A REFERENCE FRAMEWORK ON NATIVE APPLICATION FOR MOBILE MASHUP

TRITA-ICT-EX-2012:15
MASTER OF SCIENCE THESIS
KTH
STOCKHOLM, JAN 2012
A MASTER’S THESIS
BY
ATIK MAHMUD
PERSON ID: 830101-3812

SUPERVISOR
&
EXAMINER
ASSOCIATE PROFESSOR: KONRAD TOLLMAR
Abstract

Mashup is an application that uses and combines data, presentation or functionality from different sources. These data are aggregated to create values which may help in the decision making in a business organization or in daily life. A Mashup application also saves IT investment, since the enterprise does not need to make an individual application for every decision making IT tools. That is why; the growth rate of the Mashup adoption in the business organization is exponential. To a large degree does development in mobile industry follow the same path and mobile services and application rely to a large degree on combining external data into effective and new combinations.

The main aim of this study is to provide a reference framework on the native application for the mobile Mashup. In addition following sub goals are considered: point out the feasible architecture of the Mashup for mobile devices, describe and analyze the features and core elements of the different type Mashup, point out and analyze the necessary components for the Mashup, and define the best components to build a Mashup in the mobile devices. To do that research the following methods are followed: literature review, components development and performance test, propose reference framework based on performance test, implementation and analysis of two case studies based on the proposed reference framework.

Subsequently, based on the methodology, this study came up with a reference framework for the consumer type mobile Mashup on the native application which consists of the REST client, JSON data and XML data, JSON and GSON parser for JSON data and SAX parser for XML data. Additionally, the experiments show that performance of web service protocol and parser depends on the length of data format. That is why, a user guideline is included in the proposed reference framework.

Finally, two Mashup case studies are made based on the proposed reference framework. These implementations prove the acceptance of the proposed reference framework in the area of Native mobile Mashup application development.
Acknowledgements

I would like to thank my examiner and supervisor Associate Professor, Konrad Tollmar for his technical guidance and continued support. I am also grateful for my family’s support that kept on the track.
## CONTENTS

1. Introduction ...................................................................................................................................... 11
   1.1 Background ................................................................................................................................. 11
   1.2 Problem formulations .................................................................................................................. 11
   1.3 Aim and goals ............................................................................................................................... 12
   1.4 Methods ....................................................................................................................................... 12
   1.5 Case study ..................................................................................................................................... 13
   1.6 Scope of the thesis ....................................................................................................................... 14
   1.7 Thesis contribution ...................................................................................................................... 14
   1.8 Limitation ..................................................................................................................................... 15
   1.9 Thesis outline .............................................................................................................................. 15
   1.10 Key concepts ............................................................................................................................. 16

2. Mashup and its core elements ........................................................................................................... 17
   2.1 Mashup ....................................................................................................................................... 17
   2.1.1 Types of Mashup ...................................................................................................................... 17
   2.1.2 Styles of Mashups .................................................................................................................... 18
   2.1.3 Layered architecture of a Mashup .......................................................................................... 19
   2.1.4 Resources ............................................................................................................................... 20
   2.1.5 Some popular Mashups .......................................................................................................... 20
   2.2 Summary ..................................................................................................................................... 21

3. Service provider, Web service and protocol ....................................................................................... 23
   3.1 Service provider .......................................................................................................................... 23
   3.2 Web service .................................................................................................................................. 23
   3.3 PROTOCOLS ............................................................................................................................... 24
   3.3.1 Simple Object Access Protocol (SOAP) .................................................................................. 24
   3.3.2 REPRESENTATIONAL STATE TRANSFER (REST) ................................................................. 25
   3.3.3 Web API ................................................................................................................................... 27
   3.3.4 Comparison of SOAP and REST for Mobile web service application ................................. 27
   3.3.4 Summary .................................................................................................................................. 30

4. Data, parser and Framework .............................................................................................................. 31
   4.1 XML .......................................................................................................................................... 31
4.2 JSON ................................................................................................................................... 31
4.3 Comparison between XML and JSON ................................................................................... 32
4.3.1 Performance analysis of XML and JSON in respect of allocation of time and resources allocation ........................................................................................................................................ 32
4.4 Processing Data format ....................................................................................................... 34
4.4.1 XML Parser ....................................................................................................................... 34
4.4.2 Comparison among different XML parsers ........................................................................ 35
4.4.3 JSON parser ...................................................................................................................... 36
4.4.4 GSON ............................................................................................................................... 36
4.5 Framework .......................................................................................................................... 37
4.6 Developing platform analysis .............................................................................................. 38
4.7 Summary ............................................................................................................................. 40

5. Implementation ................................................................................................................................ . 43
5.1 Design Motivation and Assumptions ................................................................................... 43
5.2 System Architecture ............................................................................................................ 43
5.3 Assumptions ....................................................................................................................... 44
5.3.1 Limitation of Mobile phone and some suggested solutions ............................................... 44
5.4 Layered architecture ........................................................................................................... 45
5.5 Platform choice ................................................................................................................... 45
5.6 System design ..................................................................................................................... 46
5.6.1 GPS location provider ....................................................................................................... 47
5.6.2 RESTful service provider component ................................................................................ 48
5.6.3 SAX parser ........................................................................................................................ 49
5.6.4 JSON parser ...................................................................................................................... 50
5.6.5 Gson parser ...................................................................................................................... 50

6. Experiment and analysis .................................................................................................................... 51
6.1 Motivations and performance indicator .............................................................................. 51
6.2 Test bed .............................................................................................................................. 51
6.3 Method of the experiment .................................................................................................. 53
6.4. Experiment observation ..................................................................................................... 53
6.4.1 Character length difference .............................................................................................. 53
6.4.2 difference of Heap utilization in percentage ................................................................. 56
6.4.3 Difference of Total Heap utilization ............................................................................. 58
6.4.4 difference of the parser Execution time ....................................................................... 59
6.4.5 Difference of the REST Execution time (response time) .......................................... 61
6.4.6 Difference of the Total Bandwidth receive ................................................................. 62
6.4.7 Difference of Total Bandwidth sent .......................................................................... 64
6.4.8 difference OF CPU thread uses ................................................................................. 65
6.5 Summary ....................................................................................................................... 67
7. Result .................................................................................................................................. 69
  7.1 Result of the experiments ............................................................................................. 69
  7.2 Prove of the assumptions ............................................................................................ 70
  7.3 Proposed Reference framework ................................................................................... 70
  7.4 Case study analysis ...................................................................................................... 72
    7.4.1 Case study 1 ....................................................................................................... 72
    7.4.2 Case study 2 ....................................................................................................... 73
  7.5 The UML representation of the case study ............................................................... 74
8. Conclusion ....................................................................................................................... 79
LIST OF FIGURES

Fig 1: Mashup..................................................................................................................................................................................... 17
Fig 2: Client/Web-based Mashup ........................................................................................................................................................ 19
Fig 3: Server-based Mashup............................................................................................................................................................ 19
Fig 4: Client/Web-based Mashup...................................................................................................................................................... 44
Fig 5: Layered architecture .......................................................................................................................................................... 45
Fig 6: Character length differences of various service provider responses in test bed1 ........................................... 54
Fig 7: Character length differences of various service provider responses in test bed2 ........................................... 54
FIG8: The XML responses from the traffic service provider ................................................................................................. 55
FIG9: The JSON responses from the traffic service provider ................................................................................................. 56
Fig 10: Difference of heap utilization in percentage in test bed1 .......................................................................................... 57
Fig 11: Difference of heap utilization in percentage in test bed 2 ........................................................................................ 57
Fig 12: Difference of total heap utilization in test bed 1 ........................................................................................................... 58
Fig 13: Difference of total heap utilization in test bed 2 ........................................................................................................... 59
Fig 14: Difference of the parser execution time to process response data in test bed 1 .......................................... 60
Fig 15: Difference of the parser execution time to process response data in test bed 2 .......................................... 60
Fig 16: Difference of the REST Execution time in test bed 1 ............................................................................................ 61
Fig 17: Difference of the REST Execution time in test bed 2 ............................................................................................ 62
Fig 18: Difference of the Total Bandwidth receive in test bed1 .......................................................................................... 63
Fig 19: Difference of the Total Bandwidth receive in test bed2 .......................................................................................... 63
Fig 20: Difference of Total Bandwidth sent in test bed1 ......................................................................................................... 64
Fig 21: Difference of Total Bandwidth sent in test bed2 ......................................................................................................... 65
Fig 22: Difference of CPU thread uses in test bed1 .................................................................................................................... 66
Fig 23: Difference of CPU thread uses in test bed2 .................................................................................................................... 66
Fig 24: Sequence diagram of case study 1 ............................................................................................................................... 75
Fig 25: User location and weather forecast mashup ................................................................................................................... 76
Fig 26: Sequence diagram of case study 2 ............................................................................................................................... 77
Fig 27: The surrounding Area and the traffic information Mashup .......................................................................................... 78
LIST OF TABLES

Table 1: Mashup chart ..........................................................................................................................................................21
Table 2: Protocol supported by Mobile OS ..................................................................................................................30
Table 3: Summary and comments on experiments over protocols ..............................................................................30
Table 4: Mobile OS and supported data format ..............................................................................................................33
Table 5: supported parsing methods by Mobile OS .....................................................................................................36
Table 6: Mobile operating systems and development language ..................................................................................39
Table 7: Summary and comments on experiments of data processing ......................................................................41
Table 8: summary of the experiment and final consideration ....................................................................................68
Table 9: Proposed reference framework for native Mobile application ....................................................................72
1. INTRODUCTION

1.1 BACKGROUND

Mashup is an application that uses and combines data, presentation or functionality from different sources [4]. These data are aggregated to create values which may help in decision making in a business organization or in daily life [30]. For instance, “consider a supply chain management example in which a regional operations manager constructs a Mashup application containing a weather feed, a Yahoo! map, and enterprise inventory data to aid in the logistical management of merchandise in response to a natural disaster such as a hurricane” [30]. A Mashup application also saves IT investment as the enterprise does not need to make an individual application for every decision making IT tools [31]. That is why, the growth rate of the Mashup adoption in the business organization is exponential [31].

To a large degree does development in mobile industry follow the same path and mobile services and application rely to a large degree on combining external data into effective and new combinations. In addition, innovation of the 3G and smart phones has opened a wing to meet the potential business growth of the Mashup application.

1.2 PROBLEM FORMULATIONS

To meet the potential business growth of Mashup, a large variety of researches have been carried out in the area of mobile Mashup. Consequently, a reference framework for the mobile web Mashup has been explored in the study entitled “Mobile Web Mashup” [29]. However, there are a number of applications that cannot be done with these tools, for example background processes or applications that require an immersive experience such as augmented reality. Additionally, there are some issues presented in Mobile Web Mashups, such as the Same Origin Policy, that do not exist in native applications. Hence, this study will focus on the Native application for mobile Mashup.

In this context, the key question is what an effective native application development environment is for mobile Mashup, including mobile application development challenges and reference framework performance etcetera [1],[2],[42],[43],[44],[45] are as follows:
1. Challenges to design and develop an application to run on the large variety of mobile platforms.
2. Developers have to develop applications adjustable to a small display of mobile phones.
3. Developed applications should use less power consumption and CPU utilization for longer mobile battery power.
4. Applications should consume lower network bandwidth.

The overall architectural solution and detailed component design are the main factors to solve the above questions.

1.3 **AIM AND GOALS**

The main aim of this study is to provide a reference framework on the native application for the mobile Mashup with well-executed components. It is worth mentioning that, this study will focus on the Mashup for native application. To meet the aim and to address the research questions, the following goals can drive the study to reach the destination.

1. Point out the feasible architecture of a Mashup for mobile devices.
2. Describe and analyze different type of Mashup
3. Point out and analyze the necessary components for Mashup.
4. Define suitable components to build a Mashup in the mobile devices.

