Middleware-Enabled Mobile Framework in mHealth

Richard K. Lomotey and Ralph Deters
Department of Computer Science
University of Saskatchewan
Saskatoon, Canada
richard.lomotey@usask.ca, deters@cs.usask.ca

Abstract— **The recent advancement in mobile technology has established smartphones and tablet devices as the consumer device nodes to access the Electronic Health Records (EHR). Mobile devices further aid the healthcare professionals to access the EHR on the go and outside a centralized health facility. However, the over reliance on wireless communication mediums (e.g., Wi-FI, and 3.5G/4G) by mobile devices hampers the reliable flow of disseminating the EHR. For instance, there is no guarantee that the medical data from the main Health Information System (HIS) can be consumed on the mobile device of a healthcare professional when there is no connectivity. While a secure caching technique of the medical data on the mobile can be a solution to facilitate offline accessibility, the same technique can lead to challenges of data conflict. Specifically, when the cached data is updated in an offline mode and that information has to be synchronized with the HIS. We investigate efficient means of disseminating the EHR in unreliable networks. Our mobile architectural design consists of mobile nodes, a cloud-hosted middleware and the HIS. The proposal of the middleware is to enforce provenance, services composition, and reliable synchronization of the medical data for faster dissemination. The preliminary evaluations of the proposed approaches show high performance boost in terms of latency optimization and reliability.**

Keywords- mHealth; REST; middleware; Electronic Health Record; publish/subscribe; Mobile Cloud Computing

I. INTRODUCTION

In this paper, we focused on how efficient medical data synchronization can be achieved in a mHealth domain. Recently, the health sector is extending the accessibility of the Electronic Health Record (EHR) to mobile devices to achieve greater value. The intersection of mobile computing and health service collaboration is known as mHealth [4]. This new paradigm offers enhanced patient-physician communication, remote and on premise timely healthcare delivery, quick decision making, referencing, and so on [5]. Also, with the adoption of mobile and cloud computing technologies, health services delivery is greatly minimizing cost [1-3].

But, extending the EHR to the mobile devices leads to new challenges. These devices rely on wireless mediums as their primary mode of communication; thus, there are chances that the physicians will experience intermittent loss of connectivity as they go about their daily activities. This situation hampers seamless accessibility to the medical data which is hosted on the Health Information System (HIS). In order to achieve offline accessibility, secure caching of the medical data on the mobile has been proposed in some of our previous works [4, 20]. However, caching leads to challenges with the medical data update management and synchronization. When a physician changes the state of a medical data on the mobile in an offline mode and the same data is changed on the HIS by another physician simultaneously, there are challenges with the determination of which record is the most updated. Also, synchronizing the updates when not done properly can lead to high consumption of the wireless bandwidth.

We seek to address the challenges of medical data synchronization in this work by proposing a cloud-hosted middleware. Further, provenance and REST Web services composition are employed to maintain audit trail of the changes and keeping the synchronization task transparent. The details of the proposed approaches are explained in later sections of the paper.

The remaining sections are organized as follows. Section II reviews some existing works within the mHealth domain, services and data synchronization, and mobile cloud computing. Section III expounds our architectural design and Section IV reports the evaluation of the framework in the real-world. The paper concludes in Section V with our contribution and future direction.

II. LITERATURE REVIEW

A. mHealth

E-health is defined by Eysenbach [6] as a study that employs information and communication technology mostly with the aid of the Internet, to support the safe delivery of healthcare services. This field is useful from the perspective of patients, healthcare delivery personnel, and healthcare administrators. Eysenbach also stated that e-health is more beneficial in terms of reducing cost of diagnosis and greater improvement in the communication between patients and doctors. The exponential rise in the use of mobile phones as reported by [7] motivated us to look into how the use of mobile devices such as smartphones and tablets will better

CFP13UCC-USB/13 $26.00 © 2013 IEEE
DOI 10.1109/UCC.2013.19

enhance the delivery of E-Health. This is described by the authors in [4, 17] as mHealth.

A lot of mHealth applications are being deployed in recent times but little is known about their architectural design. Some are standalone applications and others are distributed services. Standalone applications are good for guiding patient-specific activities. Some of these applications as highlighted in [7] include: the Radiology Assistant, Epocrates Rx, iAnasthesia, Unbound Medicine, Medequations, Skyscape's PDR and the Human Atlas.

However, standalone mobile applications don't fit into medical workflows which require continuous transactional updates. This creates the need to deploy distributed mobile health apps which can aid in the management of the Electronic Health Record. This latter need leads to critical issues for consideration such as the reliability, scalability, security, and real-time accessibility.

As the Electronic Health Records are hosted on backend Hospital Information Systems, the primary question is how the services can be exposed to the client consumers. Recently, two types of architectural designs from Web services have been embraced in the medical system design; SOA and ROA. We first investigate the applications that have been deployed following each of the design paradigm, their limitations in mHealth, and proposed solutions on enhancing performance.

