CHAPTER 3

DESIGN & DEVELOPMENT OF MOBILE CLOUD COMPUTING MODEL

3.1 Overview
3.2 Middleware Design
3.3 Private Service Mash up Platform
3.4 Enhance the Interaction between Mobile Users and CWS using Cloud Computing
  3.4.1 Middleware Architecture
  3.4.2 Mobile client implementation on middleware into mobile clients
  3.4.3 Service Objects
  3.4.4 Workflow
3.5 Mobile User Design
References


3.1 Overview

The objective of the Mobile Cloud Computing (MCC) architecture is to deliver an alternative for mobile users connecting to Cloud Computing Services. Figure 3.1 illustrates an overview of the MCC as well as its key features. The architecture design consists of three parts, the mobile clients, the middleware (data center) and the Cloud services. Since Cloud services are frequently controlled by service providers, the middleware accomplishes all the essential variation to the mobile clients.

Some services require real-time updates, for e.g., real-time news, Blog, Twitter and any specific kinds of service. The middleware furthermore pushes updates of service results to mobile clients via HTTP, HTTPS or email proximately after it obtains the updates.

![Figure 3.1 Overview of Mobile Cloud Computing](image)

Figure 3.1 Overview of Mobile Cloud Computing

Performance analysis and design for Mobile Cloud Computing on various platforms
The middleware is accountable for consuming the Cloud Computing services whether they are RESTful-CWS or SOAP-CWS and provides the service result to the mobile client. On the mobile user side, clients can state CWS or mash up services and future executes the pre-defined CWS on the fly. The middleware deliver RESTful CWS edge for the mobile clients. *Figure 3.2* directs how to execute / consume a path for Cloud Web Service. Point to note that the execution starts with a HTTP / HTTPS GET request whose URL path holds the resource identifier to the CWS. When CWS are implemented through the middleware, the following steps are being involved in the middleware.

a. The mobile client sends a HTTP / HTTPS GET request with an identifier of a CWS to the middleware.

b. The middleware transactions with interactions to the CWS (and generates RESTful CWS client if necessary).

c. The middleware abstracts (JSON or XML parsing) the required check results from the original service result and form a new service results in JSON format as well.

d. The middleware stores a copy of outcome with the service-ID in the database / file and returns the enhanced result to the mobile client.
Chapter 3 Design & Development Of Mobile Cloud Computing Model

Performance analysis and design for Mobile Cloud Computing on various platforms

3.2 Middleware Design

Figure 3.3 demonstrates the components in the middleware architecture style. The middleware has a RESTful service interface for...
mobile users. Through the presentation and application tier interface, clients can define and manage user's profile, Mash up services, Service Actions, and their value parameters and results. All the requests through the presentation and application tier interface are delivered to the persistence tier which reads and write data from and into the storage. The implementation requests of Service Actions go through the presentation and application tier interface. These requests are mostly plotted to read operations in the persistence tier. The service executer / load balancer constitutes service requests and passes them to the HTTP/HTTPS client who sends outward request to the Cloud services, here as an interface uses embedded browser or native mobile application. In all-purpose, the middleware delivers the following structures to advance the communication between mobile users and Cloud Services.

![Middleware Architecture Diagram]

**Figure 3.3 Components in Middleware Architecture**

- **Middleware Push**: Mobile users can subscribe to a service resource besides explicitly update service results cached on the middleware through the management interface. When the middleware gets an update of service results, it directs the update to all the mobile users.
that subscribed to this service result. The update is pushed to the users / clients (e.g. via email).

- Protocol re transformation: The middleware transmutes the SOAP CWS into RESTful CWS. The service executor handles normal HTTP requests for RESTful CWS in addition to SOAP messages for SOAP CWS. If the service is SOAP CWS, the service executor produces a specific SOAP client based on the provided WSDL, and then uses the produced client to interrelate with the Cloud Services. Toward the mobile users, all the services executions are through a RESTful CWS interface.