1.4 **METHODS**

Initially, the literature review undertaken in the area of the Mashup, protocol, data format and parsers. This study provides brief description of the Mashup, its architecture and necessary components. Subsequently, the research papers on the protocols, data formats and parsers were considered to gain the details mechanism of the Mashup components. Additionally, some research papers of the comparison study were analyzed to narrow down the experiments area. These analyses on the research papers expanded the applicable area of this study. Furthermore, three Mashup applications and some mobile application development platform were analyzed to see the acceptance of mobile OS by the manufactures.
Next, Core components of the mobile Mashup will be implemented and tested to verify the performance. These performance analyses lead us to propose a reference framework.

Finally, two mini case studies (1.5) will be implemented by following the proposed reference framework to verify the challenges of mobile native application development.

The Methods are used in this study could be sum up as follows:

1. Literature review
2. Components development and performance test
3. Propose reference framework
4. Implementation and analysis of two case studies based on the proposed reference framework.

1.5 Case Study

In this study following case studies were carried out:

Case study 1: A location based weather forecast system.

Description
Nowadays, a mobile user may want to know the weather forecast as well as the graphical interface to specify his position and detail location information. The mobile of the user takes the latitude and the longitude information from the GPS service provider and passes it to the Google map provider, the Geocoder server and the weather forecast service provider to receive weather forecast and location specification. The Mashup application will combine and display the data to the user after receiving all the information from mentioned service providers.

Case study 2: A traffic information provider application.

Description
This Mashup application provides the traffic information of a specific location of the user along with the specification of surrounding area of provided location name. This application takes the location name of a user and passes that to the Bounding box service provider and traffic information service provider to receive the traffic information. Finally the display shows the traffic information along with the surrounding area of the provided location.
1.6 Scope of the Thesis

Primarily, this study could be helpful for mobile application developers who are willing to start in the area of mobile native application development focusing on the web service. This study will provide a guideline in the area of the web service which could lead a developer to create a quick solution.

Subsequently, this study provides a guideline to measure the performance of a web service application and parser in the mobile device. Therefore, it helps to verify the performance of already developed mobile web service and the Mashup applications, that could help to solve resource utilization problems in the mobile device, if exists.

Finally, this study might grow the interest of a researcher who is working on the Mashup and willing to develop a Mashup in the mobile devices, since it provides a feasibility study of the mobile Mashup development environment on native application. Moreover, this work could be used as a reference framework for the mobile native Mashup application.

1.7 Thesis Contribution

The main contribution of this study was to propose a reference framework for the mobile Mashup. The identified results were:

1. A fully functional and fewer resource consumed reference framework for mobile Mashup using native application has been proposed (7.3).
2. Fully functional client type Mashup architecture has been implemented and tested for mobile devices (7.5).
3. Verified performance of XML and JSON in REST protocol in respect of response time (6.4.5), bandwidth receives (6.4.6) and bandwidth sent (6.4.7). It has been observed that performance of response time varies time to time as it depends on network performance and server performance. XML uses more Bandwidth receives than JSON. In bandwidth test, I have considered bandwidth receive as main factor as it contains the response data. On the other hand, bandwidth sent only used to service call.
4. Verified performance of SAX, JSON and GSON parser in respect of execution time (6.4.4). It is observed that performance of these parsers depends on data length of the web
service responded data format. That is why; a guide line is introduced to use parser in the reference framework.

5. Verified the performance of web service and data processing components for Mashup applications in respect of memory utilization (6.4.2), memory allocation in percentage (6.4.3) and CPU utilization (6.4.8). It is observed that results are same in two different environments.

1.8 LIMITATION

This work meets four major challenges of native mobile application development for mobile Mashup application such as resource uses and bandwidth uses for receiving response from web services. Yet, there exists some challenges which were not considered here for time limitation such as portability issues, security issues.

Next, this work proposed a reference framework based on data length of a web service response but here no data range is specified for declaring the short data length and large data length (7.3). Cause it was very difficult in this study to find a real web service provider with JSON and XML data response for same information.

Subsequently, this study tested Mashup application in emulator with maximum four web services (7.5.1). Therefore it is needed to find out the mobile device performance by increasing the number of web services for user satisfaction.

Finally, the proposed reference framework (7.3) based two performance tests on two different environments with emulator due to time limitation. It is seen that some performance indicators such as execution rest execution time and bandwidth using for resource request varies from device to device. The cause behind this is network performance (jitter, server delay) and Ethernet card of devices (6.5). So it is needed to test these performances in stable network with different mobile set.

1.9 THESIS OUTLINE

The thesis work is structured as follows. Chapter 2 provides details of the mobile Mashup with its core components and architecture. Chapter 3 provides detailed description of the Service
provider, Web service and Protocol with analyses. Chapter 4 provides detailed description of the data format, parser and framework with analyses. Chapter 5 provides the design and implementation of the core components of the Mashup such as the REST client, and Parsers. Chapter 6 describes the experiment details. Chapter 7 provides the result analyses and proposed reference framework with case study implementation. Finally, conclusion reviewed the goals of this study with limitations and future works.

1.10 **Key Concepts**

**Mashup** is an application which represents a combination of resources from one or different sources. The data sources can be different from each other in terms of presentation or functionality. An example would be Nokia Taller Mashup.

**Native application** is an application that installed and runs on mobile devices with built in environment. Native programs use a software development kit varying from device to device for application development. An example would be SMS sending and receiving applications in mobile phones.

A **framework** is an architectural design for object oriented systems which describes the components of the system and the way they interact [23]. Here components of a framework reduce the difficulties of application development. This difficulty may contain application environment constrains and requirement constrains. Therefore performance analysis of necessary components of a focused topic is needed to propose a framework. Moreover, a Reference framework introduces an effective way to solve the specific topic with guild line and suitable components.
2. Mashup and its Core Elements

2.1 Mashup

The Mashup application represents a combination of resources from one or different sources[4]. Data sources can be different from each other in terms of presentation or functionality. In most of the cases it uses open APIs and data sources to produce enriching results. Therefore the process should follow a simple and rapid integration in order to get good results that make the data more useful for personal and professional use.

Figure 1: Mashup

Figure 1 shows an overview of Mashup featuring a client by integrating resource 1 and resource 2. The main characteristics [33] of Mashup are:

- Combination is a process for combining data from different resources into one interface.
- Visualization is the representation of data to the user.
- Aggregation is a process in which information is collected and expressed in a summary to the user.

2.1.1 Types of Mashup

There are three types of Mashup which can be described as below:

- **Data Mashup** combines similar types of media and information from multiple sources into a single representation [32]. It is the opposite of the Consumer Mashup in terms of data or media type. An example would be the Microsoft Popfly Mashup.
- **Consumer Mashup** combines different types of data from various public sources by organizing it with a simple user interface to serve the general user need where less-secured data is concerned [32]. An example would be Nokia Taller Mashup.

- **In Business Mashup** decisions are made using organization's own resources and external data [32].

Main difference between business Mashup and consumer Mashup can be describe as follows: the business Mashup needs the level of integration with business computing environments, security and access control features, governance and sophisticated programming tools which are missing in consumer type Mashup [31].

The most important features [33] of the Mashup technology can be described as below:

- **Data freshness:** The application should use current resources to make the right business decisions.

- **Data security:** It is important to ensure data Confidentiality, Integrity, Authenticity and Availability.

- **Rich visual interface:** It allows users to see the rich data format by offering visualization interfaces where these interfaces can be analytical or essential to meet the business logic. An example would be the IBM Lotus.

### 2.1.2 Styles of Mashups

Mashup combines data from different sources where a data combination process can be done in the end user application or in another device. This data combination entity can be defined by style, where style represents the architecture of the data communication methods. Therefore different types of Mashup styles are described as follows:

- **Client/Web-based Mashup:** In this type of Mashup, the end user requests resources from different service providers. Subsequently, the service providers respond directly according to the request. After that, the end user combines and processes the collected information from service providers [34].
Figure 2 shows a requester (client) requesting directly to different service providers (Sp1, Sp2, Sp3) for resources and service providers are providing data directly to the client. After that, the requester combines and reformats the response data.

- **Server-based Mashup**: This architecture consists of client server communication where the server analyzes and re-formats the data according to the client’s request form and responses to the client after receiving all information from the service providers. Thus all the process is done in the server side [34].

---

**2.1.3 Layered Architecture of a Mashup**

Mashup combines, analyzes and reforms data after receiving it from different sources in a definite format and provides feedback in a definite way. There are three layers in a Mashup which are as follows:

- **Presentation / User interaction** layer initiates the Mashup service by calling an underlying application method and represents the final output to the user [34]. Example:
visual client, HTML /XHTML, CSS, JavaScript, Asynchronous Javascript and XML (Ajax).

- The **Web Services** maintain client and service provider communications through protocols where client makes a request and service providers provide the responses [34]. Examples could be XMLHTTPRequest, XML-RPC, JSON-RPC, SOAP and REST.

- **Data** defines the data structure for communication between the requester and the service provider to formulate requests and responses in a simple way [34]. Examples could be XML, JSON and KML.

### 2.1.4 Resources

A Mashup service depends on the service providers’ resources where service providers provide data to different requesters for maximum utilization of resources. Some service providers and their details can be described as below:

- **Mobile operator APIs** are provided by the mobile operators to help the developer to introduce a new information service. One example is the Vodafone Betavine API which is a location based application that helps developers to develop new applications by using it.

- **Web service APIs** are provided by the companies to share their data by integrating with other applications. Examples could be Geolocation API, RSS feed and Atom of websites.

### 2.1.5 Some Popular Mashups

The following describes some popular Mashup:

The **IBM Damia** was developed by the IBM Corporation to support the daily business needs by integrating company's internal and external data obtained from various web services. It consists of feed server and web based client interface where the feed server combines and aggregates XML data from various sources, and the client interface allows various modules to obtain and process data to meet business needs [36].

Page 20 of 85
**Microsoft Popfly** is a Mashup creator tool based on Silverlight technology which co-ordinates the same type of data blocks from different web services to meet user needs. An example could be Geo Tagged Map picture [37].

**Nokia Taller Mashup** is a server and web client based obile Mashup developed by Nokia which integrates a user location into Google-Map. Here Module attached with the web client provides GPS data of a user to the server to identify a user location, and in its turn, the server provides a Google map with location information in response by combining XML and image data provided by Google incorporation [35].

<table>
<thead>
<tr>
<th>Mashup</th>
<th>Types</th>
<th>Architecture</th>
<th>Presentation</th>
<th>Web Services/method</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM Damia</td>
<td>Business</td>
<td>Client &amp; Server</td>
<td>Html</td>
<td>XMLHttpRequest</td>
<td>XML</td>
</tr>
<tr>
<td>Microsoft Popfly</td>
<td>DATA</td>
<td>web application</td>
<td>Ajax</td>
<td>XMLHttpRequest</td>
<td>XML</td>
</tr>
<tr>
<td>TELAR Mashup</td>
<td>Consumer</td>
<td>Client &amp; Server</td>
<td>Html</td>
<td>XMLHttpRequest</td>
<td>XML</td>
</tr>
<tr>
<td>IBM lotus Mashup</td>
<td>Business</td>
<td>web application</td>
<td>Html</td>
<td>XMLHttpRequest</td>
<td>XML</td>
</tr>
</tbody>
</table>

**Table 1:** Mashup chart

### 2.2 SUMMARY

The above description contains basic properties, types and styles of a Mashup that is needed to develop a Mashup framework. In addition, some popular Mashup technology is shortly described.
3. SERVICE PROVIDER, WEB SERVICE AND PROTOCOL

3.1 SERVICE PROVIDER

The main goal of the service provider is to provide services which could be document, data, image and video. In the context of Mashup application, the service providers are the main source of data. Therefore, the service provider can be defined as a third party application those provides contents or data to the Mashup application. Naturally, a service provider hosts the servers in a remote place with interface specification. A machine processable interface specification to share resources with other service called Web Service Description Language (WSDL). In addition, this XML based Web Service Description Language (WSDL) describes the communication message formats, data types, transport protocol [3]. In this consequence, SOAP uses WSDL file to exchange messages and UDDI to discover the services. Universal Description, Discovery and Integration (UDDI) is a platform independent, XML based registry system which list and publishes the SOAP for all business organization [3]. On the other hand service providing through the REST is a popular protocol to publish the services at present. In addition, the service provider registers a user with a secret key to use their service. For using the REST service there are the Terms of Service (ToS), those should be handled to use these services. Moreover, the OAuth protocol is one of the most popular techniques to share private information of one service with the other service. It provides a token instead of password to the service user. User needs to register with the REST service provider to get the token. In this mechanism user information is protected by the service providers.