B. Mobile Cloud Computing

There is also a growing demand for the integration of data and notification services in mobile application systems. Mobius [21] has been proposed to offer unified messaging and data serving for Mobile Apps. The researchers opined that current client−server architectures that are directly emulated in the mobile−cloud environment require consistent network interface. However, this requirement is a mirage since it can never be guaranteed in mobile networks. Hence, Mobius relies on client−side caching to offer offline services. Overall, Mobius achieves the following guarantees for the mobile developer community as reproduced from [21]:

- Reading/writing data and messaging
- Flexible real-time notification
- Disconnected operation handling for reading and writing
- Transparent, intelligent caching/pre−fetching
- Hosted, scalable backend service

RFS [22] has also been designed for sharing files in a mobile-cloud ecosystem. Understanding the unstable connectivity challenges in mobile networks, RFS aims at providing a cache component on the mobile end that is synchronized with the back−end cloud. The uniqueness of RFS is that the file synchronization process is adaptable to the particular network protocol. So, if the mobile establishes connection to the cloud over a weak network, the synchronization process follows a pre−fetching methodology. Pre−fetching means the history of the user requests are kept so when similar requests are issued later, the system automatically adds other requests that might have been issued in previous attempts even if these requests are not explicitly called by the user. RFS also has pre−pushing services which work similar to pre−fetching but in this case, the server pushes the files to the client nodes once the server knows those requests may be needed by the user based on the user's history without the user even making an attempt to request for the data. The combination of pre−fetching and pre−pushing techniques in RFS makes the system exchange files in real time with very minimal energy exhaustion. Before the deployment of RFS, Wukong was proposed by Mao et al. [23]. The Wukong platform equally relies on techniques such as: timestamp (to manage conflict between read/write operations applied to files in the mobile to cloud interaction), caching (to minimize the request between the mobile and cloud), file compression (to reduce the file size and manage the mobile storage space), and optional data encryption mechanisms.

Further, the instability of wireless networks has led to questions of data and application state consistency on the mobile. Burckhardt et al. [24] argue that eventual consistency methodologies should be adopted. For example, in order to facilitate access to application execution or data in offline mode (i.e., in the case of connectivity loss) mobile applications can be deployed to keep replicas or (mirrored data) in a local cache/storage. This locally cached data can be accessed in the offline mode or caching can support the user until connectivity is restored. Burckhardt et al. [24] further propose the employment of cloud types which is a way to ensure the same style of data and application formatting across the mobile and cloud components in order to enforce application level uniformity. The recognizable advantage of cloud types is that it ensures consistency at the schema level. This is a way to overcome the integration challenges of heterogeneous application services which are written in JSON, XML, HTML5, and so on. Also, the same authors put forward revision diagrams that can be followed to track changes to the application state. Revision diagrams aid in monitoring not just the stored application but the change set (or revision history) of the application when it was in offline mode. Thus, once connectivity is restored, the mobile application state will be synchronized with the server application state without any conflict.

C. Provenance

Data provenance (a.k.a., Data Lineage) is the ability to track the life-cycle of data from its creation through to its present stage. The data provenance mechanism facilitates derivation of data history which aids enterprises to enforce data transparency, accountability, security, privacy, audit trail, confidentiality, services authenticity, and so on [13].

The adoption of cloud services further fuel the debate on understanding the granularity of services security. By adopting provenance as a security measure, Zhang et al. [12] argue that cloud data and information leakage can be prevented through transparent and accountable event management. Hence, the authors focused on tracking the atomic operation on files in a cloud ecosystem such as: creating a file, deleting a file, writing to a file, reading from a file, moving a file, and renaming a file. But, the main

contribution of the authors work is the rule-based adoption for the provenance data traceability. The authors opined that data leakage can happen within the same domain (locally) or across domains. The local rule identifies activities such as file copying, file renaming, and file movements. The cross-domain rules however, track activities such as sender-side logic on the atomic operations, receiver-side logic on atomic operations, and email attachments.

D. Web Services

The Service-Oriented Architecture (SOA) is a Web protocol design that supports the creation of heterogeneous services. The SOA framework provides support to various components of web services to interoperate. According to Wicks et al. [8], SOA focuses on reusability of software and integration. Another key thing they report about SOA is pack-aging, which makes changing of legacy software very fast and at minimal cost.