- Rendering optimal results: An unrefined CWS response encompasses data within a service precise format. Though, there are some problems. First, the mobile users do not need all the data at all the time. For e.g., the client may only need 6 instead of 15 news stories. Second, the novel data format may also not be well-organized for mobile users. The result optimizer first abstracts the required part of data from the raw response, and formerly makes a copy of the extracted result in numerous formats, for e.g., mobile HTML for mobile embedded browsers and JSON for native mobile apps. The middleware furthermore stores these reproductions of result in the service repository.

- Middleware Cache: Cache is based on the mash up services. The service repository keeps the enhanced service results into system storage for the latest implementation of the mash up facilities. The service results inform when the parameters of a mash up service variation. Users can also clean the caches via the management interface.

Similar most middleware, scalability is always a key unease. My methodology is to take maximum advantage of the Cloud platforms to host the middleware. Google App Engine, Heroku App, Amazon EC2
and Microsoft Windows Azure are the Cloud Platforms examined in this research. They have very diverse service model and performance characteristics. Chapter 4 describes and recognizes the middle ware implementations based on GAE, Heroku, Android and MWA in details.

3.3 Private Service Mash up Platform

On the mobile user side, the middle ware has a user interface which lets users describe a mashup application-based service. The middleware consumes a service-storage (SS) which stores user defined service data and an execution engine which performs WSs and pipes input and output of web services. So as to support a service mash up (one kind of a task), the middleware must first provision to consuming obtainable Web Services. Particular web service requests are well-defined by users via the mobile client and stored in the service-storage for upcoming accomplishment. The subsequent provides a user state of how to consume WSs from the mobile user through the middle ware [60] [61].

Here, define open-handed some user consequence, such as User1 is a mashup-based application developer. He wants to know all the imminent events in his city with use of his mobile phone. He knows that Yahoo Upcoming (Restful web service) offers such service and read out its online API document which defines how the service is used (e.g., providing co-ordinates as a factors). Above the user interface (UI) on the mobile user, he then defines a mash up application or service which contains a service action with all the required factor and preferred results. Finally, he or she executes the mash up service and gets the result showed on the mobile user’s device.

An additional user scenario is, if User2 wants to know where he is going or at what place he wants to reach then he / she can capable to
sight live map location which is based on Google API [59] [60] features and also add some mark pictogram to recollect that points or location with use of mash up application (task). At the outcome point-of-view user can accomplish mobile app on their android based smart-phone and gets the map after that user can adding mark-point with usage of touch screen and the catch suitable result for his discriminatory location with use of dissimilar Google map types such as, Hybrid or Normal. For development of this mobile app, researcher has using Android v2.2 platforms with eclipse tool via Helios service release 2 versions. Researcher have also used Google map API to associate live map location thru use of smart-phone device.

To create this mobile mash up app researcher have follow further down steps:


✓ Create a MyApp project into the eclipse IDE

✓ Download Google Play Service using eclipse IDE

✓ Import a Google Play Service lib using eclipse into MyApp

✓ Upgrading MyApp to display map

✓ Define <uses-permission> in to MyApp activity manifest file

✓ Add User Interface (UI) and fragment controls in to main.xml file

✓ Next, Go to Google API Console and take the Google API key which is unique for each API.

✓ Configure and Define an Android Key for MyApp API Project
✔ Generate MD5 and SHA1 key for a MyApp android app with usage of `keytool` command from cmd (command prompt)

✔ Take a SHA1 key in Google API console and produce new key for android application

✔ Now take that new produced android application key and placed into MyApp manifest file

✔ Additional steps are, Create Map fragment activity, create an object for Google Map and also generate setOnMapClickListener event and onMapClick as well for receiving longitude and latitude location for mobile users

✔ In next steps, Add communicating dissimilar touch features such as Add multiple marker, Map Tagging with title (mash up the app), Zoom-In, and Zoom-Out

✔ Mobile Users can also able to view their location in hybrid Google map or in normal Google map

Figure 3.4 shows the procedure with a series of screenshots on to the mobile clients.

