Reliable Web Service Consumption Through Mobile Cloud Computing

Amr S. Abdelfattah, Tamer Abdelkader and
El-Sayed M. El-Horbaty

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.74461

Abstract

The mobile intermittent wireless connectivity limits the evolution of the mobile landscape. Achieving web service reliability results in low communication overhead and correct retrieval of the appropriate state response. In this chapter, we discuss and analyze two approaches based on middleware approach, Reliable Service Architecture using Middleware (RSAM), and Reliable Approach using Middleware and WebSocket (RAMWS). These approaches achieve the reliability of web services consumed by mobile devices and propose an enhanced architecture that achieves the reliability under various conditions with minimum communication data overhead. In these experiments, we covered several cases to prove the achievement of reliability. Results also show that the request size was found to be constant, the response size is identical to the traditional architecture, and the increase in the consumption time was less than 5% with the different response sizes.

Keywords: web service, WebSocket, mobile consumption, middleware, timeout problem

1. Introduction

The evolution of the mobile landscape coupled with the Internet nature and the recent explosion of the cloud computing technology is facilitating the deployment of web services. The web services are the perfect way to provide a standard platform for mobile application communication through the Internet.

Smartphones are gradually becoming the effective client platform to consume the services and the pool of data and information [1].

The mobiles are uncomfortable because of their limited processing power and intermittent wireless connectivity. In terms of that, there is an uncertainty of whether the web service request was successfully received, was lost on the Internet before reaching the server, or was partially processed. In the case the application retries the operation and resends the request, it may be duplicated or cause an error, such as two orders entered or two credit card charges.

Most of the cloud-oriented services and data are deployed as web services that are network-oriented applications [2]. The synchronization between a mobile device and a web service is achieved through initiating a conversation in a request-response pattern.

Mobile cloud computing (MCC) is the combination of cloud computing, mobile computing, and wireless networking to bring rich computational resources to mobile users, network operators, as well as cloud computing providers. That enables the applications that could not run on the mobile device due to the mentioned constraints and can now be delivered to users of these devices [3].

The reliable web services through mobile cloud computing is achieved using approaches that focus on ensuring the request execution under the intermittent connectivity and service unavailability conditions, moreover ensuring the appropriate response according to the request state.

1.1. Web service types and limitations

The web service architecture (WSA) [4] proposed by W3 organization relies on a number of web standards, such as Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL), Extensible Markup Language (XML), and Universal Description Discovery and Integration (UDDI) that allow services to be searched, described, and integrated by any application [5].

There are two types of web services: Simple Object Access Protocol (SOAP) and Representational State Transfer (REST).

SOAP web service consumption requires extra effort mostly due to the lack of native support. The two major SOAP-based web service limitations are categorized mainly to communication and computation overhead [5].

In the REST architecture, standardized interface and protocol are used for representing resources between clients and servers. These principles encourage REST applications to be simple, be lightweight, and have high performance. Therefore, regarding the scope of reliability, RESTFUL services overcome SOAP service limitations and achieve better results especially in mobile communications [6–8]. Using REST as service architecture is preferred for mobile devices because REST services use HTTP request and response, which means that a mobile device connected with the Internet can access the service without additional overhead, unlike SOAP web services [9].

1.2. WebSocket

The WebSocket protocol is a full-duplex protocol over a TCP connection that typically provides bidirectional communication between web browsers and web servers. It is used to facilitate the real-time data transfer from and to the server [10].

1.3. Reliability and challenges

The challenges in web service consumption using mobile client are worth analyzed from two perspectives: mobile limitations and connectivity limitations from both mobile and cloud service provider sides. The challenges are listed in the following points:

- Connection loss: The smartphones perspective, clients have an intermittent connection because of their mobility. They can be momentarily removed from the connected network and later join the available network [9]. From the perspective of cloud/server, it may lose the connection, and their web services become unreachable from clients.
- Latency/bandwidth: The very limited bandwidth mobile cellular networks are often billed based on the data transferred amount.
- Timeout: The service timeout problem is one of the issues in the mobile experience, such that the service response size, database relations' communication time, and the required computations in the services are continuously increasing corresponding to the application usage during the time, which causes repeat timeout through the consumption of these services.

1.4. Reliable web service approaches

Middleware approach and mobile agent (MA) approach are the most recent approaches that are used for achieving the reliability of web services.

In this chapter, we will discuss, analyze, and focus on the middleware-based approaches.

2. Background

Different architectures are implemented based on the following the middleware approach. The middleware is applied in more than context with a different purpose.

2.1. Middleware approach

The middleware component is responsible for retrieving the response from the web service, such that it acts as a gateway that communicates lightly with the client.

The middleware solution for mobile clients mostly focuses on application and content adaptation. The proposed communication architecture introduces a gateway between the mobile client and the web service that takes the heavy load communication with the service. The mobile client will instead have to sustain a lightweight and simple client–server communication over a fast binary protocol [11].

a. Middleware architecture

The middleware architecture, as shown in **Figure 1**, consists of the following three components:

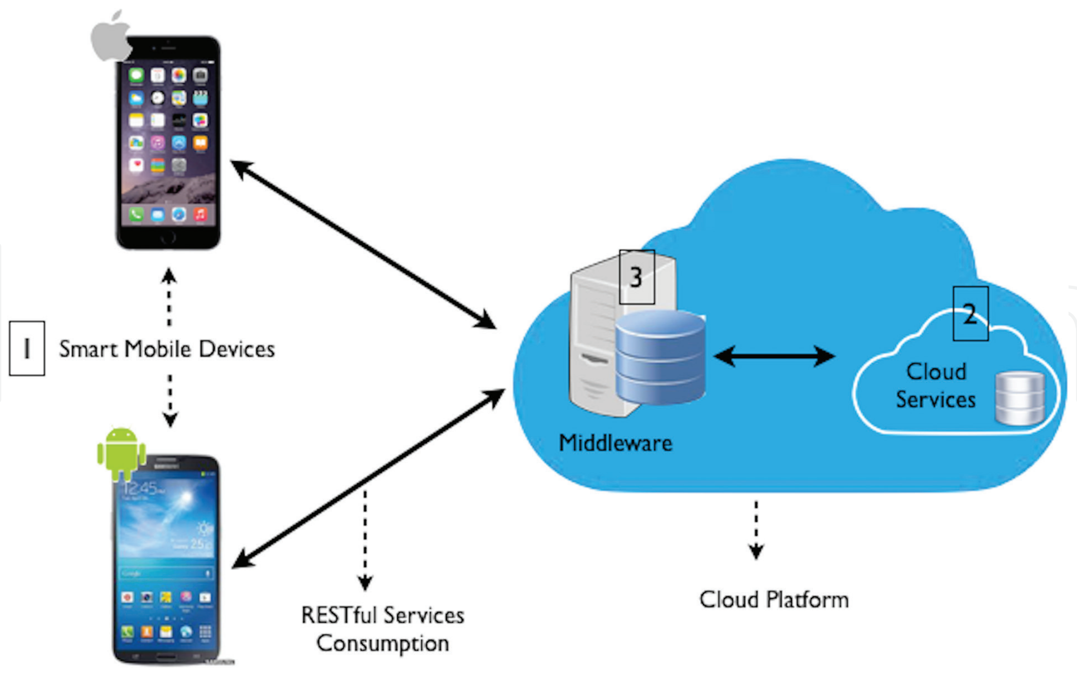

Figure 1. Middleware architecture and the communication process between its components.