Even though SOA helped web services developers to overcome interoperability problems, it has some notable limitations. SOA passes a lot of XML since it uses Simple Object Access Protocol (SOAP) and this makes simple communication between system components difficult. In most cases, to achieve cross platform interoperability between web services, a lot of standards have to be followed [9]. Lee et al. [10] report the challenges faced by Telco with issues on scalability due to earlier SOA design to build transactions. Other challenges identified with SOA design were poor system performance under heavy workload and adaptability. As a solution to address the challenges in SOA, they provided the ROA approach. The Resource-Oriented Architecture (ROA) as presented by Overdick [14] follows the guidelines of the REpresentational Sate Transfer (REST). Xu et al. [15] in their work used REST to build ROA system and compared the results with Service-based web services. ROA is more feasible in building business processes because of the use of hyperlinks that enable processes to communicate. The use of uniform interface (HTTP methods) exposes resources to be managed easily than in SOA where service components have to be created at every operation. Xu et al. also proved that ROA business transactions allow process visibility. This is good especially in the sense that the client consumer can communicate with the server periodically for updates and the vice versa. Selonen et al. [16] report building a lightweight architecture style for building ROA as a solution to their Mixed Reality project. This helped them to achieve their aim of storing, retrieving, and managing interlinked content which otherwise couldn't be successful with SOA. Specific to the mHealth domain, our previous work in [4] details the architectural design of SOPHRA which is a mobile hosting infrastructure. The evaluation of the SOPHRA project shows high services scalability in the mobile environment using REST over SOAP.

E. Use Case and Research Focus

The major research question that we seek to address is how we can reliably manage the states of conflicting medical updates. To further clarify the problem to the reader, the under-listed use cases are provided.

Case 1: Assuming the physician is gone to attend to a patient outside the health facility and there is no connectivity; and the physician updates the medical data of the patient in an offline mode. How do we push and synchronize the updated information with the main HIS?

Case 2: Assuming the physician is disconnected and another physician updated a particular medical data, how do we push the updated information to the disconnected physician when s/he re-connects? And how is the synchronization achieved on the mobile?

Case 3: Assuming physician A is disconnected and in an offline mode, updates medical data D to D^{i}. Then physician B updates the medical data D which is on the HIS to D^{ii}. Later when physician A re-connects, how do we reconcile the updates of D? Which of the updates is kept?

Case 4: What if there were multiple, concurrent, pending updates? How would the system reconcile this?

To answer this questions, some of the reviewed concepts will be adapted on mobile cloud computing and provenance. In the next section, we detail the architectural design of the proposed mHealth environment.

III. The Architectural Design and Implementation

An architectural design is put forward (as illustrated in Figure 1) that focuses on solving the challenges in the above highlighted scenarios. The architecture is an extension on the previously deployed Med App [20]. A middleware layer which is hosted on the cloud is proposed to handle the communication between the mobile devices and the main HIS. Thus, the architecture is a three-tier system that comprises the moble devices plus the physicians, the middleware, and the main HIS.

The physicians (represented as HP) are able to communicate remotely to the middleware in a Wi-Fi network or over 3.5/4G. The details of each sub component are discussed below (some parts reproduced from [20]).

A. The Mobile Clients (Using the Med App)

Since the contribution of this paper is medical data synchronization, we just used the mobile-side application of Med App [20]. The consumer devices are heterogeneous because the healthcare professionals own different consumer devices. In view of the user device preferences, we chose to support varied mobile platforms such as iOS (i.e., iPad and iPhone), Android devices, Windows Phone 7 and 8, and BlackBerry. Also, some users will prefer to use their laptops or notebooks to access the same application so our initial decision is to come up with an implementation plan that will consider the various consumer platforms.

The mobile end of the application is made up of two major components; the User Interface (UI) and the Application Layer. The UI facilitates the healthcare professionals to interact with the application through graphical designs. The UI is written in HTML5, a Web framework called jQT [18], and CSS.