With use of Normal Google Map
With use of Hybrid Google Map

Figure 3.4: Live Map on Android based Smart-phone with use of Google Map-API
3.4 Enhance the Interaction between Mobile Users and CWS using Cloud Computing

The objective is to use middleware to improve the communication between mobile clients and internet-based services as well as use Cloud platforms to improve the scalability and reliability of the middleware. For that Researcher have designed an embedded browser based mobile app, following sections are related to this goal.

RESTful is software application architecture modeled after the way data is accessed, modified and signified on the web. In the RESTful architecture, functionality and data context are considered as resources, and these resources are repossessed with usage of URIs (Uniform Resource Identifiers), characteristically links on the web. The resources are acted upon by consuming a set of modest, well defined processes. The RESTful design is fundamentally client-server centered architecture, and is considered to using as a stateless communiqué protocol, frequently HTTP. In the REST architecture, users and servers swapping re-presentations of resources by use of an interface and identical protocol. These principles encourage REST based applications to be light weight, have high performance, and simple.

RESTful Web Services are Web applications built on the REST style. They interpretation resources (usually, data and functionality) through web URIs, and use the four main H.T.T.P. methods to CRUD which are recognized as create, retrieve, update, and delete resources. RESTful CWS typically map the four main HTTP methods to the so-called CRUD procedures likewise, create, retrieve, update, delete, and further down table shows a representing of HTTP methods to these C.R.U.D. operations.
Table 3.1: HTTP Methods and their Act

<table>
<thead>
<tr>
<th>HTTP Methods</th>
<th>Operation (CRUD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieve a Resource from a server</td>
</tr>
<tr>
<td>POST</td>
<td>Create a Resource into a server</td>
</tr>
<tr>
<td>PUT</td>
<td>Update a resource into a server</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete a Resource from a server</td>
</tr>
</tbody>
</table>

The middle-ware that is suggested will act identical as re-presentation that is hosted on the Cloud platforms which deliver mobile clients access to Cloud services. The middleware design will develop relations between Cloud Services and mobile clients, for ex., optimization, caching and variation. The middle ware also provides comprehensive functions to the mobile clients. In wide-ranging, the architecture enriches the steadiness, functionality, and compatibility of the interaction among mobile clients and Cloud Web Services.

In direction to overthrow the troubles stated in the preceding sections, the Cloud Computing architecture provides the succeeding features to enhance the interaction among mobile clients and cloud web services.

No Loss of connection: Client and middle ware caching - Reproductions of service results are stored on both mobile clients and the middle ware. While the mobile users are incapable to connect to the middle
ware, the client-side cache is used. When the middleware to server connection is unavailable, the middle ware returns its cached data to the mobile users.

Bandwidth/Latency: Protocol re transformation reduces the latency as well as bandwidth of the client to service interaction. The middle ware transforms SOAP (Simple Object Access Protocol) to a much light weight format like a JSON through Restful Cloud Web Services. Transferring S O A P to light-weight protocols, like REST, shrinks processing time as well as the size of the messages.

Figure 3.5: Architecture for Mobile Clients Middle Ware

Optimization of Result: Result optimization reduces the size of the service results, thus decreases the bandwidth used to interact with internet-based services. The middle ware transfigures the format of services outcomes from XML to J S O N and eliminates unnecessary data from the original service end result. A smaller amount data transmitting also reduces network latency.

The goal of the middleware cloud manner is to provide a proxy for mobile users involving to Cloud services. Figure 3.5 shows an overview of the middleware cloud and its key features. The architecture involves
of three parts, the mobile-clients, the middle ware and the cloud services. Meanwhile Cloud services are regularly organized and controlled by service sources; the middleware accomplishes all the required variation to the mobile clients.