3.2 WEB SERVICE

Web service is a software system designed to support interoperable machine-to-machine interaction over the network [3]. This concept is based on the requesters and the service providers where a requester requests a certain service to the provider, and a provider provides services on the basis of the request. This request and response depends on the exchange of messages where the communication platform of service provider and service requester can be different. To make this message exchange successful, both the requester and the provider are
needed to agree on the semantics and the mechanics. Moreover, documentation about the mechanics is kept in the Web Service Description (WSD).

3.3 PROTOCOLS

A Protocol describes the way of communication with the service providers. In this study REST and SOAP is considered as 89% web service provider uses these [29]. Therefore, SOAP and REST are described as below:

3.3.1 Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is a protocol for exchanging messages between a supplier and a consumer in a web service communication using XML [5]. It could be used for XML remote procedure call by adding some layers. These layers provide message security and add more functionality than normal remote procedure call. It can be used in a large variety of the systems ranging from the message system to the RPC. The parts of the SOAP could be described as below:

**SOAP envelop** is the top and mandatory element that express the message body, participant and if it is an optional or mandatory. In addition, the **SOAP encoding rules** describe the serialized mechanism that can be used to define exchangeable application data types. In the end, the **SOAP RPC** defines the convention of remote procedure call. The SOAP also uses a namespace to express simplicity and modularity of the service [5]. Moreover, it uses HTTP and SMTP protocol to exchange messages between a Requester and a Service provider over the network. It also uses header to add more features to the message. Before receiving a SOAP message to the SOAP process, it follows the following actions [5]:

1. Identify all parts of the messages of the application
2. Verify all identified mandatory parts of the message whether these are supported by the application or not.
3. Verifies the destination of messages and discards unwanted messages before forwarding the message to the final destination to reduce the client authentication load in the server.

The message exchange rules [5] followed by the SAOP are as below:
1. Understanding the message exchange pattern between the client and the service provider
2. Understanding the recipients roles in implementing RPC mechanism
3. Represent or encode other symmetric necessary for the encoded data.

**TECHNICAL CRITIQUE**

The SOAP allows the HTTP, SMTP and JMS to exchange the messages. It also makes tunnel over existing firewalls and proxies without modification to the SOAP protocol as the SOAP model makes tunnels in the HTTP get/response model [6]. In contrast, the SOAP may be considerably slower than the computing middleware technology due to the verbose XML format, though it may not be an issue in case of small messages communication. In addition, the roles of interacting parties are fixed without using WS-Addressing when rely on HTTP as a transport protocol. In this case, only one party can use the service of the other [6].

### 3.3.2 REPRESENTATIONAL STATE TRANSFER (REST)

In short term, RESTful service is called Representational State Transfer (REST). It is a software architectural style for distributed hypermedia [8]. It demonstrates stateless client server architecture. In this context, the service provider is a server and a requester request for a service over HTTP. The service providers are identified by their URLs. A client communicates with a service provider using these URLs by using HTTP protocol as the under layer. Initially it was described by HTTP but it is now improved and can work on the different layers. In addition, the HTTP is very rich in terms of the URL, internet media type, request and response code. It is also able to add new proxy and gateway in the communication layers to allow additional functionality such as HTTP caching and security enforcement because of using existing features of the HTTP [8]. The features of the REST architecture could be described as below:

1. **Client–server**

A client and a server are separated from each other through a communication interface, where server and client are not concerned about each other's internal mechanism [8]. Therefore, their performances and features could be individually improved by the modification of their internal code without changing the interface.
2. **Stateless**

The communication between a client and a server should be stateless. A client context should not be stored in the server. Every request should hold all the necessary information to make a proper response. Thus, there is no need of session or other things in the server side to make the server scalable. On the contrary, some exceptions may take place for the use of the REST, where a state should be kept in the client side. In contrast, a server can work as stateful by keeping the state information in the URL. In addition, the Stateless can improve visibility, reliability and scalability of the communication properties where visibility is improved by forbidding the monitoring system to look beyond the data request which can recover partial failure of the network by keeping the states in URL. The Scalability could be improved without keeping the client session in the server but keeping the session in the URL [8].

3. **Cacheable**

The REST architecture has the Cache-able property which improves the network efficiency by a Cache constraint architecture that works as the client-cache-stateless-server style. It means a cache is added in the communication of the client-stateless-server which acts as the mediator between a client and a server. For this reason, the response of a prior request can be reused later. It helps to improve performance and efficiency of the servers by reducing the average latency of a series of interactions [8]. It is also capable to eliminate some client-server interaction partially or completely by using a well managed caching.

4. **Layered system**

In a Layered system, a client cannot know whether it is connected directly to the server or not as it cannot see the layers beyond it uses. This layered system encapsulates the legacy services and protects a new client form a legacy client. Moreover, for a reliable transaction, an intermediary could be used to improve the system scalability by providing shared caches as well as enforcing security policy [8].

5. **Code on demand** is an optional constraint where a server transfers extended logic to the client through an applet to extend or customize the client functionality [8].
6. Uniform interface

The REST architecture uses uniform interface in the client server communication to simplify the overall system architecture and improves the visibility of the system interaction. It is efficient for the large-grain hypermedia data transfer to optimize common cases of the web [8].

3.3.3 WEB API

A Web API (Application Programming Interface) is a set of Hypertext Transfer Protocol (HTTP) request along with a definition of the structure of response messages [9]. Primarily it was focused on the SOAP but from the introduction of REST the trends became changed. It uses XML or JSON to express the response message.

3.3.4 COMPARISON OF SOAP AND REST FOR MOBILE WEB SERVICE APPLICATION

1. COMPARISON OF SOAP AND REST FOR MOBILE WEB SERVICE APPLICATION IN RESPECT TO USABILITY

The following discussion shows a comparison between SOAP and REST in respect of usability:

In uses of REST

1. The Client server interaction can survive at the re-start of the server in REST since the web services are stateless [7].
2. A caching infrastructure can leverage performance when data returned from the web service are not dynamically generated and cached [7].
3. The Value added toolkits provided by the service provider in the REST contains an interface description for the developer to use meaningful resources [7].
4. REST performs better than SOPA in limited bandwidth. That is why; REST is applicable for mobile phones [7].

In uses of SOAP

1. The SOAP establishes a formal contract to describe the interfaces of web service where the WSDL describes the message format, operations, bindings and location [7].
2. The SOAP needs complex non-functional requirements such as Transactions, Security, Addressing, Trust, Coordination etc [7].

2. **Comparison of SOAP and REST for Mobile Web Service in Respect to Server**

SOAP and REST are used to enable communication for message passing between a service provider and a client. The server receives a service creation request through its HTTP listener from the client. It delegates the request to the request handler. A Request handler parses the request and extracts the name and the context data of the requested service. The deployment interface receives a service information from the request handler to create a service object in return [10]. The following facts [10] are considered to analyze the performance of a web server:

1. **Server Utilization:** “Server Utilization is the percentage of time during which the server is busy processing jobs. It is computed as the ratio of the request arrival rate and the service rate. For a stable system, the value of this ratio must be less than 1 (100 %), that is, the request arrival rate should not exceed the service rate.”

2. **Average Waiting Time:** “This is the time an arriving request has to wait in a queue until the server gets free and starts processing it.”

3. **Average Throughput:** “Throughput is the number of the service creation requests processed per unit time. It should not be confused with the service time, as the service time does not include the waiting time while it does.”

4. **Average Queue Length:** “It is the average number of the service creation requests in queue at a certain time.”

The experiment is done in respect of the mobile web service considering the above factors. The outcomes of the experiment are as follows [10]:

- Average server utilization in SOAP is higher than REST.
- Request waiting time from SOAP is higher than REST.
- Through put of REST is higher than SOAP.
- Average queue-length of SOAP is higher than REST.
Therefore, REST performs better than SOAP [10].

3. **Comparison of SOAP and REST for Mobile Web Service in Respect to Mobile Devices:**

Web service connects mobile systems to a conventional distributed system. It uses inter-portable capability with the heterogeneous hardware platform and operating systems. In contrast, the mobile phone has limited processor-speed, limited battery life time, slow and unreliable Internet connection[11]. These are constrains of the mobile phone. Therefore, a web service enabled client should overcome these limitations with acceptable performance overhead. In contrast, poor performance of a mobile client may occur for encoding and decoding of verbosity in a message, and the performance and the quality gap between wireless and wired communication [11].

In a web service, a client sends a request to the server and the server parses the request according to instructions. The performance of the return response depends on the encoding and decoding technique of the server. Message response time of a service call includes communication setup latency; transmission over head and concatenation process time. Therefore, a study was carried out in respect of SOAP and REST to evaluate their performance in response time of service call. The outcomes show that the message size for the SOAP is higher than the REST to encode and decode, and the Message response time of SOAP is higher than the REST [11]. As a result, the REST utilizes less power with faster web service communication because of using less communication messages. Therefore, REST is better than SOAP in mobile device[11].

Some mobile operating systems and their corresponding Protocol uses can be described as follows:

<table>
<thead>
<tr>
<th>Mobile phone</th>
<th>Operating system</th>
<th>Tool kit</th>
<th>Protocol/Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>I phone</td>
<td>iOS</td>
<td>iOS SDK</td>
<td>SOAP,REST</td>
</tr>
<tr>
<td>Blackberry</td>
<td>BlackBerry OS</td>
<td>MIDP 2.0</td>
<td>SOAP,REST</td>
</tr>
<tr>
<td>Sony Ericson, HTC, Samsung, LG</td>
<td>Android</td>
<td>SDK, NDK</td>
<td>SOAP, kSOAP2, JSON RPC</td>
</tr>
<tr>
<td>Nokia</td>
<td>Symbian</td>
<td>Qt SDK, MIDP</td>
<td>SOAP, REST</td>
</tr>
<tr>
<td>Qualcomm</td>
<td>BREW(qual kom)</td>
<td>Brew MP SDK/MIDP 1.0</td>
<td>SOAP, REST</td>
</tr>
</tbody>
</table>
3.5 **Summary**

Mashup uses a web service to receive data from several service providers through message transformation within the client and the server. A client receives data from the service provider after interface communication between them, called the service call. Both SOAP and REST are used to communicate with the service provider and the service consumer or client. Moreover, SOAP is Simple object access protocol where XML used as communication media and web service description language (WSDL). On the other hand, the REST defines an architecture which can use XML or JSON as the communication media and it also performs better in the server and the mobile device (3.3.4). Therefore REST is considered as the communication component for the mobile Mashup Reference Framework.

Below matrix shows the limitation, strength and comments on the experiments held in sec 3.3.4.