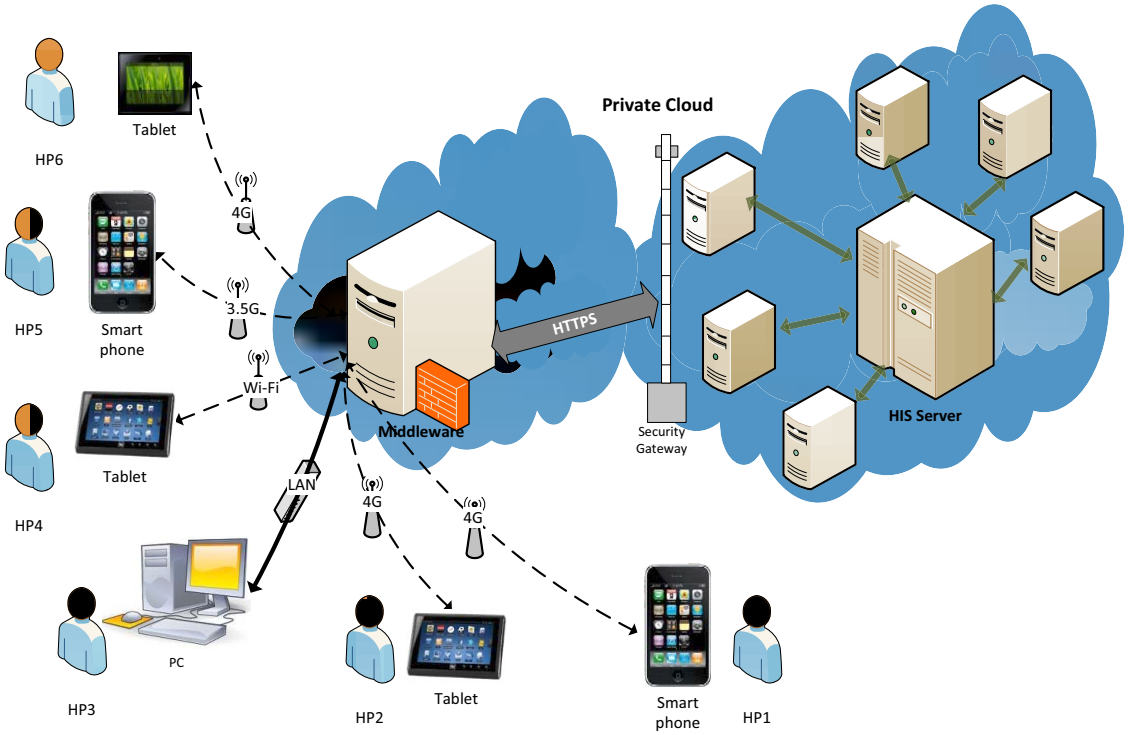

Figure 1. The architectural overview

The three Web technologies are combined in the design of the UI so that we can maintain a consistent clinical visual grammar of the Med App application across the various devices. While the UI facilitates interactivity, the application layer is responsible for the execution of the medical business logic on the client side. The application layer connects with the UI through JavaScript libraries which are built to provide lower-level services.

First, the healthcare professionals have to authenticate themselves on the mobile side before they can have access to the screen as shown in Fig. 2. The mobile side authentication is handled by the Client Security module. The module is proposed to provide offline security and prevent non-authorized users from accessing the service. In an offline mode, the mobile cannot authenticate over the network security gateway of the middleware so the authentication is handled by the client security for a single user. This means, only the authorized user of the device can have access to the mobile health record. We implement a data hoarding mechanism on the mobile side where all the mobile health records are stored in order to enable seamless access to the data even when connectivity is loss. This is in response to one of our main research questions as specified in the previous section. The healthcare professionals can also update existing medical records or create new medical records when the device is offline. The moment connectivity is restored, the updated medical records will be synchronized with the back-end database. The synchronization is aided through the publish-subscribe technique which will be discussed in the next section. The client side data hoarding is implemented directly on the mobile storage (HDD). Also, the application layer has a server interface which facilitates the communication between the mobile and the middleware-layer. We implement three server interfaces which relies on HTTP (using the XMLHttpRequest class in JavaScript), AJAX (which technically is also XMLHttpRequest but we adopt the Jquery [11] function calls), and WebSocket connection. All the activities of the application layer such as interacting with the mobile health records and the server requests are coordinated by the Web Workers module. The module is implemented to support concurrency of request processing on the mobile which aids the completion of services tasks in real-time.

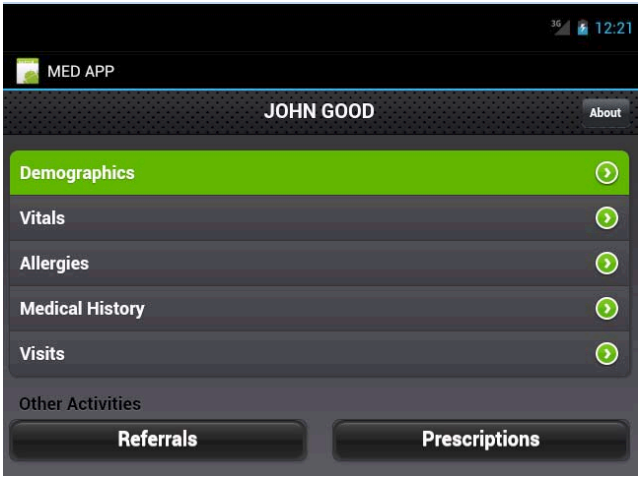

Figure 2. The UI of Med App (with dummy patient name)

B. *The Middleware*

The middleware is an extension of the Med App version which is discussed in [20]. The middleware is developed in the Erlang [19] programing environment because we aimed to achieve fault-tolerance through hot swapping. The various components of the middleware are discussed below as illustrated in Fig. 3:

Service Level: The medical system is a typical workflow management system with a lot of complexities. Some of these complexities are due to functional requirements and the rest non-functional. The medical data from the various units and departments are modeled as Web services. We refer to these unit-based medical data as *service sources*. The middleware interacts with the Service Level over an *HTTP Interface* within a LAN environment.

Protocol Transformation: The HIS supports two types of Web services standards; *SOAP* and *REST*. However, from our background findings and from our previous experience in [4], we transform all the SOAP-based services to REST. The REST standard has clear semantic on resources manipulation through HTTP methods such as POST (to create a new resource), GET (to fetch a resource), and PUT (to update a resource), and so on. Since EHRs can be seen as resources, ROA already provides the necessary verbs for manipulating EHR and the necessary concepts to uniformly link resources. Hence, adopting a REST is not a necessity, but the better choice among SOA and ROA, given the problem statement. As a result, services that are SOA-based are transformed to REST before delivered to the mobile node. After, the protocol transformation, services composition is enforced where all the distributed resources (medical data) is merged into a single view.