3.4.1 Middleware Architecture

The middleware is accountable for consuming the Cloud Services whether they are SOAP-based or REST-based services like JSON and delivers the service outcome to the mobile users. On the smart phone devices, mobile clients can define WSs and later perform the predefined Web Services. The middleware be responsible for REST interface for the mobile clients. Figure 3.2 indicates how to execute a predefined Web Services. Here note that the implementation starts with a HTTP GET request whose URL path holds the resource identifier to the web. When Web Services are accomplished through the middleware, the subsequent steps are elaborate in the middleware.

• The mobile users send a HTTP GET request with an identifier of a Cloud Web Services (CWSs) to the middleware.

• The middleware transactions with exchanges to the CWSs (and produces SOAP client, if necessary).

• The middle ware extracts (XML / JSON parsing) the essential service results from the original service result and form a novel service results in JSON format.

• The middleware provisions a reproduction of the result with the SERVICE_ID in the DB and returns the improved outcome to the mobile client.
3.4.2 Mobile client implementation on middle ware into mobile clients

In order to fulfilled the planned smart phone based design with client side native libraries. Smart phones have an embedded-browser which includes JavaScript (JS) libraries that implement several common functionalities of the client-side browser, for e.g., access a file-based system and location service.

To verify the smart phone based client design, Researcher have develop the project with an indication of generate a list of you tube channel as shown in Figure 3.6. The application is receiving response with usage of JSON parsing and a REST CWS with the mobile client design on smart-phone device. Via the application, the mobile clients can access the CWSs of the portal which is hosted on Google cloud over the smart phones.

The mobile client app can be distributed into three layers; Controller, Cache Manager, and User interface. The UI layer has two representations, embedded browser (web) UI and native UI. Figure indication how they look like on the device. Figure also confirmations the architecture of both applications. The controller is the key manager among the UI, cache-manager, and middleware. The controller forms the User Interface and catches data from the RESTful client or cache-manager. Suppose, at that time network connections are not reachable, the controller passes cached data to the UI modules.
Furthermore, Researcher have design and established one more mobile/website app design with an indication of admission system of college and day-to-day create a list of students with the full information which was given by the their parents or by students, this info was retrieved by an admin, and admin can also download that data in JSON format and in CSV format as well, from their mobile devices or computer system as shown in Figure 3.7, 3.8. The application is getting response through use of PHP, JSON construing with the mobile client design on smart-phone. By use of this app, the students or parents/guardians can right to use the Cloud Web Services of the portal which is hosted on Google cloud server over and done with the usages of smart-phones.

This mobile client’s app can be distributed into three layers; Google App Engine [60], Google Cloud Server and User interface. The UI layer has embedded browser (web) UI. Figure appearance how they look like on the mobile client’s device. The Google App Engine and GCS are the fundamental coordinators between the UI, and middleware. The GAE accomplish the user-based requests and gets data from the mobile client and storing is on Google Cloud Server. If network connections are not reachable, the control passes cached data to the U.I. components.

Figure 3.6: Layout on Android Mobile Clients
Performance analysis and design for Mobile Cloud Computing on various platforms

Chapter 3 Design & Development Of Mobile Cloud Computing Model

Figure 3.7: Student admission procedure at college using Mobile App

Figure 3.8: Admin can download in JSON and/or CSV format

Otherwise, it invokes the RESTful client to get data context from the middleware. The cache manager then keeps current received data on
a local file system. With the native UI, the client act together with the middleware asynchronously. When the native U.I. entails data, it permits a call-back to the controller and further carries on to receive UI events as shown in Figure 3.9.

The controller jumps a new thread to interrelate with the middleware. When the data arrives, the U.I. gets updated through the notification. By this model, the Native U.I. can be updated as soon as the data changes. The android app requests to wait for the facts and data to reach, because the native library can’t receive a Java Script notification. The android app also can’t be updated automatically when the data alterations.