1. A mobile device that has Internet access.
2. A web application that contains web services connected to a database.
3. A web application that represents the middleware contains web service consumption and data handling modules.

b. Middleware advantages

The middleware advantages are summarized as follows:

- Small bandwidth usage because of the light communication with the mobile client.
- This architecture brings more opportunities toward ensuring a more reliable communication with the web service such as:
  - The middleware will often run on dedicated hardware that will justify the search for solutions to ensure some kind of state of the communication with a web service.
  - Retry mechanisms can be explored in case of connection failures.
  - The middleware can retain the state of the overall communication and retry to continue when all parties come back online, such as in the case of communication failures (mobile device to middleware or middleware to web service).

c. Middleware limitations

The main middleware limitation is the increasing of the overall duration of a request to the web service and to process the data format, but deploying the middleware and the actual web service in the same server instance network can enhance this.

3. Reliable service architecture using middleware (RSAM)

This section describes the proposed architecture RSAM [12]. In this section, the proposed architecture is presented and how it is used to improve the web service consumption reliability through mobile cloud computing. It also illustrates how the proposed approach overcomes the web service consumption and their reliability limitations mentioned in the above sections.

The enhanced middleware architecture focuses on the integration between the mobile client and service layer, so the architecture is defined as client middleware component and service middleware component. This integration increases the system awareness for each request state to be notified to the user appropriately.

3.1. RSAM architecture

In this architecture, the mobile consumer component has the permission to consume a cloud service directly without passing through the middleware, which creates the flexibility to customize system communications.

The enhanced architecture components and their communications are shown in **Figure 2**, which contain two main components:

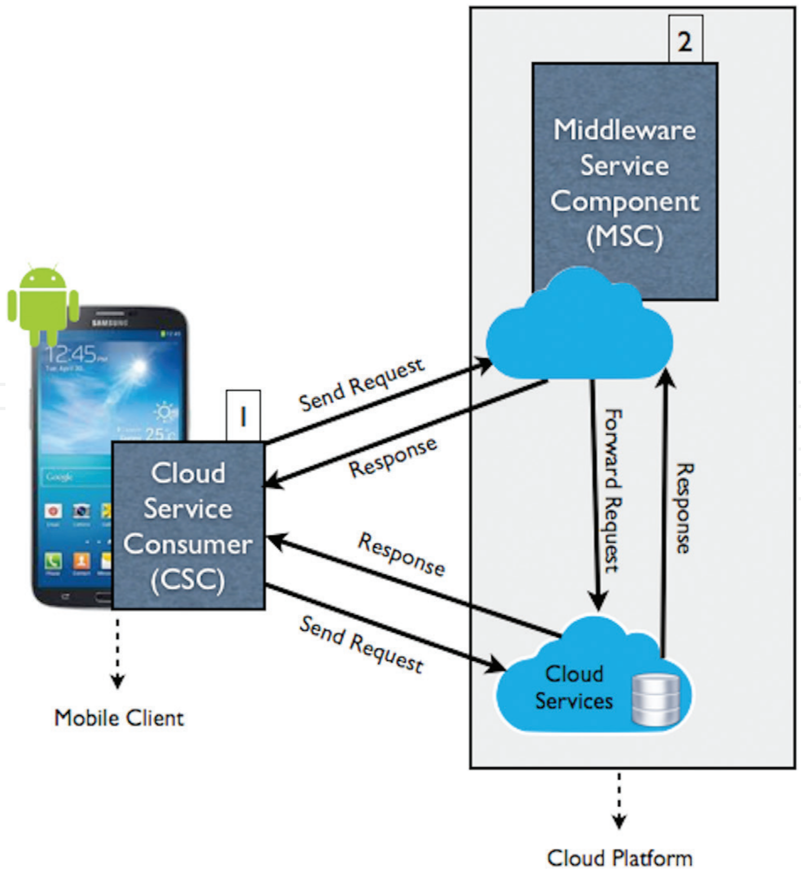

Figure 2. RSAM architecture and the communication process between its components.

- Cloud service consumer (CSC) is a mobile client middleware component responsible for:
  - Constructing the request with its appropriate attributes to be ready for sending
  - Handling the request communication cycle either to the cloud service directly or through the service middleware component
  - Receiving the response and notifying the client with the appropriate state
- Middleware service component (MSC) is a cloud service middleware component responsible for:
  - Receiving the request from the client service consumer and constructing the appropriate response regarding the sent request attributes.
  - Communicating with the required cloud service.
  - Caching the request and response states to be tracked for later usage.
  - A detailed description of the architecture is in the following subsections.

3.1.1. Cloud service consumer (CSC)

The CSC is the client middleware component responsible for handling the request communication cycle starting from the service call till the response notification.

There are attributes that affect the consumer and support it to track and handle the consumption process as a separate component, as shown in the following:

- Service descriptor is the module that contains the routing mechanism regarding the base cloud service URL and the middleware component URL and has the description of each service in the system. This descriptor consists of:
  - Basic attributes that describe the services, such as the service name, request type, parameter names, parameter types, and expected response type with its appropriate parser.
  - Semantic attributes that affect the behavior of the execution, such as:
    - The forced attributes that force the MSC to use the succeeded cached results or forward the request to the cloud service each time
    - The direct attribute that controls the request route to the middleware or to the cloud service directly
  - Request client Id is the unique identifier overall requests in the system; it consists of two main parts:
    - Basic part is the unique auto-generated key that consists of subparts that ensure its distinct property, such as mobile device manufacture number, the request timestamp, and the requested service name.

- Additional attributes part is the other part that includes labeled attributes that change the MSC behavior regarding the received request such as the request number of trials and forced attribute.

The sequence of the client service consumer is illustrated in **Figure 3**. The following algorithm highlights its steps:

1. [Mobile User] select a specific service in the application to invoke.
2. [Client Id generator] generate valid client id as designed in the middleware validation method.
3. [Service Descriptor] construct the request with its specific attributes.
4. [Service Consumer] send the request to the middleware or direct to the specified cloud service as attributed. [Service Client Manager] update the request attributes in the persistence storage based on the response state.
5. [Service Client Manager] adapt the response based on its state.
6. [Service Client Manager] notify the user with the appropriate response that shows the suitable state.