Storage Queue: This layer stores all the data under consideration including caches of the medical data from the HIS. *Provenance*: In the medical system, keeping audit trail is paramount. Audit trail aids in the determination of user activities as well as process execution transparency. We propose a provenance layer which keeps the record of every physician's activity regarding creating new data, accessing the data, and updating existing new medical data. However, the most critical task is keeping and managing the updated states of the data which will be discussed into detail in the next section. *Data Composition*: This layer controls the size of the provenance data including compression. But, the primary goal of this layer is to merge the provenance record of a single physician. Since the physicians are actively using the system, their provenance data is being generated and collected continuously. However, the records are stored at different times so there is the need to maintain a collective view of each physician's activities. Thus, the data composition layer organizes each physician's traces into a single view. As well, the *Conflict Manager* is responsible for the determination of medical records which are in existence and their state changes. When a conflict in a record is detected, the appropriate action is taken and the provenance layer is notified so that the actions are recorded. In this case, actions which are deemed wrong can be reverted later since the process execution is transparent. It is important to state that the provenance data is only visible to (or, accessible by) the system administrators and those who are permitted to carry out system audit.

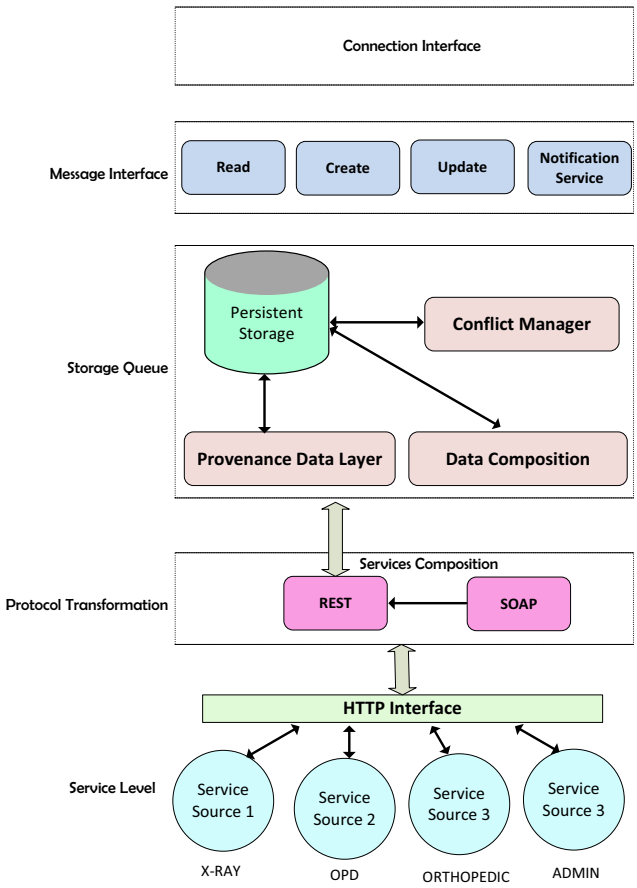

Figure 3. The design of the middleware stack

Message Interface: The message interface is where the various REST-based calls are tracked. The incoming request types such as (GET, POST, PUT) are clearly embedded in the HTTP request so the middleware is able to respond to the exact request need. Updates which are pushed to the middleware are propagated to other physicians through the Notification Service.

Connection Interface: This is the layer through which the middleware receives the incoming requests and also push out responses to the mobile nodes.

C. *The Update Management (Synchronization)*

In this section, a detailed sketch of how the proposed architecture deals with the challenges of data synchronization is presented. The middleware maintains a global clock of all interactions in the system. This ensures that every activity is timestamped. However, timestamps alone are not enough to determine updates in our use case. So, we focused on the E-tag values of the various resources as well since every medical data is assigned an E-tag value. This further strengthens the choice of REST. In Fig. 4, the medical data is consistent and synchronized on the mobile,

the middleware, and the HIS. Technically, the middleware is not caching the entire medical resource but the storage of only the meta-data of each medical record. This is to avoid duplication of data.

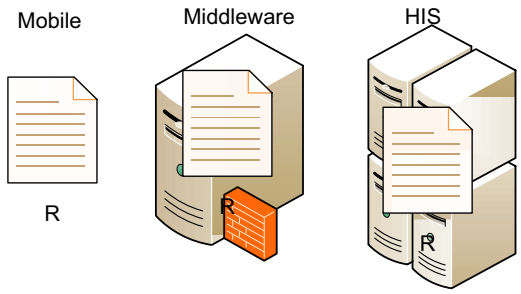

Figure 4. All records are synchronized

1) Use Case 1:

This is the case where the mobile version of the data is updated and the older record is still on the HIS and the middleware also keeps the meta-data of the older record as shown in Fig 5.