The entire middleware application is hosted on a cloud platform, GAE, which usages the Services oriented architecture (SOA). The middleware architecture is implemented as an android app. The application uses the REST Internet-based Service interfaces to mobile clients; subsequently REST cloud web services are more suitable for mobile devices, because the middleware uses RESTful and JSON API of core Android libraries.

The middleware likewise procedures a popular JSON client library which provides functions of composing custom HTTP requests, distribution and getting HTTP requests and responses. The middleware architecture presumes the Cloud Web Services to return JSON responses, so that outcomes can be pull out using the android build-in library. User defined tasks, service actions, factors and outcomes are Java objects which map to database entities using the android API.
The middle ware still has a RESTful interface to mobile clients, however the Google Application Engine platform themself is a Web application server which can only handle server-based requests. The middleware creates and sends HTTP requests over the URL fetch service which implements the android RESTful structure interface.

To develop the interaction between mobile users and Web Services

- Evaluate the cross-platform ability of the mobile users design.
- Implement for the mobile client in different models.
- Consume RESTful CWS through the middleware.
- Transfers SOAP CWS to RESTful Services to be consumed by mobile clients.
- Reduce bandwidth consumption of mobile users.
- Push updates towards mobile users in real-time.

Using the Cloud platform as a way to improve consistency, robustness and scalability of the middle ware
• The middleware can be implemented on CloudSim and Google App Engine (GAE).

• Cloud platform increases the robustness, scalability and consistency of the middleware.

### 3.4.3 Service Objects

User defined CWS calls are kept in the service storage as a service entities. A CWS fundamentally consists of two fragments of information: service configuration describing the properties of the CWS (meta-data) and how to consume the service (module / parameters), and the user-specific parameter values needed to be passed to the CWS. There is a format describing a RESTful CWS (WADL), but it is not widely adopted. In the middleware, service entities abstract the essential elements of RESTful CWS. In the future, service entities will also be compatible for both SOAP and RESTful CWS.

System storage is a database or data file implementation. Each kind of entity is presented as a table. There are four kinds of entity: Application as Mash up Service (MS), Module as Service Action (SA), Version as parameter, in addition Instance is usage for getting the result and result value. Table 3.2 lists and describes the key attributes of each kind of entity. Figure 3.10 shows the hierarchical / categorized relations of each kind of entity.
Modules are a feature that lets mobile clients' factor large apps into logical components that can share “STATEFUL” services and interconnect in a secure way. An app that handles client requests might include distinct modules to switch other tasks, like as:

- API requests from mobile devices
- Internal, admin-like requests
- Back-end handling such as data analysis

Modules can be constructed to use different runtimes and to function with different performance settings.

At the peak level, an app is made up of one or more modules. Every module contains of configuration files and pseudo code. The files used in a module signify a version of the module as well. When you deploy a module, you constantly deploy a specific version of the module.

User can deploy multiple versions of the same module, to account for substitute executions or an enlightened upgrade as time goes on.

Every module and version essential has a name. A name can hold hyphens, numbers and letters. It can’t be longer than 63 chars and can’t start or end with a hyphen.
While running, a particular module/version will have one or more instances of an app. Each instance runs its own isolated executable. The number of requests running next to any time is determined by on the type of module's scaling as well as the amount of incoming requests:

- Application as Mash up Service (MS): MS is a container for service actions. The MS delivers clients with an abstract grouping of related service actions in addition to an edge for preventing external access. Clients can also able to share their MS among the other clients.

- Module as Service Action (SA): SA is a prime object on which the app or mash up service is created. SA describes all the essential characteristics to consume a CWS; the URL to catch the CWS, the interface procedure, parameters required by the CWS, the preferred results and so on.

- Version as Parameter: Parameter is not only a name and value pair, but also contains of meta-data, for e.g., the source of a value (user input / output of other app / service) and how they are delivered to the CWS (through the HTTP-Header, URL query-string, content or else one).

- Instance as Result: The result defines how the representation processes the CWS response and how the users existing the result. Only data stimulating to the client will be removed from the response. According to the result type, the users’ display it in particular forms (maps, tags, text and so on).