<table>
<thead>
<tr>
<th>topic</th>
<th>Experiment</th>
<th>result</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP vs REST(3.3.4)</td>
<td>Usability</td>
<td>Rest is good than XML in respect of Client server interaction, caching, bandwidth uses.</td>
<td>It determines network performance.</td>
</tr>
<tr>
<td>Mobile web service in respect of server</td>
<td></td>
<td>REST is good in respect to Average server utilization, request waiting time, throughput, average queue length</td>
<td>It determines server side performance.</td>
</tr>
<tr>
<td>Mobile Web service in respect of mobile device</td>
<td></td>
<td>Message size of SOAP is higher than REST to encode and decode, message response time of Soap is higher than REST, causes faster web service communication and uses less power.</td>
<td>This experiment defines performance in respect of mobile handset provider.</td>
</tr>
</tbody>
</table>

**Table 3: Summary and comments on experiments over protocols**

**Note:** These performances table 3 tell that REST is suitable for mobile handset, server and network utilization.
4. Data, Parser and Framework

Web service uses data formats to exchange messages between the client and the service provider. In this Communication, XML and JSON are widely used [14]. Therefore, the following discussion will provide a simple overview of JSON and XML.

4.1 XML

Extensible markup language (XML) is defined as a set of rules that should be used to encode documents. This document should be machine readable and is made of Storage Units called Entities which contain data to be parsed along with unparsed data. The parsed data contains characters and markup. A Markup encodes document storage layout and provides constraints on the storage layout and the logical structure [13]. Furthermore, some features of XML could be described as below [13]:

1. It is straight forwardly usable over the Internet
2. It supports a wide variety of application
3. Provide uncomplicated XML processing program
4. Human can read XML clearly
5. The design of XML is formal and concise

Furthermore, XML has a good security in the form of serialized data and tag scalability. Tag scalability helps the developer to configure web services application depending on their characteristics and adapt different situations. [14]. Yet, XML is verbose, complex and self-descriptive [12]. It takes some character to define the tags, the description of XML, version name and name space. It is difficult to map XML to a programming language or database. XML objects are analyzed as DOM which takes long time to parse [14].

4.2 JSON

JavaScript Object Notation (JSON) is a text based light weight data exchange format [14]. It is derived from java Script to represent data structure and associative array. This data structure is called object [15]. These objects are analyzed as a string array. Furthermore, JSON consists of a collection of name/value pairs and an ordered list of values.
All modern languages support JSON for data interchange. It is called well matched data interchange format as it has the following goals [17]:

1. It can be easily read by both humans and machines.
2. It is allowed to store every kind of information by using native support of Unicode.
3. Self-documentation format is added here to describe the structure and content.
4. It has strict syntax that helps to keep the parsing algorithm simple and efficient.
5. It has ability to represent records, lists and trees.

In contrast, JSON has the following limitations [15]:

1. It does not contain namespaces.
2. There exists no validator.

Every JSON object has a unique set of keys which meets the purpose of namespaces like XML [19]. Every individual domain application validates JSON and construction flexibility makes it extensible [18].

4.3 COMPARISON BETWEEN XML AND JSON

The following comparisons between XML and JSON describe why JSON can be used to replace XML:

1. Simplicity

JSON can map more directly to the data structure than XML because of its smaller grammar where grammar helps to make data format machine and human readable [16].

2. Extensibility

Data can be represented in JSON format without defining a new tag or attribute where as XML needs to define tag. So JSON data is easy to extend [16].

4.3.1 PERFORMANCE ANALYSIS OF XML AND JSON IN RESPECT OF ALLOCATION OF TIME AND RESOURCES ALLOCATION
An experiment was carried to evaluate performance between XML and JSON in respect to resource utilization and data transmission where time indicates which data format is faster in data transmission and resource utilization includes user CPU utilization, System CPU utilization and memory uses. User CPU utilization describes that some process should be done before calling an application. On the other hand, System CPU uses indicate how much CPU is utilized to execute the application in kernel. The experiment was carried out considering the following scenario [18]:

1.”The first scenario consists of running a single time-consuming transmission of a large quantity of objects in order to achieve accurate average measurements.”

2.”The second scenario consists of running a series of test cases with increasingly higher number of objects.”

To conclude the experiment, the outcomes show that JSON is faster and uses fewer resources than XML including user CPU utilization, System CPU utilization and memory uses [18].

Following matrix demonstrates a short overview of the operating systems and their supported data format:

<table>
<thead>
<tr>
<th>Mobile phone</th>
<th>Operating system</th>
<th>Tool kit</th>
<th>Data Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>I phone</td>
<td>iOS</td>
<td>iOS SDK</td>
<td>XML,JSON</td>
</tr>
<tr>
<td>Blackberry</td>
<td>BlackBerry OS</td>
<td>MIDP 2.0</td>
<td>XML,JSON</td>
</tr>
<tr>
<td>Sony Ericson, HTC, Samsung, LG</td>
<td>Android</td>
<td>SDK, NDK</td>
<td>XML,JSON</td>
</tr>
<tr>
<td>Nokia</td>
<td>Symbian</td>
<td>Qt SDK, MIDP</td>
<td>XML,JSON</td>
</tr>
<tr>
<td>Qualcomm</td>
<td>BREW (qual kom)</td>
<td>Brew MP SDK/MIDP</td>
<td>XML,JSON</td>
</tr>
<tr>
<td>Hp</td>
<td>Palm os</td>
<td>PDK/MIDP</td>
<td>XML,JSON</td>
</tr>
<tr>
<td>HTC, Samsung</td>
<td>Windows phone 7</td>
<td>.net framework</td>
<td>XML,JSON</td>
</tr>
</tbody>
</table>

Table 4: Mobile OS and supported data format
4.4 Processing Data Format

Data format needs to be processed to represent the data in front of the requester. The Web services use XML and JSON data formats widely. Different technologies are used to process these two data formats. A short description of data processing techniques are follows:

4.4.1 XML Parser

XML uses variety of approach to do data processing. Existing approaches for processing data formats are as bellow [12]:

1. Stream oriented API, e.g. SAX.
2. Tree traversal API, e.g. DOM.

The XML parser uses the following steps to parse the document [21]:

- Character conversation converts character sets of XML document to understandable language for the destination program.
- Lexical analysis step makes portion of character streams into a sub-sequence called tokens which are expressed by a regular expression. For this reason, a finite state machine is used to match the regular expression.
- Syntactic analysis verifies well-formation of the data file by making sure that the tokens are properly nested tags or not.

In addition, a short description of some XML parsers such as DOM, SAX, StAX and VDT could be described as below:

1 DOM

DOM creates a tree object by using many node objects. Here element name, attributes, name spaces and pointer to indicate the parent-child-sibling relationship are stored in each node. DOM uses long lived structural data for a sophisticated operation in the stage of access and modification. It parses data by loading full data in the memory which may cause delay and impact on memory size [21].
2 SAX AND STAX

SAX uses a push model with a callback function to report events from the function to the applications where StAX uses a pull model by skipping uninterested elements. Both of them create a sequence of events by not creating long-lived data structures. They create objects for data representation and associate different objects with different events [21]. Their contents are reported as callbacks to various methods on user-defined handler objects by reading documents in a serialized manner as they are lexical event-driven interfaces [20]. They also have the ability to access partial data before parsing the document completely and destroy associated events objects regularly to stop increasing memory size along with the document size [20].

3 VTD

VTD creates an integer array and uses a long-lived data structure for sophisticated operations. It stores the original document and produces a 64-bit integer array called VTD records (VRs) and location caches (LCs). Here VRs keep token positions in the original document where LCs store the parent-child-sibling relationships. The VTD can also parse data after all the data is loaded which may cause delays for large documents as memory use increases with the growth of document size [21].

4.4.2 COMPARISON AMONG DIFFERENT XML PARSERS

The stream-oriented task is based on the linear traversal process which needs low latency and uses low memory. It can parse some portion of the document sequentially without knowing the entire document structure. On the other hand, the tree traversal and data binding API needs more memory as it uses a parser after the document has been stored in memory [21]. That is why, it can be concluded that SAX is better than DOM and VTD as it uses a stream-oriented API.

An experiment was done before to find out the performance analysis of the XML parsers. The outcome of the experiment shows that DOM and VTD are good for back-and-forth data access where VTD is better for simple and rare modification of data and DOM is good for complex and frequently changing data. VTD is faster and uses less memory than DOM [21]. On the other hand, SAX and StAX use less memory and resources which are applicable for extremely restrictive memory and for streaming applications [21].
4.4.3 JSON PARSER

The JSON parser converts JSON documents into JSON objects and keeps them in a JSON array at the time of parsing. In this parsing method, a parent object works as the root element of the JSON document.

4.4.4 GSON

GSON is an open source JAVA library which converts a JSON document into a JAVA object and vice versa. In this parsing method JAVA objects are created for every node of the JSON document which provides data by accessing itself. The objectives of GSON could be described as below [22]:

- Uses easy mechanism to convert JAVA object and vice-versa.
- Pre-existed un-modifiable objects are allowed in object and JSON conversion.
- Custom representations of objects are allowed.
- Complex objects are supported.
- Generated JSON outputs are compact and readable.

The below matrix shows mobile operating systems and supported parsing methods:

<table>
<thead>
<tr>
<th>Mobile phone</th>
<th>Operating system</th>
<th>Tool kit</th>
<th>Data Format</th>
<th>Parser</th>
</tr>
</thead>
<tbody>
<tr>
<td>I phone</td>
<td>Ios</td>
<td>Ios SDK</td>
<td>XML,JSON</td>
<td>DOM, SAX, JSON parsing Api</td>
</tr>
<tr>
<td>Blackberry</td>
<td>BlackBerry OS</td>
<td>MIDP 2.0</td>
<td>XML,JSON</td>
<td>DOM, SAX, JSON parsing Api</td>
</tr>
<tr>
<td>Sony Ericson, HTC, Samsung, LG</td>
<td>Android</td>
<td>SDK, NDK</td>
<td>XML,JSON</td>
<td>DOM, SAX, JSON parsing Api, Gson</td>
</tr>
<tr>
<td>Nokia</td>
<td>Symbian</td>
<td>Qt SDK, MIDP</td>
<td>XML,JSON</td>
<td>DOM, SAX, JSON parsing Api</td>
</tr>
<tr>
<td>Qualcomm</td>
<td>BREW(qual kom)</td>
<td>Brew MP SDK/MIDP 1.0</td>
<td>XML,JSON</td>
<td>DOM, SAX, JSON parsing Api</td>
</tr>
<tr>
<td>Hp</td>
<td>Palm os</td>
<td>PDK/MIDP</td>
<td>XML,JSON</td>
<td>DOM, SAX, JSON parsing Api</td>
</tr>
<tr>
<td>HTC, Samsung</td>
<td>Windows phone 7</td>
<td>net framework</td>
<td>XML,JSON</td>
<td>DOM, SAX, JSON parsing Api</td>
</tr>
</tbody>
</table>

Table 5: supported parsing methods by Mobile OS
4.5 Framework

A framework is an architectural design for object oriented systems which describes the components of the system and the way they interact [23]. Here components of a framework reduce the difficulties of application development. This difficulty may contain application environment constrains and requirement constrains. The key features of a framework could be described as below:

- **Modularity**--Framework uses stable interfaces where it encapsulates volatile implementation details which help to enhance modularity. It also helps in improving the software quality by localizing the design and implementation impacts [24].

- **Re-usability** -- Framework provides re-usability by using its generic components to the various application domains to improve productivity, performance quality and interoperability of software [24].

- **Extensibility** -- Framework provides extensibility by overriding its methods to add more features to the main components. It provides hook methods for extending static interfaces of applications which decouples stable interfaces and behavior systematically to meet application needs in particular context [24].

- **Inversion of control** -- Framework controls the flow of data and commands of a program by using its components. These components follow working mechanisms of the systems and operated by calling each-other [24].