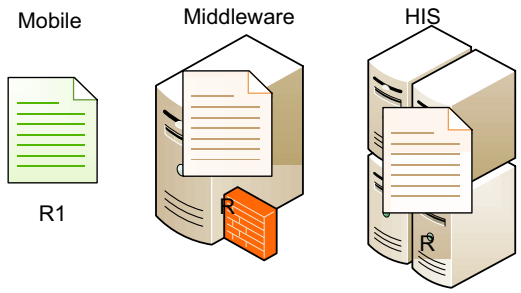

Figure 5. The mobile data is updated

When the physician reconnects to a network, the mobile side data is pushed to the middleware. The middleware will determine that the E-tag of R from the mobile has changed to R2 at a newer (or latter) time. This is achievable because the middleware will compare the E-tag of the incoming data with the version of the HIS and the middleware's meta-data; and, realize that the metadata and the HIS versions are equal but the mobile version has changed. Once this is established, the new state of the medical data (i.e., R1) will be pushed to the HIS and the middleware will update its meta-data with the new information.

2) Use Case 2:

This case involves the changing of a medical record, R, to R1 on the back-end layers while the physician was disconnected (illustrated in Fig. 6). When the mobile re-connects, the new state will be pushed to the mobile and the version R will be updated to R1. The synchronization is facilitated through the publish/subscribe and mirroring mechanisms. Firstly, as the physician is offline and the data is updated, the middleware will queue all the updates in a channel. The moment connectivity is established, the channel queue is pushed to the mobile. The incoming record will override all existing records on the mobile in order to maintain mirrored copies of what is on the HIS (i.e., specific data accessible by a particular physician but not the entire data on the HIS). This mechanism is only employed when the mobile has no new record.

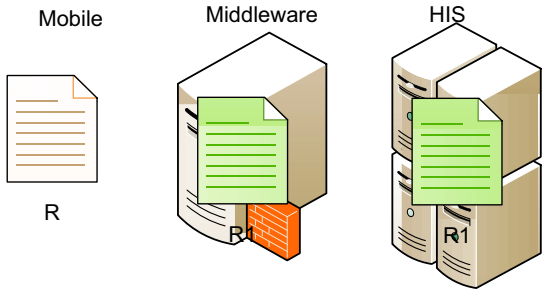

Figure 6. The back-end is updated

3) Use Case 3:

In this scenario, the medical record is updated to R1 by the physician in an offline mode; then, the same record is updated on the HIS by another physician to R2 (Fig. 7).

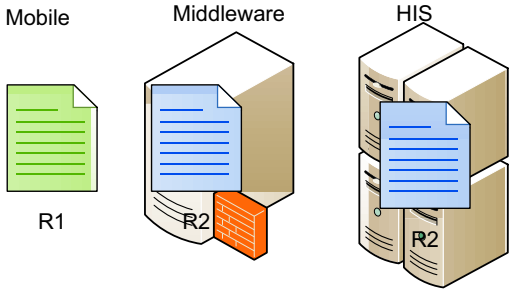

Figure 7. The mobile record and the HIS are updated

Though not mentioned earlier, all the synchronization follows the weak consistency approach where all updates are synchronized within some time frame. Anytime a physician is disconnected and updates are available on the HIS, those updates are queued in a channel to be delivered whenever the mobile re-connects (as in case 2). So, when the connection is established between the mobile and the middleware, the latter first receives the mobile side data and treats it as use case 1. The middleware realizes from the provenance storage that a medical record is changed to a new state R1 so it will try to update the old version since priority is given to the incoming mobile data. Then, the middleware will again determine that R is updated to R2 since it is queued in the pub/sub channel to be delivered.

Specific to our domain, an update of a record is not DELETING. Rather, it is making changes to the existing record through append. This means when a record R contains information A and this information is updated to B, the new record will contain AB for the purposes of transparency. So, when the middleware determines that R2 is in the channel

queue, it will append the mobile version to R2 to change the entire record to say R3 and push the new state of R3 to the HIS to replace R2; and, push the same R3 to the mobile to replace R1.

4) Use Case 4:

In this scenario, the medical record R is updated by multiple physicians when they were offline and probably, the record on the HIS is also updated by multiple physicians. Since the data is not deleted or over-written on the HIS but follows the append style, this use case is just an extension on the previous. Concurrency is handled on the HIS which is outside the scope of our work since the HIS is managed by the IT staff of the health facility. Further, every physician has a channel to publish and receive new updates. So, for the record R which is updated by multiple users, the middleware will keep updating the queue with the incoming updates from a single user perspective. Hence, the record R will be updated by the middleware with physician X data and push the new state back to physician X. Then physician Y's data will arrive and the middleware will update the meta-data and the record on the HIS and then put the new state back to Y. But, in this case, physician X will enter use case 2 scenario so the middleware will push the new state to Y as well. This will continue until all the information is synchronized.