- Instance as Result value: The reason for splitting result and result value is that a result can have multiple copies of values dependent on targeted users. The alternative keeps copies of results in
local database for different users, for e.g., HTML for browser, JSON / CSV for native application.

<table>
<thead>
<tr>
<th>Entity type</th>
<th>Attributes</th>
<th>Description</th>
<th>Probable Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mash up Service</td>
<td>owner</td>
<td>The creator of the MS is the owner</td>
<td>user id (i.e. full name, e-mail)</td>
</tr>
<tr>
<td></td>
<td>isPrivate</td>
<td>Whether or not let others access the MS</td>
<td>True/False</td>
</tr>
<tr>
<td>Style</td>
<td></td>
<td>CWS style (RESTful) of the containing SA</td>
<td>RESTful</td>
</tr>
<tr>
<td>base URL</td>
<td></td>
<td>The common base URL of the containing SA</td>
<td>URL as (r-gkck-msc.appspot.com)</td>
</tr>
<tr>
<td>Service Action</td>
<td>Name</td>
<td>Name of the SA</td>
<td>name identifier</td>
</tr>
<tr>
<td></td>
<td>method</td>
<td>HTTP</td>
<td>GET / PUT /</td>
</tr>
<tr>
<td>Parameter</td>
<td>Name</td>
<td>Description</td>
<td>Source</td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>request method</td>
<td>POST</td>
<td>The format that the CWS accepts</td>
<td>Format standards (XML, CSV, JSON)</td>
</tr>
<tr>
<td>consume format</td>
<td></td>
<td>The format of the CWS response</td>
<td>Format standards (XML, CSV, JSON)</td>
</tr>
<tr>
<td>produce format</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>parameters</td>
<td>Parameters passed to the CWS</td>
<td>Reference to parameter entities</td>
<td></td>
</tr>
<tr>
<td>results</td>
<td>Result definition</td>
<td>Reference to result entities</td>
<td></td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Parameter name used in the CWS</td>
<td>name identifier</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>Parameter value</td>
<td>Depend on source</td>
<td></td>
</tr>
<tr>
<td>source</td>
<td>Source of the parameter</td>
<td>Mobile Client / Mash up</td>
<td></td>
</tr>
<tr>
<td>Embedded</td>
<td>How the parameter</td>
<td>Path / Content /</td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>Name</td>
<td>Result name</td>
<td>Result name identifier</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
<td>-------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Path</td>
<td>Path for extract result from CWS response</td>
<td>XPath for XML, CSV file, JSON file</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Result content type</td>
<td>Text</td>
<td></td>
</tr>
<tr>
<td>Values</td>
<td>Result values</td>
<td>Reference to result value entities</td>
<td></td>
</tr>
<tr>
<td>Result value</td>
<td>received date</td>
<td>When the value is received (cached)</td>
<td>Date</td>
</tr>
<tr>
<td>Expire</td>
<td>How long the value expires</td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>Target</td>
<td>The targeted client</td>
<td>Mobile / Embedded browser</td>
<td></td>
</tr>
</tbody>
</table>
3.4.4 Workflow

Following is the design of workflow using use case diagram and UML diagram, to define a workflow of an app for instance e.g., “MyApp” plus additional one is “r-gkck-msc” to show how they can work for mobile users, for “MyApp” as well as for another one is for students and admin for “r-gkck-msc” Web UI app using as mobile users. Figure 3.11 expressions an example of mash up application for mobile users and Figure 3.12 and Figure 3.13 shows an example of another mash up application for students and administrator of an “r-gkck-msc” domain.
Performance analysis and design for Mobile Cloud Computing on various platforms

Chapter 3  Design & Development Of Mobile Cloud Computing Model

Figure 3.11: Use Case for Mobile Users for Private Service Mash up Application
Chapter 3 Design & Development Of Mobile Cloud Computing Model

Performance analysis and design for Mobile Cloud Computing on various platforms 89
The above example includes two Service Action (SA), one is for admission counseling where students can find information about organization, course information such as course eligibility criteria, fees structure from the college based on updated student's information on student's current registration using "r-gkck-msc" service and administrator of this process can login into the system and able to view the students list, download the students list for college references in various formats such as CSV / JSON. Another Service Action (SA) is for finding location from the mobile devices and is also able to add multiple markers and map tagging for their own usage with use of "MyApp" native app.