In addition, the architecture and design of the framework depends on its uses. Therefore classifications of framework in respect of its uses are described below:

- **System infrastructure frameworks** -- A portable and efficient system infrastructure is the concern area for this type of framework. Software organization uses this internally to simplify their system development [24]. Examples could be operating system and Communication framework etc.
- **Middle ware integration frameworks** -- This kind of framework enhances the ability of software developers to modularize, reuse and extend software infrastructure which are commonly used to integrate distributed applications and components. This helps the developer to work seamlessly in a distributed environment [24]. Examples could be Object Request Broker (ORB) framework, transactional database framework and message oriented middle ware etc.

- **Enterprise application frameworks** -- This type of framework is applicable for a broad application domain and enterprise business activities which supports the end user applications development. Yet it is expensive to develop and purchase [24]. Examples could be Telecommunications, manufacturing, financial engineering etc.

4.6 **DEVELOPING PLATFORM ANALYSIS**

Smart phone uses advanced computing and network connectivity facilities for data communication and data processing. These phones also have the ability to run multitasking applications that are native to the underlying hardware. It helps the developer by providing a complete operating system along with a mobile application development platform. Some Smart phones and their details can be described as below:

**The Iphone** is a line of an Internet and multimedia enabled Smart phone designed and marketed by the Apple Inc. It uses IOS as its operating system and its application development language is Objective-C which is an object oriented programming language made on top of the C language. It follows the message passing style and interprets the request and response message at the run time.

**Blackberry** is a line of email enabled Smart phone which functions as a personal digital assistant and portable media player. It uses the BlackBerry OS as operating system and the Java mobile version MIDP as its application development environment.

The new generations of **Nokia Mobiles** are capable of running a mobile application and multimedia. It uses the Symbian as the operating system and C++ as its application development language. It interprets the syntax very easily to communicate with hardware level and also supports java mobile edition MIDP.
**Qualcomm** is a line of CDMA multimedia enabled mobile phones which uses the BREW mobile operating system. It gives opportunity to C, C++ developer by providing the Brew MP SDK. It also supports JAVA ME.

**Android** is a software stack for mobile devices including an operating system based on Linux kernel. It supports the Java-based object-oriented software framework running on a Delvic virtual machine with JIT compiler which is developed for a limited memory and processor speed. Additionally, the Android application developers write application codes using Google-developed Java libraries supplied with the SDK. This SDK includes debugger, libraries, and the emulator. It also provides an environment for C and C + +developers by offering NDK. Currently Android is popular because many famous mobile hand set manufacturers use it, i.e. Sony Ericsson and Samsung .The matrix below shows the operating systems and corresponding development language:

Windows phone is an operating system provided by Microsoft. It’s development tools based on .net framework. Therefore it supports c# and vb.net to develop application and with Silverlight and Microsoft expression technology for graphical user interface.

<table>
<thead>
<tr>
<th>Mobile phone</th>
<th>Operating system</th>
<th>Tool kit</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>I phone</td>
<td>IOS</td>
<td>IOS SDK</td>
<td>Objective-C</td>
</tr>
<tr>
<td>Blackberry</td>
<td>BlackBerry OS</td>
<td>MIDP 2.0</td>
<td>J2ME</td>
</tr>
<tr>
<td>Sony Ericson, HTC,Samsung,LG</td>
<td>Android</td>
<td>SDK,NDK</td>
<td>JAVA,C,C++</td>
</tr>
<tr>
<td>Nokia</td>
<td>Symbian</td>
<td>Qt SDK,MIDP</td>
<td>C++/java ME</td>
</tr>
<tr>
<td>Qualcomm</td>
<td>BREW(qual kom)</td>
<td>Brew MP SDK/MIDP 1.0</td>
<td>C,C++/java ME</td>
</tr>
<tr>
<td>Hp</td>
<td>Palm OS</td>
<td>PDK/MIDP</td>
<td>C,C++/java ME</td>
</tr>
<tr>
<td>HTC, Samsung</td>
<td>Windows phone 7</td>
<td>.net framework</td>
<td>C#/VB.net</td>
</tr>
</tbody>
</table>

Table 6: Mobile operating systems and development language
4.7 SUMMARY

The cited research proves that JSON is better than XML in respect of structural design, usability and utilization of system resources and data transmission (4.3). Therefore, JSON is suitable for mobile communication and mobile devices.

Section 4.4 presents details about the parsing mechanism and performance analysis of various parsers which provides an account of the idea about their application circumstances. Furthermore, the performance analysis shows that SAX is applicable to less memory and resource oriented devices. In contrast, no performance analysis have described among SAX, JSON and GSON.

Section 4.5 provides details information about framework and its classification and fetures.

Section 4.6 presents that; the Android operating system is the most frequently used by cellular phone manufacturers. This is a platform that can easily convert the code into machine-readable form. Therefore, Android is considered an operating system and application development language for this study.

Below matrix shows the limitation, strength and comments on the experiments held in sec 4.3 and sec 4.4.3.

<table>
<thead>
<tr>
<th>topic</th>
<th>Experiment</th>
<th>result</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML vs JSON (4.3)</td>
<td>Usability</td>
<td>JSON is good in respect to Simplicity and extensibility as it does not needed tags.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Network performance</td>
<td>JSON is faster than XML in data transmission.</td>
<td>Network performance will be good for using JSON.</td>
</tr>
<tr>
<td></td>
<td>Resource utilization</td>
<td>JSON uses less System CPU, CPU utilization and memory.</td>
<td>Performance evaluated in respect of client end.</td>
</tr>
</tbody>
</table>

**Note:** We can get information about transmission performance information as other experiment does not uses mobile device or emulator but we can assume that JSON will take less resources than XML. So it is needed to
perform the experiments in mobile platform.

<table>
<thead>
<tr>
<th>XML parsers</th>
<th>SAX is applicable for restrictive memory and streaming application.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4.4.3) LE</td>
<td>DOM, VTD and SAX</td>
</tr>
</tbody>
</table>

Note: Here SAX is considered, as Streaming API parses some portion of document without knowing the entire document. It works on linear traversal which needs low latency and uses low memory. But DOM and VTD needs to store full document in the memory before parsing. Though there is no performance analysis experiment held in mobile devices considering the JSON parser and GSON parser in comparison with SAX parser.

Table 7: Summary and comments on experiments of data processing
5. IMPLEMENTATION

5.1 DESIGN MOTIVATION AND ASSUMPTIONS

Nokia tellar Mashup uses client-server architecture where IBM Lotus and Yahoo pipe use a rich client platform to make the Mashup (2.1.5). These Mashup applications process data using computers which have more processing capabilities than the mobile phone. However, the technological development of mobile phone and mobile communication gives an opportunity to develop Mashup application. In the field of mobile Mashup, many Web Mashup frameworks have been proposed depending on web API. Though, there are many web service provider those provide responses in XML and JSON data. Therefore, these APIs provide final output of a particular requirement by keeping a processing mechanism hidden. That is why, a reference framework which can handle raw data along with the web APIs is considered in this study.

5.2 SYSTEM ARCHITECTURE

Here the client/web based Mashup architecture (2.1.2) is used to make the reference framework for mobile Mashup. The innovation of the 3G mobile technology is the main reason behind this, as it allows huge data bandwidth in mobile devices.

Furthermore, the technological development of the mobile processor and power capacity has given opportunity to develop a Mashup application in the Mobile devices. Consequently, a Mobile phone supports a minimum data rate of 2 Mbit/s for the stationary or walking user and 384 kbit/s in a moving vehicle for the 3G technology [25]. The considered Mobile Mashup style is defined below to show that a cell phone device contains the Mashup application which uses GPS information and collects data from the various resources using the Internet.
5.3 ASSUMPTIONS

5.3.1 LIMITATION OF MOBILE PHONE AND SOME SUGGESTED SOLUTIONS

A mobile phone is a small device with limited resources in respect of power consumption, data processing capacity and memory in comparison with a computer. To apply a native application in the mobile phone, following strategies are described below to overcome these limitations:

1. USE OF RADIO TRANSMISSION REDUCTION

The Radio transmission equipment of the mobile phone consumes a lot of power [26]. Naturally, a mobile Mashup uses this radio device for data communication between a mobile device and a service provider. Therefore, the REST is considered here as the communication protocol, because it uses less messages than the SOAP to establish communication between a requester and a service provider (3.7.3). Furthermore, the REST supports both the XML and the JSON data format whereas the SOAP supports only the XML.

2. REDUCTION OF THE EXECUTION TIME AND RESOURCE USES IN MOBILE DEVICES

The mobile Mashup application receives data from various resources and parses these before representing on the mobile display by using the CPU and the Processors. Naturally the processing units consume power and memory during the data processing. Therefore performance analysis is taken in section 6 to find out the appropriate parser for the JSON and the XML. To narrow down the experiment, section 4 describes the parsing technology of the XML parser and the JSON parser. It also shows that the SAX parser works better for the limited resource enabled devices (4.10) among DOM and VTD. The SAX parser, JSON parser and GSON are therefore considered for the experiment.
5.4 Layered Architecture

Fig 5 below shows the Layered architecture of client based Mashup style in mobile devices which takes data from the service providers. Here the communication protocol communicates between the mobile phones and the service providers. Additionally the data processing and the representation units represent the data after parsing and processing.

![Layered Architecture Diagram](image)

Fig 5: Layered architecture

5.5 Platform Choice

At present many mobile applications play decisive role in the market. Among them the Android plays a good role in respect of market share and rich-full class library (4.6). Android has following features:

1. **Kernel**: The Android operating system uses the Linux Kernel 2.6 which handles the core system services and also acts as the hardware abstraction layer between the physical hardware and the android software stack. The core functions that are handled by the kernel are [27]:
   1. Application permissions and security enforcement.
   2. Low level memory management.
   3. Thread and process management.
   5. Able to handle the devices included with mobile phones such as display, keypad input.

2. **Android Application Run time Environment**
The Android application uses the Delvic virtual machine to run the applications which is based on the JAVA VM. It uses small memory and runs every application in a separate process. It drives multiple instance of the Delvic VM concurrently in a mobile hand set [27].

3. Security and Permission
Verities of secured measures are used to confirm the integrity of the Android platform. It ensures the user data security and the mobile device malware [27].

4. Explicitly Defined Application Permissions
The Android application needs to register for the specific privileges to access the shared resources of the mobile system. These permissions are read-only or read-write. These kinds of permission help to control over applications and the privileges help to enable the phone functionality that is used by the applications [27]. Example: Making calls, access network. These permissions are also needed to access the shared data containing private and personal information. Example: Preferences, user’s location and contact information [27].

5. Limited Ad-Hoc Permissions
This type of permission helps applications, those are acting as content provider and provide some on the fly permission to other applications for specific information they want to share [27].

6. Application signing for the trust Relationships
Applications are signed by the public key of the developer to make authentic and secure applications. This also helps to control the system resources which should be shared with others and increases trust relationship between the user and the developer [27].

7. Developing Android Applications
The Andorid SDK provides robust and modern sets of APIs to access the core system services of a mobile system to make rich-full applications [27].

8. Commonly Used Packages
The developer can use common rich-full class libraries exposed through the Android JAVA packages to perform tasks. Example: graphics, database access, network access etc [27]. Therefore it can be conclude that, the Android is more convenient application development platform than other mobile application development platforms. So android is chosen in this study to test the components of mobile Mashup.

5.6 System Design
Mashup application receives data from service provider by sending a request message through protocol and receives response in XML and JSON data format. Therefore, components for communication protocol, GPS data and parsers were developed whereas the web API needs API keys and other specifications which are provided by the service provider.

The mobile Mashup architecture of this study shows (5.2) that the mobile client sends input parameters to the service provider through the REST. Here, the GPS location provider component is used to get the latitude and longitude of the user from the GPS service provider which is the input parameters for the request. The reply message contains the data in the XML or the JSON format. Therefore, the SAX, JSON parser and GSON are implemented to parse data.