This sequence is performed in the mobile client part in two stages:

1. Preparation: The CSC prepares and constructs the appropriated request with its attributes, in order to be fit in the middleware request validation agreement.
2. Consumption: The stage of consuming the specific web service using the constructed request through middleware and receiving the appropriate response to show it to the user in its reliable form.

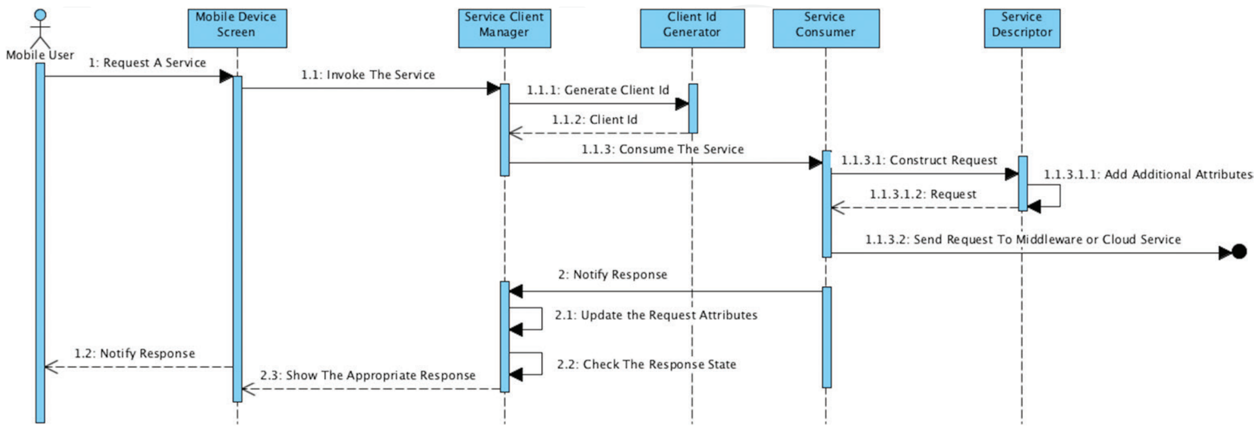
Mobile User
Mobile Device Screen
Service Client Manager
Client Id Generator
Service Consumer
Service Descriptor
1: Request A Service
1.1: Invoke The Service
1.1.1: Generate Client Id
1.1.2: Client Id
1.1.3: Consume The Service
1.1.3.1: Construct Request
1.1.3.1.1: Add Additional Attributes
1.1.3.1.2: Request
1.1.3.2: Send Request To Middleware or Cloud Service
2: Notify Response
2.1: Update the Request Attributes
2.2: Check The Response State
1.2: Notify Response
2.3: Show The Appropriate Response

Figure 3. Cloud service consumer sequence.

3.1.2. Middleware service component (MSC)

The cloud middleware component is the other integrated part with the cloud service consumer, such that it is responsible for handling the received request and proceeding to the appropriate function to send the response back to the client consumer.

The middleware component sequence is shown in **Figure 4**. It begins acting once a client request is received.

The sequence of the service middleware component is illustrated in **Figure 4**. The following algorithm highlights its steps:

1. [Mobile device] sends a request.
2. [Middleware request filter] receive the request to ensure it is included in the allowed ones.
3. [Request manager] initialize the request holder.
4. [Parser Manager] parse the request to extract the attributes included in the request.
5. [Validation Manager] ensure the request and the attached client id attribute validity.
6. [Database Manager] save the validated request in the persistence storage.
7. [Web service Consumer] consume the attached cloud service using the sent request.
8. [Database Manager] save the response in the persistence storage with a relation with the saved request.
9. [Request Filer] respond to the mobile device with the response.

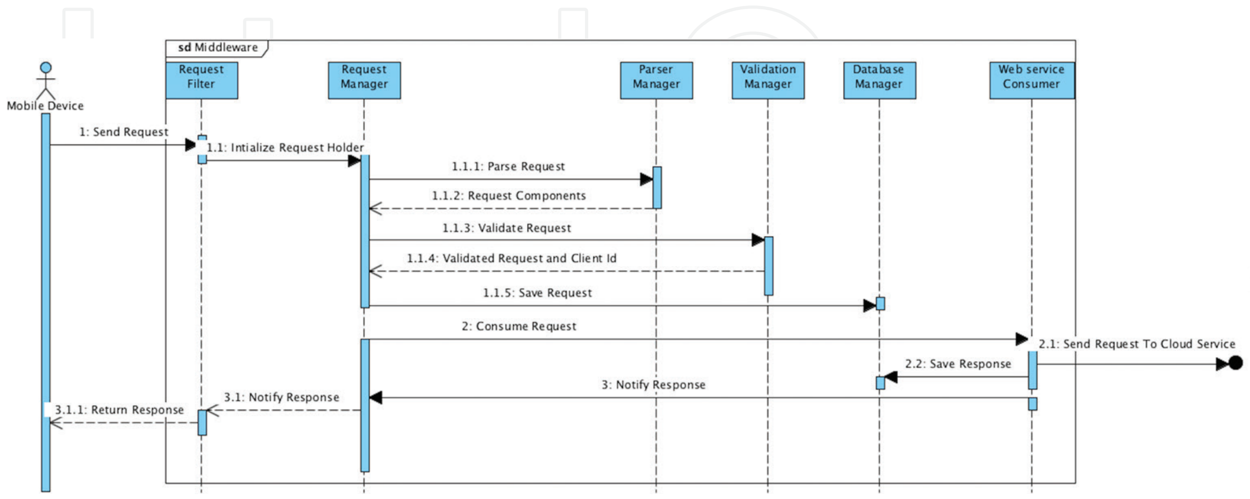

Figure 4. Service middleware component sequence.

This process follows three stages:

i. Pre-processing: The service middleware component parses the request to extract the client identifier and its attributes and then validates the request to ensure whether it is correct and secure or not.

ii. Processing: In the case of request validation success, the service middleware retrieves the similar previous cached request or saves this request details in its cache if it did not exist before; then there are two possible cases to follow regarding the different request cases:

a. Forward the request to the cloud service if:
   i. It was the first time invoking this request.
   ii. It is intended for the client consumer to retry this request.
   iii. The request is labeled with the forced attribute.

b. Return the middleware component cached results, if this request is invoked before with a successful result.

iii. Post-processing: The service middleware component constructs the appropriate response and caches its details according to the action taken in the processing step. It then sends this response to the mobile consumer to notify the user with the appropriate request state.

3.2. RSAM protocol

The enhanced architecture achieves cloud service reliability while considering the most affected cases illustrated in **Table 1**.

The architecture process flow shown in **Figure 5** covers these possible cases, such that it mainly depends on the middleware components in client consumer and middleware to track the state of each request to be able to notify the user with the appropriate response.

