geriatrics Ward of the City Hospital, Saskatoon Canada. Medical records regarding patients are modeled as Web services following the REST and SOAP design principles. The use of open data standards such as XML for SOAP and JSON for REST in designing the WS aided SOPHRA to be integrated into the existing Health Information System.

Also, *SOPHRA* employs a middleware framework that is hosted on a private internal cloud platform. The middleware uses resources replication technique to ensure high information availability. In addition, it is encouraging to know that in a multi-node system, it takes segments of a second for all connected mobile participants to be updated.

The next version of *SOPHRA* which is our future work, will consider autonomic computing to ensure resilient and fault-tolerance design.

REFERENCES

[1] C. Gibbs, "The Rise of Tablets in the Enterprise," GigaOM Pro, June 2011.

[2] C. Warren, "Native App vs. Web App: Which Is Better for Mobile Commerce?," http://mashable.com/2011/05/23/mobile-commerce-apps/

[3] PhoneGapWiki. Available: http://wiki.phonegap.com/w/page/33313613/Changelog WebView

[4] A. Meads, A. Roughton , I. Warren, and T. Weerasinghe, "Mobile Service Provisioning Middleware for Multihomed Devices," Proceeding WIMOB '09, Networking and Communications IEEE Computer Society Washington, DC, USA 2009.

[5] S.N. Srirama, M. Jarke, and W. Prinz, "Mobile Web Service Provisioning. Telecommunications," (AICT-ICIW '06), International Conference on Internet and Web Applications and Services/Advanced International Conference on 19-25 Feb. 2006.

[6] C. Barnatt, "Explaining Cloud Computing" [Online], 10th May 2009, Available: http://www.explainingcomputers.com./cloud.html.

[7] S.N. Srirama, M. Jarke, and W. Prinz, "Mobile Host: A feasibility analysis of mobile Web Service provisioning," 4th International Workshop on Ubiquitous Mobile Information and Collaboration Systems, UMICS 2006, a CAiSE'06 workshop, June 5-6th, 2006.

[8] M. Hassan, Z. Weiliang, and Y. Yang, "Provisioning Web Services from Resource Constrained Mobile Devices," Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on 5-10 July 2010.

[9] X. Feng, J. Shen, and Y. Fan, "REST : An Alternative to RPC for Web Services Architecture," First International Conference on Future Information Networks, ICFIN 2009, p 7-10, 2009.

[10] J. Christensen, "Using RESTful web-services and cloud computing to create next generation mobile applications," Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA, p 627-633, 2009, OOPSLA 2009 .

[11] G. Wicks, E. Van Aerschot, O. Badreddin, K. Kubein, K. Lo, and D. Steele, "Powering SOA Solutions with IMS," Pg. 9 Publisher: IBM Redbooks Pub., Date: March 30, 2009, Part Number: SG24-7662-00, Pages in Print Edition: 410.

[12] M. Fowler, "Richardson Maturity Model: steps toward the glory of REST," March 2010, Available: http://martinfowler.com/articles/richardsonMaturityModel.html

[13] C. Pautasso, Z. Olaf, and F. Leymann, "RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision," Proceeding of the 17th International Conference on World Wide Web 2008, WWW'08, p 805-814, 2008, Proceeding of the 17th International Conference on World Wide Web 2008, WWW'08

[14] V. Beal, "Understanding Web Services" September 2010. Available: http://www.webopedia.com/DidYouKnow/Computer_Science/2005/web_services.asp

[15] Q. Wang, R. Deters, "SOA's Last Mile-Connecting Smartphones to the Service Cloud," cloud, pp.80-87, 2009 IEEE International Conference on Cloud Computing, 2009

[16] P. Farley, and M. Capp, "Mobile Web Services," Published in: BT Technology Journal archive, Volume 23 Issue 3, July 2005, Kluwer Academic Publishers Hingham, MA, USA.

[17] C. Monash, "DBMS2: Read-your-writes (RYW), aka immediate, consistency," A Monash Research Publication, May 1, 2010. Available: http://www.dbms2.com/2010/05/01/ryw-read-your-writes-consistency/

[18] J. Pearce, "HTML5 and the Dawn of Rich Mobile Web Applications," InfoQ Sections: Development, Architecture & Design, Jun 24, 2011, Available: http://www.infoq.com/presentations/HTML5-Dawn-of-Rich-Mobile-Web-Applications

[19] F. Aijaz, S. Z. Ali, M. A. Chaudhary, and B. Walke, "Enabling High Performance Mobile Web Services Provisioning" , Published in: Vehicular Technology Conference Fall (VTC 2009-Fall), 2009 IEEE 70th, 20-23 Sept. 2009.

[20] J. Ranck, "The Rise of Mobile Health Apps," GigaOM Pro, October 2010.

[21] M. Rusu, G. Saplacan, G. Sebestyen, N. Todor, L. Krucz, and C. Lelutiu, " eHealth: Towards a Healthcare Service-Oriented Boundary-Less Infrastructure," Original Research:Applied Medical Informatics Vol. 27, No. 3/2010, pp: 1-14.