Components details and their working procedures are described as below:

5.6.1 GPS LOCATION PROVIDER

The following APIs are used to make the GPS location component:

The LocationManager class accesses mobile system location services which provide periodic updates of user’s geographic location [41].

The getSystemService() method returns handler to a system service by name using context [41].

The Context is provided by Android system to access application specific resources and classes. It also helps to call up application level operations. Example: lunching activities and broad casting etc [41].

The object of the Criteria class is used to define conditions to select a best service provider. Example: criteria should be implemented in respect to accuracy, power usage, ability to report altitude, speed.

The getLastKnownLocation() method provides last location determined by the device from the location service provider [41].

The Location class provides latitude and longitude of the user’s geographic location at the current time.

The getLatitude() provides the latitude data of a specific location.

The getLongitude() provides the longitude data of a specific location.
Working procedure

Here the locationmanager uses getSystemService() method and context parameter to make a locationmanager object. In addition, the Provider object is created to provide the best GPS service provider using criteria parameter. At last a Location object is created using getLastKnownLocation() method of the locationmanager object and provider object as parameter containing latitude and longitude of that location. The getLastKnownLocation() is used to get last known location of the user. Programmer uses getLatitude() and getLongitude() to collect latitude and longitude information whenever they need.

5.6.2 RESTful Service Provider Component

This study chooses REST service to make communication between a mobile phones and service provider. Therefore, the REST client is implemented here, details of which are described as below:

The following APIs are used to make the REST function component:

The HttpClient executes http request to handle cookies, authentication and connection management. It also provides thread safety through implementation and configuration of the specific client by working over the http protocol [41].

The HttpPost requests original server to accept the entity enclosed in the request [41]. This request works as a subordinate of the resource identifier, which is identified by the Request-URI in the Request-Line. It covers the following function:

- Annotation of the existing resources
- Posts a message in a bulletin board
- Gives a block of data after submitting the form to handle process

The HttpGet retrieves information identified by the Request-URI. It also provides response after processing the data if the URI contains the data processing information [41].

The HttpResponse is the final response of request which means, it never provides intermediate response. It gets this response by executing httpclient using the get or post method. The Methods involvement with http client depends on implementation and configuration [41].
This **HttpEntity** is used with the HTTP to send and receive message. Three kinds of http entity are described [41] as below:

- **Streamed**: This entity receives data from the http connection which is unique and non-repeatable.
- **Self-contained**: This is a repeatable entity.
- **Wrapping**: The content of this entity is obtained from another entity.

The **Input stream** reads data from the source in byte wise manner.

**Working details**

Here HttpClient uses the HttpGet method to make an instance of HttpResponse using URL as parameter. The URL contains necessary input parameter according to the service provider format to provide resources. The HttpResponse uses its GetEntity to make an HttpEntity. Finally, an instance of InputStream is created by using Httpentity’s getcontent() method to receive resources in the XML or JSON format.

### 5.6.3 SAX PARSER

The following APIs [41] are used to make the SAX parser component:

The **SAXParserFactory** defines a factory API to configure and enables SAX based parser to parse the XML [41].

The **SAXParser** defines API’s to wrap XMLReader which helps to parse XML from a variety of input source by making an instance. To do that, It uses a under lying parser called XML handler [41].

The **MyXMLHandler** is a java class which initiates parsing methods to parse data from a defined starting node to an ending node by calling an instance of the SitesList object.

The **SitesList** is a java class whose properties are made according to a XML document to map the XML tags [41].

**Working details**

In this component, an instance of the SAXParserFactory is created first to make an API configuration. After that, an instance of SAXparser and XMLhandler are created, where the XMLhandler keeps required node information and the starting point. Here, XMLhandler takes
help form SitesList to map data according to the XML documents node. Finally, an instance of the SAXparser is created by using XML document source and an instance of the XMLhandler as parameter to initiate XML parsing. After doing all this mechanism; it is needed to use an arraylist to fetch the data from the SiteLists object.

5.6.4 JSON PARSER

The following APIs are used to make JSON parser component:
The JSONObject is a modifiable set of the name/value mapping where the name cannot be null string though the value may be null [16].
The JSONArray contains a sequence of values with the index respective to the JSON string which helps to parse the node value by using index [16].

Working details
At first, a JSON object is created using the getJSONObject() to make a JSON array. After that, values are fetched from the array using list array.

5.6.5 GSON PARSER

In this parsing method classes for all nodes from the JSON stream are created first. Then the following coding [22] structure is used:
The Gson creates a Gson object.
The Reader reads data from the input stream as character-wise manner of the source.
The fromJSON() maps the reader with root class of the JSON stream.

Working procedure
Here an instance of reader is created from the input stream. Then GSON objects map values with the respective element classes by using fromJSON() method .Finally, data is retrieved from Gson objects.
6. EXPERIMENT AND ANALYSIS

6.1 MOTIVATIONS AND PERFORMANCE INDICATOR

The core components of the mobile Mashup application were described in the previous section (5.6). Attempts at verifying the performance of the above components experimentally were tested in sec 6.4. In these experiments performance of the components are tested considering the following performance indicators to provide a good reference framework [1]:

1. **Total character length** represents the character length of the XML and the JSON data. It is theoretically assumed that XML is more verbose than JSON to represent the same data [14]. For this reason, comparison of the character length of those files was attempted.

2. **Execution time** represents the spent time to conduct the codes of an application.

3. **Bandwidth receives and sent** represents the total amount of bandwidth received and sent through the network interfaces.

4. **Heap information** represents the total amount of dynamic memory allocation by the application.

5. **Total CPU uses by the program thread** provides the thread uses time by the application.

6.2 TEST BED

The experiment is done with two environments using Android emulator. Specifications of environments are as follows:

**Test bed1:**

- **CPU configuration:** ACER aspire AS3810T-6775, Processor: Intel core2 Duo,
- **RAM:** 4 GB, Operating system: win7 32 bit.
- **Network:** Comhem, Speed 10 Mbit/s
- **Eclipse Integration environment:** Android emulator with ADT 16.0.1
- **Development environment:** Android SDK 2.2
Test bed 2:

**CPU configuration:** Hp desktop pc with VPRO technology,
Processor: Intel core2 Quad CPU, RAM: 4 GB, Operating system: win7 32 bit

**Network:** KTH open

**Eclipse Integration environment:** Android emulator with ADT 16.0.1

**Development environment:** Android SDK 2.2

In order to check components (5.6) performance with success, all applications and internet and network useful software remain closed in the PC, and time interval of the experiments between two test beds was 20 days. Next, described computers were scanned with antivirus and anti spyware software before the experiments to get good quality of data. Furthermore, following Android APIs were used to collect information about the performance of the indicators:

**android.os.SystemClock.uptimeMillis()** provides execution time of an application in milliseconds. It was used at the starting point as well as the ending of the application steps. The execution time was calculated by deducting the starting time from the ending time. Example: To get the execution time of the REST client response, the starting time of the response was deducted from ending time of the request initiation.

**android.net.TrafficStats.getTotalRxBytes()** provides total number of bytes received through all network interfaces in the mobile devices. The total byte receive was calculated by deducting the starting point bytes received from the ending point bytes receive.

**android.net.TrafficStats.getTotalTxBytes()** provides the total number of bytes sent through all the network interfaces. The total byte sent was calculated by deducting the starting point bytes sent from the ending point bytes sent.

**Debug.getNativeHeapAllocatedSize()** returns amount of allocated memory in the native heap. Thus, it helps to investigate the memory allocation information of the application.

**Debug.getNativeHeapSize()** returns the total size of the native heap used to run an application. It helps to indicate how much heap size is used by the application.

**Debug.getNativeHeapFreeSize()** returns the amount of the free memory in the native heap by the application.
Debug.threadCpuTimeNanos() provides an indication of the thread CPU usage. The value returned indicates the amount of time that the current thread has spent for executing the code or waiting for the specific types of I/O. The time is expressed in nanoseconds.

String.length() is used to investigate the character length that is contained in the XML or the JSON file. This input stream was thus converted to string and used to specify the character length.

6.3 Method of the Experiment

Three sets of experiments were designed to consider the behavior of the service provider response. The first set of the experiments is set up to analyze the performance of the factors described in sec 6.1, data size of the service provider response is limited and the frequency of data update is six hours. The second set of the experiment is used to analyze the impact of the indicator performance when the data size of the service provider response is greater than that of the 1st experiment. The frequency of data updates was every minute. The third experimental setup was identical to the second one but here the frequency of the data never change and contains very large amounts of data. In test bed 1, each of the experiments runs three times and the arithmetic mean of the outcomes is the final score. In test bed 2, each of the experiments runs five times and the arithmetic mean of the outcomes is the final score.

6.4 Experiment Observation

The observations of the experiment in respect of performance indicators (6.1) are described below:

6.4.1 Character Length Difference

The experimental result of the character length represents the total character length of the data format received from the service provider. The response contains same information of both in the JSON and the XML format. The experimental outcome is as below:
Figure 6 and fig 7 shows character length differences of various service provider responses in JSON or XML format for two configurations. The X-axis represents case studies such as the

---

Page 54 of 85
traffic information, weather information and bounding box though each of them in the order mentioned presents SAX, JSON and GSON parser experiment. The Y-axis represents the character length in integer. As shown in fig 6 and fig 7, the traffic response of the XML file contains less character length in comparison with JSON which is unexpected though the XML character length is higher than the JSON in the weather and the bounding box experiments. Therefore, the raw file of the traffic response is analyzed to investigate causes behind the fact. Now, the observation shows that the JSON file uses a tag called tmcs (see Fig 8 and Fig 9) which is missing in the XML file of the traffic response. In test bed 1 it takes 87 characters and for test bed 2 it takes 790 characters. After deducting this tag character from the total length of the JSON file, it is observed that the JSON file contains less character length than the XML. In the end, the result shows that JSON contains less data length in comparison with the XML in service provider responses, as expected.

FIG 8: The XML responses from the traffic service provider
6.4.2 *DIFFERENCE OF HEAP UTILIZATION IN PERCENTAGE*

The experimental result of the heap utilization represents the percentage of the total heap utilization by the program. In addition, the response contains same information both in JSON and XML format.

Figure 10 and figure 11 shows difference of heap utilization in percentage for receiving and processing of various service provider responses in JSON or XML format for two configurations. The X- axis represents data length of the traffic information, weather information and bounding box though each of them in the order mentioned presents SAX, JSON and GSON parser to parse XML and JSON. The Y-axis represents the percentage of the total heap utilization by the application.
As shown in Fig 10, the JSON parser experiment uses fewer heaps than the SAX and the GSON parser experiment among the three different case studies. It is mentionable that, though here considered parser to represent the figure, it also contains service provider response receiving process. Fig 11 shows SAX uses fewer heaps than JSON and GSON parsing experiments though
other experiments are as fig11. It happened, because in traffic JSON contains 790 extra characters. Therefore, the summary of this experiment can be narrow down as the percentage of the heap uses in the JSON parser experiment < the SAX parser experiment < the GSON experiment.

6.4.3 Difference of Total Heap Utilization

The experimental result of the total heap utilization represents the total heap utilized by the program. The heap size is dynamically allocated by the program that is why; the total heap utilization of an application is investigated. In addition, the response contains same information both in the JSON and the XML format. Figure 12 shows difference of total heap utilization in bytes for receiving and processing of various service provider responses. The X-axis represents the data length of the traffic information, weather information and bounding box though each of them in the order mentioned represents the SAX, JSON and GSON parser to parse the XML and the JSON. The Y-axis represents the total heap utilization in bytes by the application.