The possible response states shown in **Figure 5** contain the following:

- Succeeded and failed states that explicitly indicate the success and failure of the request execution.
- Cached state that informs the user that the response was cached in the middleware from previous execution for the same request. This state covers two cases:
  - Preventing the duplicated request execution
  - Optimizing the time of the request execution if it will get the same response from the cloud service because it no longer needs to be passed to the cloud service again with its complex query

  Doubt state counted as the most significant one that marks the request as doubt request to inform the user to contact the responsible one in order to ensure the request state to prevent its significant duplication

Mobile state	Middleware state	Cloud service state	Case number
Reachable	Reachable	Reachable	1
Reachable	Reachable	Non-reachable	2
Reachable	Reachable	Server error	3
Reachable	Non-reachable/server error	Any	4
Non-reachable	Any	Any	5
Timed out	Reachable	Reachable	6

Table 1. The covered state summary of enhanced middleware architecture.

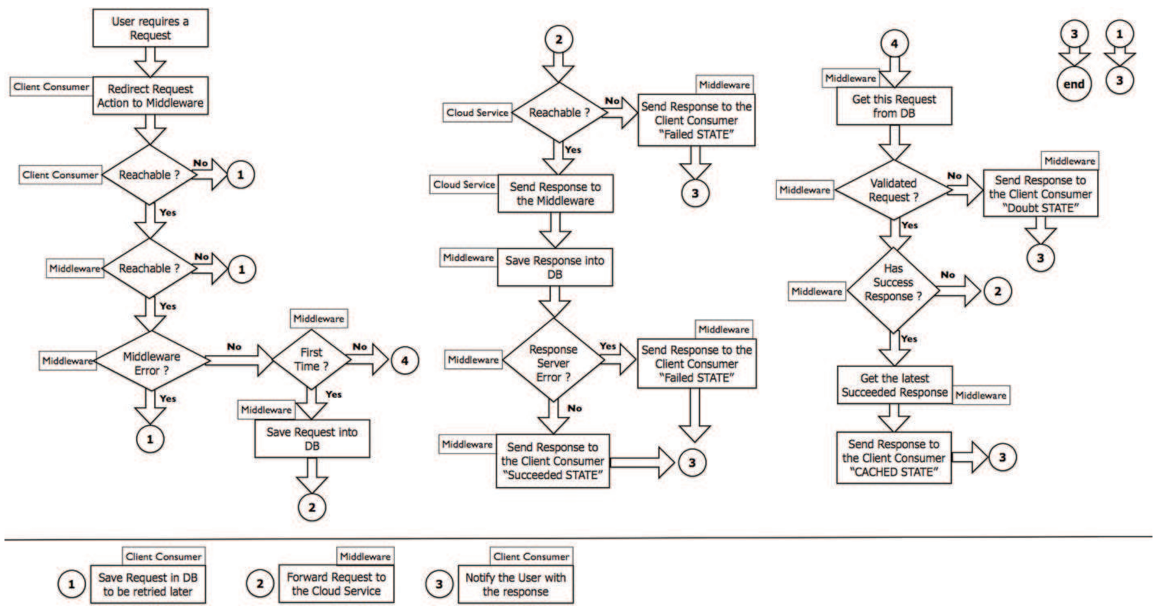
User requires a Request
Client Consumer
Redirect Request Action to Middleware
Client Consumer
Reachable ?
No
Yes
Middleware
Reachable ?
Middleware
Middleware Error ?
First Time ?
Save Request into DB
Cloud Service
Send Response to the Middleware
Save Response into DB
Response Server Error ?
Send Response to the Client Consumer "Failed STATE"
Send Response to the Client Consumer "Succeeded STATE"
Get this Request from DB
Validated Request ?
Send Response to the Client Consumer "Doubt STATE"
Has Success Response ?
Get the latest Succeeded Response
Send Response to the Client Consumer "CACHED STATE"
end
Save Request in DB to be retried later
Forward Request to the Cloud Service
Notify the User with the response

Figure 5. Enhanced architecture process.

Table 1 contains the following states:

- **Reachable**: available through Internet access.
- **Server Error**: The service responds with an internal server error.
- **Timed Out**: The Service execution takes more time than that allowed for the consumer to get the response.

4. Reliable approach using middleware and WebSocket (RAMWS)

This section describes the proposed architecture RAMWS. In this section the proposed architecture is presented and how it is used to achieve the reliable web service consumption in terms of overcoming the timeout problem. It achieves the immediate notification with the appropriate response once it is available to the user.

The enhanced approach depends on the integration between the middleware approach and the WebSocket protocol. This integration enables the middleware to always represent the client with its arguments while consuming the required web service and enables the WebSocket to represent the open connection protocol between the server and the client.

4.1. RAMWS architecture

The proposed approach is based on the architecture that was shown in **Figure 6**, which contains three main components:

1. Integrated socket cloud service consumer (SCSC) is a mobile client middleware component responsible for:
   - Constructing the request with its appropriate attributes to send it to the integrated socket middleware service component (SMSC) layer
   - Handling the request communication cycle between the client and the SMSC
   - Managing the client side WebSocket communication with the SMSC