IV. EVALUATION

A pilot testing of the implemented system is conducted using the following device: *Asus Transformer Prime* — OS: Android 4.0, Processor: 1.30 GHz Quad Core, Storage: 32GB, RAM: 1GB. The middleware is deployed on a privately owned cloud with the following specifications: *Processor: Intel Core i-5, CPU 2400@ 3.10 GHz 3.10 GHz, RAM: 16 GB, System 64-bit operating system.* The mobile devices connect to the middleware through 802.11g Wi-Fi 54Mbps connection.

- *Payload Transfer Time*

In Table I, we report the result of the data payload transfer from the mobile node to the middleware. The reported time also includes the time for the middleware to respond to the mobile.

TABLE I. PAYLOAD TRANSFER DURATION

Payload Size (MB)	Maximum (ms)	Minimum (ms)	Average (ms)
10	303.98	70	197.98
20	347.39	80	206.25
30	407.97	81	241.62
40	496.47	86	264.73
50	611.26	101	267.86
60	720.69	102	387.48
70	827.11	128	400.53
80	982.37	137	404.24
90	1041.40	159	502.56
100	1359.45	186	537.19
110	1561.43	226	603.12

- *Services Processing on the Middleware*

In the next experiment, the processing capacity of the middleware is evaluated based on the read/write request. This experiment follows scenario (i.e., use case 3) where updates are coming from both the mobile node and the HIS. The result is summarized in Table II.

TABLE II. READ/WRITE REQUEST PROCESSING TIME

Reads	Writes	Maximum (ms)	Minimum (ms)	Average (ms)
1000	0	49.94	20.23	32.57
900	100	97.11	27.48	46.18
800	200	109.80	35.36	58.01
700	300	159.31	57.98	92.86
600	400	183.30	61.72	98.65
500	500	217.62	85.32	133.11
400	600	251.23	84.84	137.32
300	700	253.65	101.32	160.08
200	800	262.23	104.06	163.95
100	900	327.46	133.04	209.19
0	1000	377.23	162.51	233.44

In the above experiment, the error rate of processing is 0.001% and the average payload size is 1 MB. For each request level, 1000 requests are considered and shared between read and write operations. From the table, it is clear that when there are a lot of write operations, the processing time increases.

- *Scalability Testing of the Middleware*

The third experiment focuses on determining the performance characteristics of the middleware. The primary goal is to determine how the middleware is able to scale in view of the fact that it is responsible for handling all the incoming and outgoing requests. Since the required users for scalability testing is huge, a simulation is employed. A HTTP load generator is designed in Erlang that emulates the behavior of 900 human users who can issue concurrent requests. These requests (which are 95% reads) range from 1000 to 9000. The result is graphed in Fig. 8.

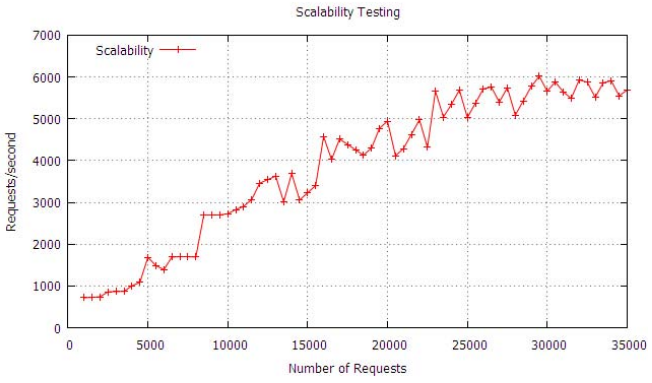

Figure 8. Scalability Testing of the Middleware

The overall average throughput is 3870.34 request/second. The distribution follows the exponential distribution of mean 0.23. From the graph, we can infer that the throughput stabilizes after 23000 requests. This means there are chances that as the number of requests keeps growing, the processing time will be increasing. However, the throughput value from 23000 is encouraging in the simulation environment because not that many users are expected in the real-world scenario in our case.

V. CONCLUSION

The era of employing mobile computing in collaboration with health-related equipments (known as mHealth) is gaining significant research attention. Especially, physicians are facilitated to access the Electronic Health Record on the go using their smartphones and tablet devices. The challenge however is that, the medical data is accessed over wireless mediums and when the physicians are moving from one location to another, there are chances of losing connectivity. This can lead to undesired situations such as denial of service, and inability to access up to date information.

One of the main ways to keep accessing the electronic health record on the mobile in a situation of network loss is secure caching on the mobile. But, caching also poses its challenges when it comes to synchronizing the medical data that is updated when the device is offline with the existing record on the Health Information System.

Our work therefore proposes a cloud-hosted middleware that facilitates the synchronization of medical data between the mobile node and the HIS. The medical data is designed following the REST standard so the parsing between the various components is found efficient. However, the current state of the work leads to more open questions. For instance, we are able to achieve reliable data synchronization because the data update follows the append style. In situations when the file is completely overwritten by a new update, synchronization cannot be achieved using our current proposal and methodology. Thus, the future work will focus on this latter scenario though it will be explored outside of the health domain.