Fig 12: Difference of total heap utilization for receiving and processing of various service provider responses in test bed 1
As shown in fig12 and fig13, the JSON parser experiment uses less of total heap than the SAX and the GSON parser experiment in traffic information where the XML data length is smaller than the JSON although the total heap utilization of the JSON is same or greater than the SAX. On the other hand, GSON experiment utilizes highest of the total heap among them. It is mentionable that, though here considered parser to represent the figure, it also contains service provider response receiving process. Therefore, it is observed that Total heap utilization in GSON experiment is high in all cases considering large and small data size. On the contrary, SAX and JSON experiments represents low heap allocation in all cases and difference between SAX experiment and JSON experiment is very low.

6.4.4 DIFFERENCE OF THE Parser Execution TIME

The experimental result of the execution time of the parser represents the total time duration which keeps busy the processor to execute the parser by the program. In addition, the response contains same information both in the JSON and the XML format. Figure 14 and figure 15 below shows a comparison of time utilization by the parser in the application in respect to the character length of the JSON and the XML data where the X-axis represents the data length of the traffic
information, weather information and bounding box though each of them in the order mentioned represents the SAX, JSON and GSON parser to parse the XML and the JSON. The Y-axis is representing the total execution time of the parser by the application in millisecond.

Fig 14: Difference of the parser execution time to process response data in test bed 1

![Graph](image1)

Fig 15: Difference of the parser execution time to process response data in test bed 2

As shown in fig 14, the SAX parser uses less time than others in traffic information where the data length of the traffic information response is few but eventually the execution time of the SAX parser goes higher for the increment of the XML data size (937). This claim shows the same path in fig 15, as it contains almost double data length (2354) than figure 14. In addition, It
is observed that the JSON file uses extra tag which contains 82 characters (6.4.1.1). Therefore, this experiment result could be concluded that the execution time of the XML parser uses more execution time than the JSON parser for fewer data length but for large amount of data GSON parser takes less time.

**6.4.5 Difference of the REST Execution Time (Response Time)**

The experimental result of the resource response time of the REST represents the total time utilized to request and receive response for the document such as the XML, JSON by the program. In addition, the response contains same information both in the JSON and the XML format. The Fig 16 and fig17 below shows a comparison of the REST execution time in millisecond in respect to the character length of the JSON and the XML data where the X-axis representing the data length of the traffic information, weather information and bounding box though each of them in the order mentioned represents the SAX, JSON and GSON parser to parse the XML and the JSON. The Y-axis is representing the total execution time of the REST in the application.

![Execution time of REST vs Character length](image)

**Fig 16: Difference of the REST Execution time in test bed 1**
Fig 17: Difference of the REST Execution time in test bed 2

As shown in fig16, the XML in the traffic information and the weather information uses less execution time than the JSON though it utilizes more execution time than the JSON in the bounding box information. In fig 17, it shows the JSON experiment contains more time than others in traffic information but other two experiments remain almost same. Cause behind that is network performance as it said by Microsoft that “Network Load Balancing scales performance by increasing throughput and minimizing response time to clients” [39].

6.4.6 Difference of the Total Bandwidth receive

The experimental result of the bandwidth received amount represents the total bandwidth received for receiving the response of the service provider. In addition, the response contains same data both in JSON and XML format. Fig 18 and fig19 below show a comparison of the bandwidth received in bytes in respect to the character length of the JSON and the XML data where X- axis representing the data length of the traffic information, weather information and bounding box though each of them in the order mentioned represents the SAX, JSON and GSON parser to parse the XML and the JSON. The Y-axis is representing the total bandwidth received for receiving the response of the service provider.
As shown in fig 18 and fig 19, the XML uses larger bandwidth received in all case studies but the huge reflection bandwidth received is found in bounding box information where data size is large.
6.4.7 **Difference of Total Bandwidth Sent**

The experimental result of the bandwidth sent amount represents the total bandwidth sent for the request to the service provider to receive data. In addition, response contains same information both in the JSON and the XML format. The fig20 and fig 21 below show a comparison of the bandwidth sent in bytes in respect to the character length of the JSON and the XML data where the X- axis represents the data length of the traffic information, weather information and bounding box though each of them in the order mentioned represents the SAX, JSON and GSON parser to parse the XML and the JSON. The Y-axis is representing the total bandwidth sent for the request to the service provider.

![Total bandwidth sent vs character length](image)

**Fig 20: Difference of Total Bandwidth sent in test bed1**
As shown in fig 20, the XML uses less bandwidth sent in traffic information than JSON and GSON parsing experiment where as it uses larger bandwidth sent in the weather information and bounding box information in addition huge reflection bandwidth sent is found in bounding box information where data size is large. In fig 21, it is observed that JSON parser experiment takes larger bandwidth sent than other in all cases. Cause, packet send depends on network architecture and devices Ethernet interface [40].

6.4.8 DIFFERENCE OF CPU THREAD USES

The experimental result of the CPU thread uses represents the total thread of CPU uses in millisecond to perform the application. In addition, the response contains same information both in the JSON and the XML format. The Fig 22 and fig 23 below show comparison of the total CPU thread uses in respect to the character length of the JSON and the XML data where the X-axis represents the data length of the traffic information, weather information and bounding box though each of them in the order mentioned represents the SAX, JSON and GSON parser to parse the XML and the JSON. The Y-axis is representing the total time of the CPU thread uses by the application.
As shown in fig 23 and fig 24, the XML in the traffic information and the weather information uses less CPU than the JSON though it utilizes more CPU than the JSON in the bounding box information. Therefore, it is observed from these experiments that CPU uses is the lowest in JSON parsing experiment for small amount of data (937-1246) and CPU uses is the lowest in for GSON parsing experiment for large amount of data (86342-188166).
### 6.5 Summary

<table>
<thead>
<tr>
<th>Exp No</th>
<th>Experiment</th>
<th>Result</th>
<th>Consideration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Character length difference (6.4.1)</td>
<td>JSON contains less data length in comparison with the XML</td>
<td>JSON data format</td>
</tr>
<tr>
<td>2</td>
<td>Difference of Heap utilization in percentage (6.4.2)</td>
<td>Percentage of the heap uses in the JSON parser experiment &lt; the SAX parser experiment &lt; the GSON experiment. In all cases.</td>
<td>JSON data processing</td>
</tr>
<tr>
<td>3</td>
<td>Difference of Total Heap utilization (6.4.3)</td>
<td>Total heap utilization in GSON experiment is high in all cases considering large and small data size. On the contrary, SAX and JSON experiments represent low heap allocation in all cases and difference between SAX experiment and JSON experiment is very low. Though XML processing uses more heap than JSON.</td>
<td>JSON data processing</td>
</tr>
</tbody>
</table>
| 4      | Difference of parser Execution time (6.4.4) | The execution time of the XML parser uses more execution time than the JSON parser for fewer data length but for large amount of data GSON parser takes less time. | • JSON parser for small data.  
• GSON parser for large amount of data. |
| 5      | Difference of REST Execution time (response time) (6.4.5) | Result of two experiments differs in traffic information experiment and other two case studies are | Depends on network performance. Therefore, this factor is so not considered. |
almost same. Network performance minimizes response time [39].

<table>
<thead>
<tr>
<th>6</th>
<th>Difference of the Total Bandwidth receive (6.4.6)</th>
<th>XML uses larger bandwidth received in all case studies but the huge reflection bandwidth received is found in bounding box information where data size is large.</th>
<th>JSON file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Difference of Total Bandwidth sent (6.4.7)</td>
<td>Bandwidth sent differs in JSON experiment for all case studies. Cause packet send depends on network architecture and devices Ethernet interface [40].</td>
<td>Depends on network architecture and devices. Therefore, this factor is so not considered this factor.</td>
</tr>
<tr>
<td>8</td>
<td>Difference of CPU thread uses (6.4.8)</td>
<td>CPU uses is the lowest in JSON parsing experiment for small amount of data (937-1246) and CPU uses is the lowest in for GSON parser for large amount of data (86342-188166). Here considering the character length of extra tag tmcs in JSON response.</td>
<td>• JSON parser for small data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• GSON parser for large amount of data.</td>
</tr>
</tbody>
</table>

Table 8: summary of the experiment and final consideration

As shown in table 8, I choose JSON data format and, for small amount of data JSON processing with JSON parser is suitable. On the contrary, JSON processing with GSON parser does well for large amount of data. Subsequently, it is observed from table 8 that data processing performance for web service communication depends on data length.
7. RESULT

7.1 RESULT OF THE EXPERIMENTS

It is observed from the experiment that the behavior of the performance indicators varies for two situations such as large amount of data length and relatively small amount of data length. The summary of the experiment therefore described as below:

1. Performance of the Mashup components:

It concludes from sec 6.5 that performance of the core component of mobile Mashup depends on data length of the service providers’ response.

2. Data format with short amount of data length

The observation from sec 6.5 shows that the SAX parser and the JSON parser uses almost same execution time for small amount of data but the XML uses more resources than the JSON in the bandwidth uses, the percentage of the total heap allocation, total heap allocation, bandwidth received and CPU thread utilization for communicating with the service provider and parsing the response. As a result the JSON data format and the JSON parser is preferable for the processing and representing the short data response. It is worth to mention that both XML and JSON data formats are considered as the response of the service provider.

3. Data format with large amount of data length

The observation from sec 6.5 shows that the GSON parser uses the least execution time and the resources for large amount of data than the SAX parser and the JSON parser though it utilizes more resources of a mobile phone for short amount of data than the JSON parser and the SAX parser in respect of the bandwidth uses, the percentage of the total heap allocation, total heap allocation, bandwidth receive and CPU thread utilization for communicating with the service provider and parsing the response. As a result the JSON data format and the GSON parser is preferable for large amount of data response processing and representation in the mobile phone. It is worth to mention that, both XML and JSON data formats are considered as the response of the service provider.
7.2 PROVE OF THE ASSUMPTIONS

Proofs of the two assumptions (5.3) are described as below:

1. Use of radio transmission reduction

The experiment (6.4.6) shows that, JSON data format is applicable for communicating with the service provider which uses less data than XML. Therefore, the radio devices pass less data than the XML. Furthermore, it is suggested to use the REST client than the SOAP because of uses of less communication messages with the service provider (3.5). It is also found that, JSON is faster than XML in data transmission (4.3). Therefore, it can be concluded that using REST and JSON will reduce the radio transmission of the mobile device which will reduce power utilization of the mobile phone as the radio device consumes power from the mobile battery.

2. Reduction of the execution time and resource uses in the Mobile devices

The experiment result (6.5) shows that the JSON data format with the JSON parser performs better than SAX and GSON for short length of data response in respect of execution time, heap utilization and CPU usages. In spite of this, the GSON parser with the JSON data format performs better than the JSON parser and the SAX for large data processing in respect of execution time, heap utilization and CPU usages. These two results were carried out from experimental results of sec 6.4.4 and sec 6.4.8. Therefore, it can be concluded that, the assumption of the execution time and the resource uses reduction in the mobile phone is proved.

7.3 PROPOSED REFERENCE FRAMEWORK

From all the discussion up to now, the study is reached in a position to propose a reference framework on the native application for mobile Mashup. The summary of the proposed reference framework for Mashup is in table 9 based on table 8. The study considered the Consumer Mashup type with client based architecture, because of the technological development of the mobile phone and mobile communication. The proposed reference framework is also applicable for the Business Mashup type but needs to be provided with the data security. Eventually, it is needed to know the protocol and service provider terms of services (ToS) to use their service. Therefore, the service provider specification and protocol is described in the reference framework. Here the REST protocol is emphasized as a communication protocol of the Mashup application. Next, the
client architecture is chosen to provide the Mashup implementation location such as the mobile phone. Finally, the JSON data format with JSON parser and GSON parser is proposed based on the data length of service provider response to perform better.