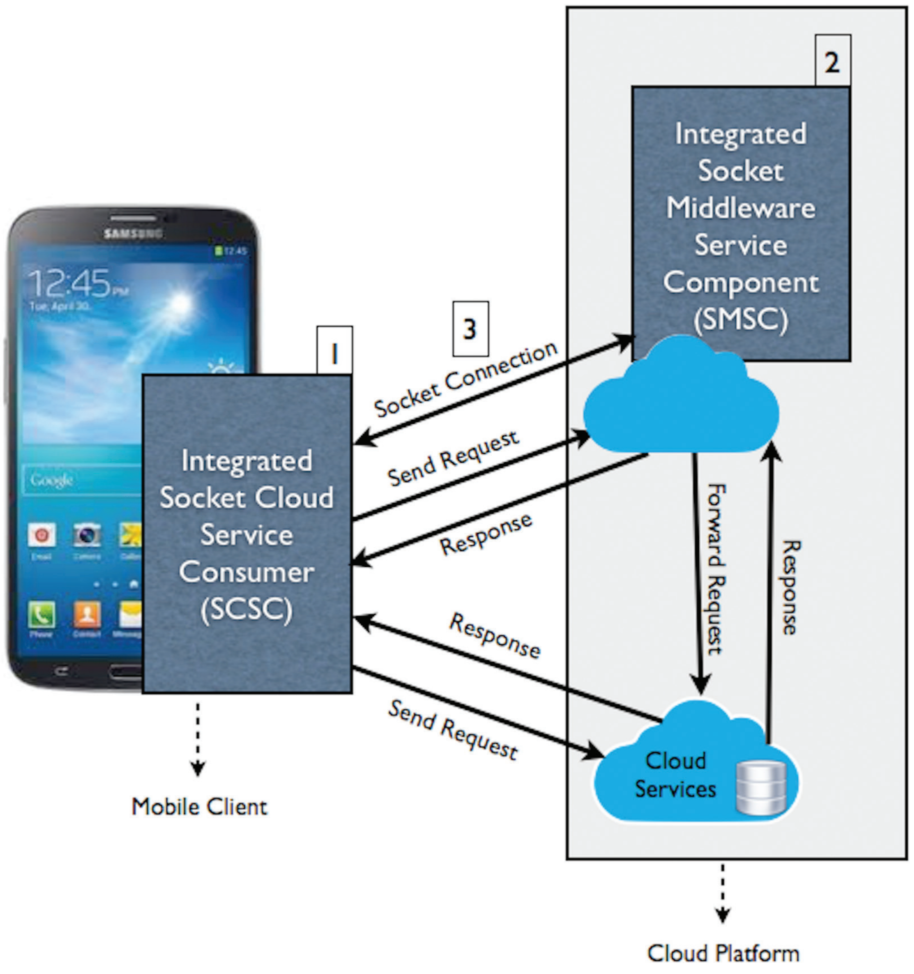

Figure 6. RAMWS architecture and the communication process between its components.

- Receiving the response and notifying the client with the appropriate result based on the service response and the WebSocket data connection

2. Integrated socket middleware service component (SMSC) is a cloud service middleware component responsible for:
   - Receiving the request from the integrated socket client service consumer (SCSC) and constructing the appropriate response regarding the sent request attributes
   - Consuming the required cloud service to get the requested response
   - Managing the server side WebSocket communication with the SCSC layer
   - Handling two-way response to the SCSC, such that short time response to the client request with an appropriate state and actual response through WebSocket connection contain the expected result from the cloud service when available
3. WebSocket protocol is a standard communication way for the server to send content to the client without being solicited by the client. It plays the most important role to overcome the request timeout problem, such that it allows messages to be passed back and forth while keeping the connection open during the service communication time, and it is responsible for:
   - Achieving the other way of communication between the SCSC and the SMSC besides the required REST service communication
   - Sending the actual response to the SCSC once this response returned from the cloud service

4.2. RAMWS protocol

The RAMWS architecture achieves reliability for the heavy cloud service while overcoming the timeout issue.

The architecture process sequence shown in **Figure 7** shows different experiences in mobile applications to prevent the heavy services from the timeout response, such that it mainly depends on two different responses: temporary response for the request and actual response through the socket connection.

The RAMWS protocol contains the following:

1. The request additional attributes: They are attributes that control some important behaviors in the service consumption process, such as:

a. request_id: Identifies the request, to distinguish the responses returned through the WebSocket connection from multiple requests.

b. temporary_response_type: The temporary response is the returned response to be shown in the mobile device till the actual response returns from the cloud service. There are some of the defined types that control the suitable temporary responses for each service, such as latest_cached_response, waiting_message, and limited_cached_response.

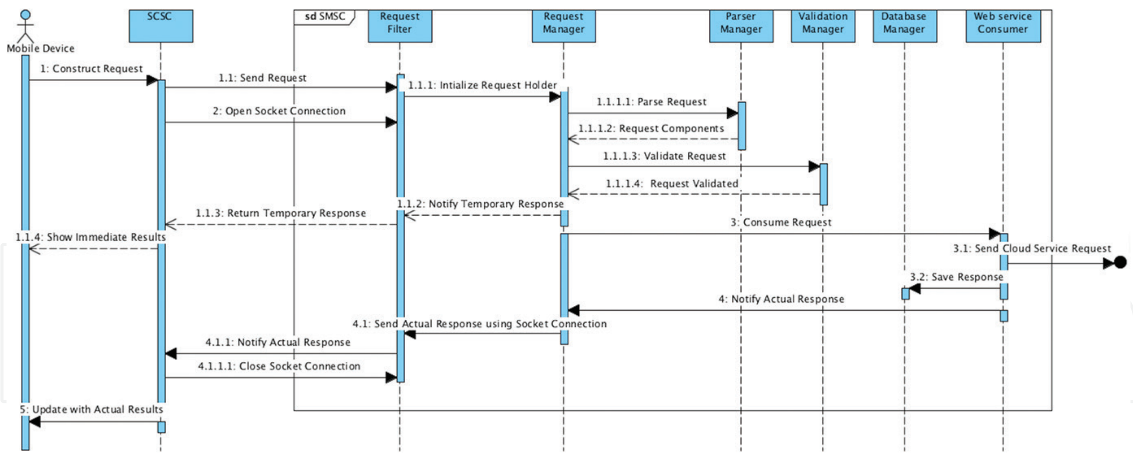
Mobile Device
SCSC
sd SMSC
Request Filter
Request Manager
Parser Manager
Validation Manager
Database Manager
Web service Consumer
1: Construct Request
1.1: Send Request
1.1.1: Intialize Request Holder
2: Open Socket Connection
1.1.1.1: Parse Request
1.1.1.2: Request Components
1.1.1.3: Validate Request
1.1.1.4: Request Validated
1.1.2: Notify Temporary Response
1.1.3: Return Temporary Response
3: Consume Request
1.1.4: Show Immediate Results
3.1: Send Cloud Service Request
3.2: Save Response
4: Notify Actual Response
4.1: Send Actual Response using Socket Connection
4.1.1: Notify Actual Response
4.1.1.1: Close Socket Connection
5: Update with Actual Results

Figure 7. RAMWS protocol sequence.

c. socket_enabled: The is a flag attribute, which flags if this service is heavy and requires this proposed approach, or it is light enough to get the response within the defined timeout without additional overhead

5. Implementation and results

This section is the implemented part of the chapter; it converts that research trial from architecture to implemented framework and shows the used environment setup. The mentioned proposed architectures are applied in the following proof-of-concept case, such that a social network mobile application called "Social Contacts" allows the user to send a message to his/her mobile contacts and show customized messages regularly during the whole day.

5.1. Social contacts application

This application includes the combination of the different request types and different amounts of data flow as shown in **Table 2** and will be clarified below.

The social contacts modules included in this use case are shown in **Figure 8**.

1. Cloud services modules.

- Authentication module: Allows the user to login to an existed account or register a new account.
- Feeds module: Shows the user the frequently post feeds from his contacts and allows the user to send a new post to the contacts.

2. Middleware module:

- Request states module: This is a middleware-related module that gains the benefits from the enhanced architecture and introduces a new mobile user experience to the sort of reliable applications, such that it contains all sent requests states to presented and be clear to the user action.