REFERENCES

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," Future Generation Computer Systems, Volume 25, Issue 6, June 2009, Pages 599-616.

[2] H. Li, J. Sedayao, J. Hahn-Steichen, E. Jimison, C. Spence, and S. Chahal, "Developing an Enterprise Cloud Computing Strategy," Intel White Paper, January 2009.

[3] H. E. Schaffer, "X as a Service, Cloud Computing, and the Need for Good Judgment," IT Professional , vol.11, no.5, pp.4-5, Sept.-Oct. 2009, doi: 10.1109/MITP.2009.112.

[4] R. K. Lomotey, S. Jamal, and R. Deters, "SOPHRA: A Mobile Web Services Hosting Infrastructure in mHealth," Mobile Services (MS), 2012 IEEE First International Conference, vol., no., pp.88-95, 24-29 June 2012 doi: 10.1109/MobServ.2012.14

[5] R-G. Jahns, "mHealth apps: 8 reasons why it matters for Pharma", 30 November 2010, Accessed last: http://www.research2guidance.com/mhealth-apps-8-reasons-why-it-matters-for-pharma/

[6] G. Eysenbach, "What is e-health?" J Med Internet Res 2001;3(2):e20. 2003 Regional Office for the Eastern Mediterranean - World Health Organization. http://www.jmir.org/2001/2/e20/

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

[8] 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.

[9] Exforsys Inc. SOA Disadvantages. SOA Tutorial. http://www.exforsys.com/tutorials/soa/soa-disadvantages.html

[10] W. Lee, C. M. Lee, J. W. Lee, and J. Sohn, "ROA Based Web Service Provisioning Methodology for Telco and its Implementation," Proceedings 12th Asia-Pacific Network Operations and Management Symposium, APNOMS 2009, p 511-14, 2009.

[11] http://jquery.com/

[12] Zhang, O.Q., Ko, R.K.L., Kirchberg, M., Suen, C. H., Jagadpramana, P., Lee, B. S. 2012. How to Track Your Data: Rule-Based Data Provenance Tracing Algorithms. Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on , vol., no., pp.1429-1437, 25-27 June 2012

[13] Cheney, J., Chong, S., Foster, N., Seltzer, M., and Vansummeren, S. 2009. Provenance: a future history. In Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications (OOPSLA '09). ACM, New York, NY, USA, 957-964.

[14] H. Overdick, "The Resource-Oriented Architecture," IEEE Congress on Services, p 340-7, 2007.

[15] X. Xu, L. Zhu, Y. Liu, and M. Staples, "Resource-Oriented Architecture for Business Processes," 15th Asia-Pacific Software Engineering Conference, p 395-402, 2008.

[16] P. Selonen, P. Belimpasakis, and Y. Yu, "Experiences in Building a RESTful Mixed Reality Web Service Platform," Web Engineering. Proceedings 10th International Conference, ICWE 2010, p 400-14, 2010

[17] Vital Wave Consulting. mHealth for Development: The Opportunity of Mobile Technology for Healthcare in the Developing World. February 2009. http://www.vitalwaveconsulting.com/pdf/mHealth.pdf

[18] jQT, http://jqtjs.com/

[19] Erlang Programming Language, http://www.erlang.org/

[20] Lomotey, R. K., and Deters, R. 2013. Efficient Mobile Services Consumption in mHealth. Proc. of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013), and the International Symposium on Network Enabled Health Informatics, Biomedicine and Bioinformatics (HI-BI-BI 2013), pp:8 pages, Niagara Falls, Canada, August 25-28, 2013.

[21] Chun, B. G., Curino, C., Sears, R., Shraer, A., Madden, A., and Ramakrishnan, R. 2012. Mobius: unified messaging and data serving for mobile apps. In Proceedings of the 10th international conference on Mobile systems, applications, and services (MobiSys '12). ACM, New York, NY, USA, 141-154.

[22] Dong, Y., Zhu, H., Peng, J., Wang, F., Mesnier, M. P., Wang, D., and Chan, S. C. 2011. RFS: a network file system for mobile devices and the cloud. SIGOPS Oper. Syst. Rev. 45, 1 (February 2011), 101-111.

[23] Mao, H., Xiao, N., Shi, W., and Lu, Y. 2010. Wukong: Toward a Cloud-Oriented File Service for Mobile Devices. In Proceedings of the 2010 IEEE International Conference on Services Computing (SCC '10). IEEE Computer Society, Washington, DC, USA, 498-505.

[24] Burckhardt, S., Fähndrich, M., Leijen, D., and Wood, B. P. 2012. Cloud types for eventual consistency. 2012. In Proceedings of the 26th European conference on Object-Oriented Programming (ECOOP'12), James Noble (Ed.). Springer-Verlag, Berlin, Heidelberg, 283-307.