Figure 3.13: Use case for students and admin for Admission Counseling App
3.5 Mobile User Design

Continuously utmost of the new mobile platforms like Android 4.4.4, Blackberry OS 6.0 and Windows 7.1, mobile web-service users can be run either as embedded browser applications or pure native apps. Native apps are platform or device dependent. They must be functional using the programming languages which is the platform supports and alive in the application layer of the Android platform. Figure 3.14 displays where native apps localize on the Android based platform, and Figure 3.15 displays where native apps localize on the Windows based platform. Performance, functionality and scalability of native apps primarily depends on the core API, libraries file and the mobile OS platforms.

Figure 3.14: Native App on Android Platform
Additional method to implement the mobile CWS user is by using an embedded (rooted) browser. The user app runs on a Web browser which is rooted inside of a native app. Figure 3.16 displays how the embedded browser know-how works. The user application can be entirely applied using a browser supported languages likewise HTML, JavaScript and CSS, whichever is more applicable.

The rooted browser can load custom JavaScript libraries which can access the native codes exclusive by the app.
Performance analysis and design for Mobile Cloud Computing on various platforms

Chapter 3 Design & Development Of Mobile Cloud Computing Model

Figure 3.16: Client App on Embedded Web Browser

Table 3.3 lists pro and cons of pure native and embedded web browser apps (with-out custom libraries).

<table>
<thead>
<tr>
<th></th>
<th>Native Application</th>
<th>Web Work (Embedded Browser) Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros</strong></td>
<td>• Performance (compiled code)</td>
<td>• Platform liberated</td>
</tr>
<tr>
<td></td>
<td>• Full access to native API</td>
<td>• A lesser amount of specialty requisite</td>
</tr>
<tr>
<td></td>
<td>• Rich GUI features</td>
<td>• Easy to maintain and upgrade</td>
</tr>
<tr>
<td></td>
<td>• Easy to test and debug</td>
<td></td>
</tr>
<tr>
<td><strong>Cons</strong></td>
<td>• Platform dependent</td>
<td>• Browser compatibility</td>
</tr>
<tr>
<td></td>
<td>• Maintenance and upgrade cost</td>
<td>• Performance (interpreter)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Browser limitations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• No access to</td>
</tr>
</tbody>
</table>
Table 3.3 Pros and cons of pure native and embedded web browser apps

The projected mobile user design is a hybrid solution which associates both native and embedded web browser app. Figure 3.17 is the overview of the user proposal. It displays a straight-forward Model View Controller (MVC) form. The User Interface (UI) is considered within the embedded browser through use of HTML, JavaScript and CSS. When the UI modules want service data, they raise the custom JavaScript libraries to pull the facts and data from local side cache. If the native cache doesn’t hold a latest copy of queried data, the RESTful user interrelates with the middle ware to get the data. The data are before accepted to the data module and kept in the native file system. Here note down that the data accepted to the embedded browser is in JSON format which can be definitely parsed by a JavaScript.

![Figure 3.17 Mobile user design](image-url)
The split-up of UI modules and the user makes the design platform independent. To transformation the application to a pure native app UI, the built-in / embedded browser UI can be interchanged by native UI and the user can be re-processed. The r-restful-ws user can moreover implement push technology. Push technology allows a server to push contented to the users, so as to improve the device energy, data traffic, and bandwidth used. The next chapter describes the implementations on Windows mobile and on cloud servers for the mobile users.
REFERENCES