Module	**Function name**	**Flow**	**Request type**	**Data size**	**Forced attribute**
Authentication module	Login	• The user enters his phone number and chosen password • If the user authorized, the cloud service gets the saved user account to the mobile response; otherwise an error response is sent to the mobile client	Get	Small (only phone number and password)	Yes
	Register	• The user enters his phone number and chosen password • The application takes the permission to read the contacts info and sends them to the server • If the user doesn’t exist, the cloud service posts and inserts the user account with its related contacts into database; otherwise an error response is sent to the mobile client	Post	Medium (phone number, password, and all contact numbers)	No
Feeds module	**Send Post**	• The user enters/choses his/her contact phone number to send the post to • The user writes his post text • The cloud service adds this post in the database related with the sent contact	Post	Small- large (regarding the post content size)	No
	Delete Post	• The user choses to delete any post that appears in his/her posts • The cloud service deletes this post from the database	Delete	Small (the post id)	No
	Get Posts Feed	• The user opens or refreshes the screen to show the posts that are sent from and to him • The cloud service gets all the related posts to this user	Get	Medium-large (regarding the number of returned posts)	No
	Get Posts Feed Timeout	• The same functionalities as Get Posts Feed, but it does heavy computations and enlarge the response to consume a lot of time to make a timeout response	Get	Very large response	No

Module	Function name	Flow	Request type	Data size	Forced attribute
[Middleware component]—request states module	**Show Requests**	• The user opens or refreshes the screen to show the requests states and the request details sent from him using any of his/her mobile devices • The middleware service gets all the related requests with their exact states to this user			
	Retry Request	• The user chooses to retry the execution of specific failed request The middleware service posts the request in its reliable corresponding scenario			
	Delete Request	• The user chooses to delete any of his/her sent requests The middleware service deletes this request from the database, in order not to be shown or retried from the user another time			

Table 2. Social contact application functions.

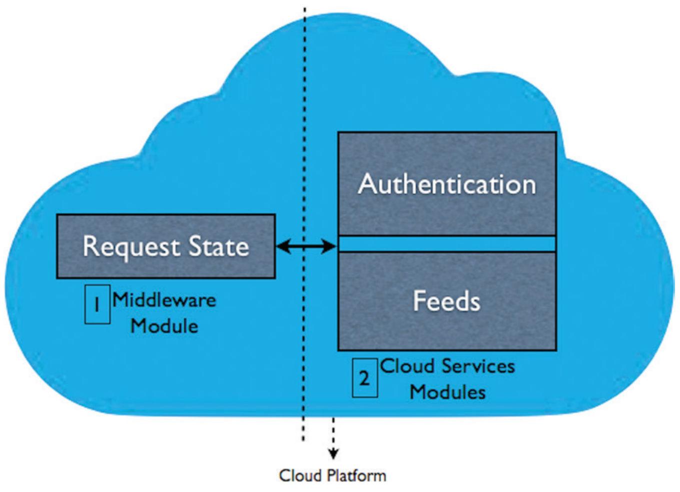

Figure 8. Social contacts modules.

5.2. Environment setup

The working environment of the programming languages and the tools that are used in building these enhanced architectures are explained as follows in **Table 3**.

The following notes regarding making the architecture as a standalone solution:

- The S/CSC is built as an independent and separate android library.
- The middleware and backend components are deployed in a separate cloud instance than the other. This ensures that middleware remains available in the case of any down issues that occur in the backend cloud instance.

Cloud services and middleware service component				
Language and frameworks	**Web service**	**Web socket**	**Mobile agent**	**Database and text format**
Java enterprise edition and hibernate framework [13]	RESTFul (JAX-RS)	Java API for WebSocket (JSR 356) [14]	Java Agent Development Framework (JADE) [15]	MySQL [16] and JSON [17]
Cloud service consumer (client)				
Platform		**Response caching**	**Web socket**	**Text format**
Android		File system	Java WebSocket Client [18]	JSON [17]
Service deployment				
Cloud service provider		**Application server**		
Openshift Cloud [19] for both middleware and backend each on separate instance		Glassfish application server [20]		

Table 3. Environment setup.

5.3. Result assessment

The proposed middleware architecture handles the effective cases that covered the corresponding reliability concept. Moreover, the performance comparison will be considered to ensure the usability and discuss the trade-off factors related to the middleware approach. The following factors are considered as the most affective factors regarding the mobile computing scale:

- Request size: The size of the request body that constructs and sends in the mobile client; it will be affected because of the additional attributes necessarily added for the middleware to ensure the reliable request.
- Response size: The size of the response body that it is received in the mobile client; it will not vary because there are no additional properties that need to be added from the middleware for any reason.
- Consuming time: The time taken from sending the request till receiving its response may be longer because of the additional middleware connection rather than the direct cloud service connection.

5.3.1. RSAM results

Regarding the social contact application functions, **Table 4** and **Figures 7–11** show the performance measurements through middleware component and through direct cloud service connection.

The measurements indicate the middleware component cost performance. Regarding the request size factor, the middleware adds additional 226 bytes to the request. These bytes used for the required attributes discussed in its description section. Although this number of

		Direct cloud connection			Middleware connection		
Module	**Function name**	**Request size (byte)**	**Response size (byte)**	**Consuming time (s)**	**Request size (byte)**	**Response size (byte)**	**Consuming time (s)**
Authentication module	Login Success	55	5	> 0	281	5	> 0
	Failure	53	315	> 0	279	315	> 0
Feeds module	Send Post	20217 (~20K)	3072 (~3 kB)	1	20443 (~20.5K)	3072 (~3kB)	1
	Get Posts Feed	25	191745 (~191kB)	2	251	191745 (~191kB)	2
		25	2167000 (~2MB)	21–26	251	2167000 (~2MB)	22–28
		25	4592949 (~4.5MB)	44	251	4592949 (~4.5MB)	44
		25	6828571 (~7MB)	65–66	251	6828571 (~7MB)	65–67

Direct cloud service results **timeout in each time** in the large response size with heavy required computations; the middleware connection results **timeout for the first time** but returns the response successfully from the middleware storage in the next retry.

Table 4. Social contact application performance measurements.

additional bytes in the request is considered a cost to the mobile client, this cost is low, and its complexity is O(1) as it is constant regardless the original request size.

The response size is the factor that is not changed, as shown in **Table 4**, response size columns. The middleware forwards the response exactly as returned from the cloud service. Therefore, we avoid using transformation overhead and additional cost from the middleware or the mobile client.