<table>
<thead>
<tr>
<th>Section</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type (2.1.1)</strong></td>
<td>Consumer Mashup Uses different types of data with different sources to combine and represent visually</td>
</tr>
<tr>
<td><strong>Service provider (3.1)</strong></td>
<td>Provides the service according to the client request by obeying the Terms of service (ToS).</td>
</tr>
<tr>
<td><strong>Architecture (5.2)</strong></td>
<td>Client Based Executes and presents all the received data in the mobile phones.</td>
</tr>
<tr>
<td><strong>Protocol (3.3.4)</strong></td>
<td>REST Widely available and provides best performance on the client side.</td>
</tr>
<tr>
<td><strong>DATA format (6.5)</strong></td>
<td><strong>XML</strong> The W3C recommended and widely used in web service communication</td>
</tr>
<tr>
<td></td>
<td><strong>JSON</strong> Performs better than the XML in communication and parsing.</td>
</tr>
<tr>
<td>Guideline of uses</td>
<td>Use XML if the service provider only responses in XML. It is also applicable for RSS feed.</td>
</tr>
<tr>
<td></td>
<td>Use the JSON if the service provider responses both in XML and JSON format. To do that, first consider the data size of response message.</td>
</tr>
<tr>
<td><strong>Parser (6.5)</strong></td>
<td><strong>SAX</strong> Performs better with limited use of resources among all the XML parser for the restricted devices such as the mobile Phone.</td>
</tr>
<tr>
<td></td>
<td><strong>JSON parser</strong> It is a popular parser which is used to parse only the JSON data by using the JSON array.</td>
</tr>
<tr>
<td></td>
<td><strong>GSON parser</strong> Used to parse only the JSON data using the object serialization.</td>
</tr>
<tr>
<td>Guideline of uses</td>
<td>Use the SAX parser if the service provider responses only in XML.</td>
</tr>
</tbody>
</table>
Table 9: Proposed reference framework for native Mobile application

<table>
<thead>
<tr>
<th>uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use the JSON parser if the service provider response contains a small amount data both in XML and JSON format</td>
</tr>
<tr>
<td>Use the GSON parser if the service provider response contains a large amount of the data length.</td>
</tr>
</tbody>
</table>

7.4 Case Study Analysis

Two case studies (1.5) are implemented below by obeying the reference framework table 6. Here the weather forecast Mashup is implemented by using the web API and proposed reference framework (7.3) to prove that the native application is also able to use the web API. In contrast the traffic information Mashup is implemented based on only the proposed reference framework.

7.4.1 Case Study 1

The Location based weather forecast Mashup

Description

This Mashup application provides weather forecast of a user location along with details address and graphical map. This application takes GPS location of a user and passes that to the service providers with the request message.

Type

This Mashup application is an example of the Consumer Mashup type.

Service providers and their specifications

**GPS provider:** The users need to subscribe with the GPS service provider available in their places to get the GPS information by making an agreement.

**Google Maps and Google Geocode:** These service providers provide the service in free but user need to register with the service provider. According to the specification of the service provider they use the REST architecture and the JSON data format.

**The World weather online:** This service provider provides the weather forecast of a location for 5 days. It is free for test application and needs to register with them to get
access key. This provider provides data through the API and raw data (XML and JSON). It uses the REST architecture to provide data. The raw data format is considered to investigate the research question.

**Architecture**
This Mashup application follows the client architecture mentioned in section 5.2. Here the Mashup application in the Mobile device parses all collected information from the service providers and represents it to the user.

**Data format**
Here JSON format is considered as the response of service provider.

**Protocol**
This weather forecast Mashup used the REST API to get location details and the REST client to communicate with weather service provider.

**Parser**
Here the JSON parser is used to parse the response of the weather information service provider.

**Reference framework**
Here, the proposed reference framework (7.3) was used for receiving weather forecast information.

### 7.4.2 Case study 2

**The traffic information provider application**

**Description**
This Mashup application provides the traffic information of a specific location of the user along with the specification of surrounding area of provided location name. This application takes the location name of a user and passes that to the Bounding box service provider with request message. In a response, it receives the surrounding area specification such as Latitude and Longitude. Next the application calls traffic information service provider with the Surrounding area information to get the traffic information. Finally the display shows the traffic information along with the Surrounding area of the provided location.

**Type**
This Mashup application follows the Consumer Mashup type.

**Service providers and their specifications:**
**Mapquest**: This service provider provides the updated *traffic information* of the roads in the USA by updating information in every minute. It also provides the *Surrounding area* of a specific location. It is free for test application and needs to register with them to get access key. This provider provides data through the API and the raw data (XML and JSON). It uses the REST architecture to provide data. The raw data format is considered to investigate the research question.

**Architecture**
This Mashup application follows the client architecture mentioned in section 5.2. Here the Mashup application in the mobile device parses all the collected information from the service providers and represents it to the user.

**Data format**
Here JSON format is considered as the response of service providers

**Protocol**
The Mashup application considered here used the REST client to communicate with the Bounding box information service provider and the traffic information service provider.

**Parser**
This traffic information Mashup used the JSON parser to parse the response of the Traffic information service provider and the GSON parser to parse the response of the Bounding box service provider.

**Reference framework**
Here the proposed reference framework (7.3) was used for receiving weather forecast information.

### 7.5 The UML Representation of the Case Study
The UML (Unified Modeling Language) is a standard notation to represent the object oriented methods. In this technique the graphical notation is used to define the methods or activity or message communication among the classes. Therefore, the above case studies are represented using sequence diagram to show how processes operate with one another.
### 7.5.1 Sequence Diagram and Snapshot of Case Study 1

**Fig 24: sequence diagram of case study 1**

As shown in fig 24, the Actor is a user of the application who initiates the Mashup application through the activity to call the GPS module for the latitude and the longitude data. The Map view component uses this latitude and longitude data to represent the user specific location on the map by using the position overlay class. Then the Geocoder Api uses these latitude and longitude data to provide the user address. After that, GPS data again goes to the weather forecast service provider by calling the RESTful class to retrieve the weather forecast data in XML or JSON format. At last, the result goes to the parsing function to parse and display the information.

**Snapshot**

Fig 25 shows the output of the weather forecast Mashup. It shows details of the user address from the geocoder service provider, weather information from the weather service provider and the Google map of the user with specification (I am here) for the Latitude of -74.0063889 and Longitude of 40.7141667.
Fig 25: User location and weather forecast for the location of Latitude of -74.00 63889 and Longitude of 40.7141667.
7.5.2 Sequence diagram Snap shot of case study 2

As shown in Fig 26, the Actor is a user of the application who initiates the Mashup application first through the activity to call the REST service to communicate with the bounding box service with the location name to get the surrounding area. Then the Bounding box service provider provides specification of surrounding area in a JSON file. Next the JSON file with surrounding area goes to the GSON parser to parse and display the surrounding information. After that the Mashup application passes this data to the traffic information provider through the Restful service to get the traffic information. The response of the traffic information comes in a JSON file and the response goes to the JSON parser to parse and display the traffic information. Here the Mashup application uses the URL, KEY and surrounding area information as parameters to the traffic service provider through the Restful service component.

Snapshot
Fig 27 shows the output of the Traffic information Mashup. It also shows the surrounding area form the bounding box service and the traffic information from the traffic information service provider for the location of Denver, Colorado, USA.

**Fig 27: The surrounding Area and the traffic information for the location of Denver, Colorado, USA**
8. CONCLUSION

In this study the mobile Mashup for the native application is examined to provide necessary elements to build a mobile Mashup. The main objective of this study was to propose an effective native application development framework for a mobile Mashup. The main research question (1.2) was investigated to provide core elements of the mobile Mashup with the best performance. Therefore, all the experiments and descriptions described above answer the main research questions and the sub questions. Finally, the answers could be described as below:

**Sub question 1: Challenges to design an application for a large variety of Mobile environment**

The Android operating system is used here to implement two case studies of the mobile Mashup based on the proposed reference framework (6.5). At present the android is a widely used mobile operating system (4.6). That is why, it can be concluded that the proposed reference framework can be implemented in a large variety of the mobile environment.

**Sub question 2: Adjustable with small display of the Mobile devices**

The implementation of two case studies in 7.5 shows the weather information and the traffic information with map to meet the requirements of the user. However, the outcome of the native application is adjustable with mobile device display.

**Sub question 3: Applications with less power and CPU utilization**

This is one of the most critical questions that drove this study to analyze all the components of the mobile Mashup experimentally. The outcome of the experiment shows that the REST protocol with the JSON parser and the GSON parser for the JSON data use fewer resources than the XML depending on the data length of the service provider response (7.1). That is why; a reference framework is proposed (7.3) to build the Mashup application in the mobile phone. It is natural that the applications using this reference framework for the Mashup application will utilize less power and CPU of the mobile phone.
Sub question 4: Application with limited bandwidth uses

The answer to the 2nd most critical question that the REST protocol consumes less data in comparison with the SOAP to make communication with the service provider and the service requester. In addition, it was found that the XML data length is higher than the JSON data which drives to use more bandwidth than the JSON data format (6.4.6, 6.4.7). That is why the Reference framework (7.3) proposed the REST as a communication protocol and the JSON as the communication data format. Eventually it can be concluded that the Mashup application built on the reference framework consumes less bandwidth.

Finally, the experiments and the implementations of the case studies (7.4) based on the proposed reference framework prove the main research question (1.2). That is why; the proposed reference framework is a suitable native application development environment for the mobile Mashup. Moreover, the proposed reference framework could be classified as a Middle ware integration framework (4.11) as it combines received data before presenting to the users.

Above description also proves goals (1.3) of this study which are as below:

1. Two considered Mashup case studies (7.5) implemented using proposed reference framework (7.3) in the mobile phone. These implementations prove that client based Mashup architecture is implementable in a mobile devices. Furthermore, Section 2.1.2 is introduced in this study to provide a brief description about Mashup architecture.

2. Next, the proposed reference framework (7.3) is defined as a consumer type Mashup. This conclusion came from brief description of different types of Mashup in section 2.1.1.

3. Subsequently, section 5.6 describes implementation of core components for mobile Mashup. This consequence came from the details analysis of core components of Mashup in section 2.1.3 and section 2.1.5.

4. Finally, a reference framework (7.3) on native mobile application is proposed after verifying the performance of core components of mobile Mashup in chapter 6. The summary of these experiments are described in section 6.5 with analysis and results.

After proving above four goals two mobile Mashup case studies are implemented and described with sequence diagram in sec 7.4.
The main **contribution** of this study was a reference framework for mobile Mashup. The identified results were:

1. A fully functional and fewer resource consumed reference framework for mobile Mashup using native application has been proposed (7.3).
2. Fully functional client type Mashup architecture has been implemented and tested for mobile devices (7.5).
3. Verified performance of XML and JSON in REST protocol in respect of response time (6.4.5), bandwidth receives (6.4.6) and bandwidth sent (6.4.7).
4. Verified performance of SAX, JSON and GSON parser in respect of execution time (6.4.4). It is observed that performance of these parsers depends on data length of the web service responded data format. That is why; a guide line is introduced to use parser in the reference framework.
5. Verified the performance of web service and data processing components for Mashup applications in respect of memory utilization (6.4.2), memory allocation in percentage (6.4.3) and CPU utilization (6.4.8).

To conclude the study primarily shows a potential direct implementation in the mobile Mashup. Secondly, it could also help the mobile application developer to work with the web services. Thirdly, it can help the developers to choose an appropriate parser for the data format. However, it is worth noticing that the limitation of this study is that the provided reference frame work is only applicable for the consumer type Mashup though it is also applicable in the business type Mashup but needs to be provided with the data security. In the business type Mashup, data are aggregated from many sources and therefore analysis of the load test is important. That is why the future work could be the analysis of the load test and the data security in the field of the mobile Mashup in the mobile devices. In addition, development of a mobile Mashup tools can be considered as a future work. Since this study provides the well executed components for mobile Mashup, those can work as backend processing unit for the mobile Mashup.

-------- The end--------
REFERENCES


[29]. Viedma. Cristobal,” Mobile web Mashups”, KTH. Sweden 2010