The consuming time is one of the most important factors for the mobile applications, which require quick responses. The measurements show that time may slightly vary about 2 s in

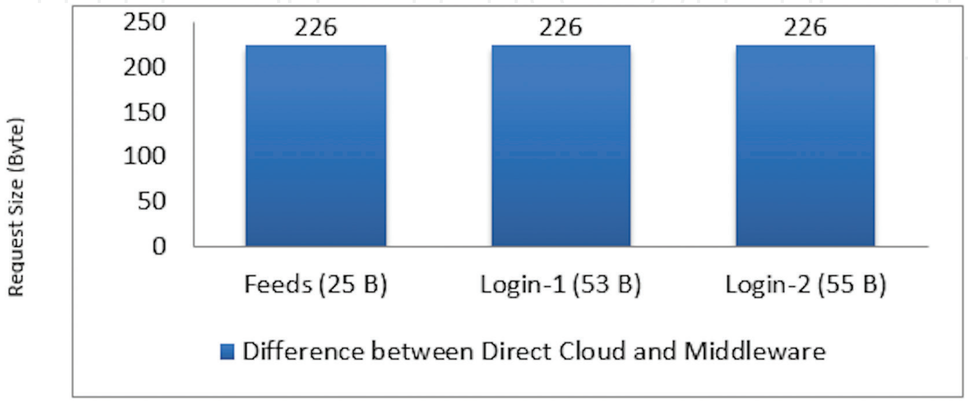

Figure 9. Request size factor of different services regarding the direct cloud and the middleware connection.

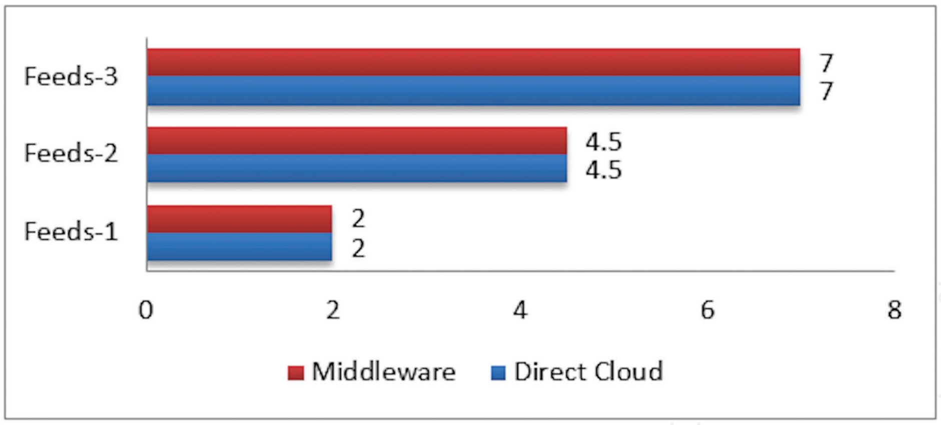

Figure 10. Response size of different services regarding the direct cloud and the middleware connection.

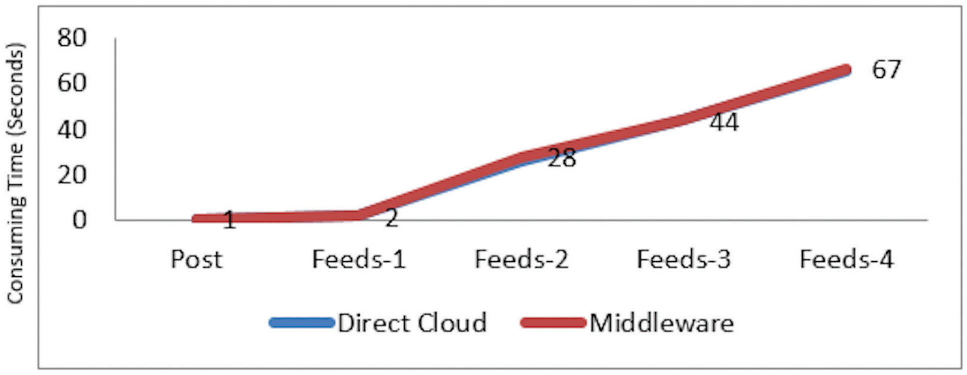

Figure 11. Consuming time for different services regarding the direct cloud and the middleware connection.

the medium response size (~2 MB) data class and the very large response (~7 MB) data class. However, it is exactly the same in the other cases without any overhead due to the middleware connection. This is because the middleware uses lightweight connections and lightweight data formats. In addition to its concerns with the request and response states resolving, the middleware does the minimal efforts in data transformation and data conversion.

5.3.2. RAMWS results

The proposed approach overcomes the timeout problem that threats the cloud service reliability. The two main concepts used to achieve this objective are middleware and the two-response technique. Each of them participates in solving the problem and has some aspects in the other side to be measured and compared with their responsibilities.

As shown in the above sections, the middleware achieves the availability and reliability in responding to the mobile client with the appropriate actions and data. From the cost perspective, it requires extra resources for this middleware handling and deployment, such that it is deployed in an application server, which is located in another cloud instance.

Regarding the two-response technique, it is managed by the middleware using two different protocols: the HTTP REST protocol and the WebSocket protocol, respectively. The first response is a temporary one that enhances the mobile experience by showing appropriate results to the user. After that, in the second response, the user receives the actual results on time through the WebSocket open connection. This technique costs an extra data communication because of the temporary response that is transferred using the REST response. **Figure 12** shows this cost regarding the different temporary response types. The waiting message response type (waiting_message) is the minimum data required to transfer simple message to the mobile client and has constant response size. The limited cached response type (limited_cached_response) sends limited size from the previously cached responses and has small variable response size relative to the size of the single object of data required. The latest cached response type (latest_cached_response) is the latest cached response in the middleware and has a size that is identical to the actual response size of the whole required data. **Figure 13** shows the network consumption difference between the proposed technique and the traditional one, such that the traditional technique consumes the network resource about 2–7 times depending on the network strength and the response size, compared with the proposed technique that uses the network resource without waiting time.

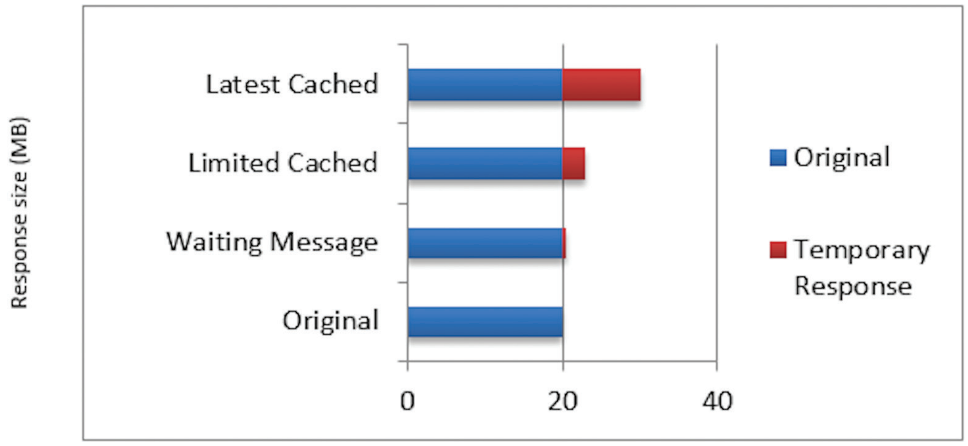

Figure 12. Different temporary response data size with its types compared with the original actual response size.

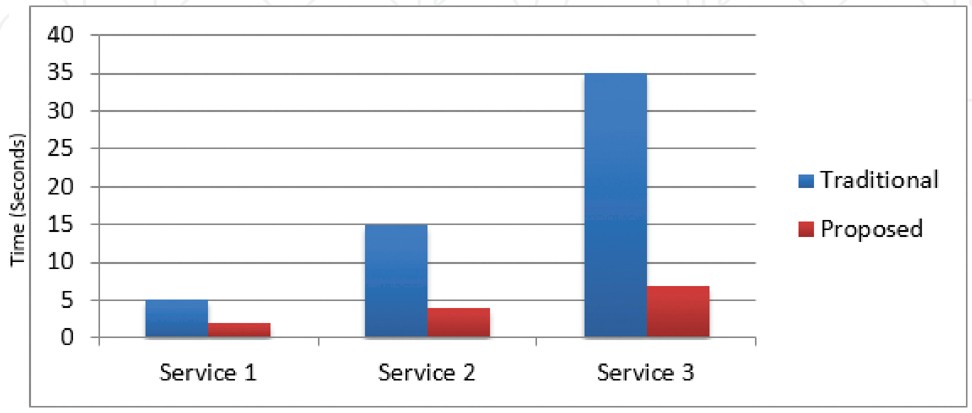

Figure 13. Network usage time for consuming three different services regarding the traditional and the proposed techniques.

6. Conclusion

In this chapter, we proposed and discussed two middleware-based approaches that achieve reliable web services through mobile cloud computing. These approaches focus on ensuring the service request execution under the different mobile environment conditions including intermittent connectivity and service unavailability conditions. Moreover, it ensures returning the appropriate response according to the request state.

The proposed Reliable Service Architecture using Middleware (RSAM) achieves the reliability by focusing on the request behavior rather than the request structure.

The RSAM focuses on ensuring the request execution and preventing the duplicate request execution as a result of the intermittent mobile connection. In addition, they consider the most important factors for the mobile client such as the request size and the response size for mobile client data transmission limitations and the service consuming time, which is critical for the mobile applications and their usability.

The service response size, database query time, and the required computations in the services are continuously increasing corresponding to the application usage, which causes timeout during the consumption of these cloud services. The timeout problem degrades the mobile usage experience and waste network usage time and threats the cloud service reliability. This problem requires restructuring the cloud service itself or the database schema to overcome such problem.

The RSAM solves this issue in the perspective of getting the response, such that the timeout problem occurs in the first consumption, but it retrieves the ready stored response in the middleware storage in the next retry. The proposed Reliable Approach using Middleware and WebSocket (RAMWS) achieves the web service consumption reliability in the aspect of overcoming the timeout problem. The RAMWS integrates the middleware approach and the two-response technique to achieve the cloud service consumption reliability. The two-response technique is used between the mobile client and the middleware, such that the first response is temporary and shows the appropriate results to the user till the actual results are received on time. The actual response is retrieved through WebSocket open connection once it is fetched using middleware service.

The proposed approach proved to overcome the timeout problem, which occurs in requests through mobile cloud computing, reduces the network usage time, and enhances the mobile experience to be more usable.

Author details

Amr S. Abdelfattah*, Tamer Abdelkader and El-Sayed M. El-Horbaty

*Address all correspondence to: amr.elsayed@cis.asu.edu.eg

Faculty of Computer and Information Sciences, Ain-Shams University, Cairo, Egypt

References

[1] Lomotey Richard K, Deters R. Reliable consumption of web services in a mobile-cloud ecosystem using REST. In: IEEE 7th International Symposium of Service Oriented System Engineering (SOSE); 2013. pp. 13-24

[2] Christensen JH. Using RESTful web-services and cloud computing to create next generation mobile applications. In: 24th ACM SIGPLAN Conference Companion Object Oriented Program; 2009. pp. 627-634

[3] O'Sullivan MJ, Grigoras D. Delivering mobile cloud services to the user: Description, discovery, and consumption. In: IEEE International Conference on Mobile Services (MS); USA. IEEE; 2015. pp. 49-56. DOI: 10.1109/MobServ.2015.17

[4] W3C Working Group. Web Services Architecture [Internet]. February 2004. Available from: http://www.w3.org/TR/ws-arch [Accessed: November 2017]

[5] Cobârzan A. Consuming web services on mobile platforms. Informatica Economica. 2010; **14**(3):98-105

[6] Chen M, Zhang D, Zhou L. Providing web services to mobile users: The architecture design of an m-service portal. International Journal of Mobile Communications. 2005; **3**(1):1-18

[7] McFaddin S, et al. Modeling and managing mobile commerce spaces using RESTful data services. In: 9th International Conference on Mobile Data Management, MDM'08; IEEE. 2008. pp. 81-89

[8] Kleimola J. A RESTful Interface to a Mobile Phone [Internet]. January 2008. Available from: http://www.researchgate.net/publication/228974867_A_RESTful_Interface_to_a_Mobile_Phone [Accessed: November 2017]

[9] Gonsai AM, Rushi RR. Enhance the interaction between mobile users and web services using cloud computing. Oriental Journal of Computer Science & Technology. 2014; **7**(3):416-424

[10] Yasumoto K, Yamaguchi H, Shigeno H. Survey of real-time processing technologies of iot data streams. Journal of Information Processing. 2016;**24**(2):195-202

[11] He Y, Salih OS, Wang C-X, Yuan D. Deterministic process-based generative models for characterizing packet-level bursty error sequences. Wireless Communications and Mobile Computing. 2013;**15**(3):421-430

[12] Amr S, Abdelfattah TA, EI-Horbaty EI-SM. RSAM: An enhanced architecture for achieving web services reliability in mobile cloud computing. Journal of King Saud University—Computer and Information Sciences. 2017:1-11. DOI: 10.1016/j.jksuci.2017.03

[13] Redhat. Hibernate Framework [Internet]. Available from: http://hibernate.org [Accessed: November 2017]

[14] Oracle. JavaTM API for WebSocket [Internet]. Available from: https://jcp.org/en/jsr/detail?id=356 [Accessed: November 2017]

[15] Jade (Telecom Italia SpA). JAVA Agent Development Framework [Internet]. Available from: http://jade.tilab.com [Accessed: November 2017]

[16] Oracle. MySQL Documentation [Internet]. Available from: https://www.mysql.com [Accessed: November 2017]

[17] JSON Data format [Internet]. Available from: http://www.json.org [Accessed: November 2017]

[18] Java WebSocket [Internet]. Available from: org.java-websocket: Java-WebSocket [Accessed: October 2016]

[19] Redhat. OpenShift Cloud Provider [Internet]. Available from: https://www.openshift.com [Accessed: November 2017]

[20] Oracle. GlassFish Java Server [Internet]. Available from: https://glassfish.java.net [Accessed: November 2017]