RESTful PUBLISH/SUBSCRIBE FRAMEWORK FOR MOBILE DEVICES

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of Master of Science

In the Department of Computer Science

University of Saskatchewan

Saskatoon

By

RAHNUMA KAZI

PERMISSION TO USE

ABSTRACT

The growing popularity of mobile platforms is changing the Internet user's computing experience. Current studies suggest that the traditional ubiquitous computing landscape is shifting towards more enhanced and broader mobile computing platform consists of large number of heterogeneous devices. Smartphones and tablets begin to replace the desktop as the primary means of interacting with IT resources. While mobile devices facilitate in consuming web resources in the form of web services, the growing demand for consuming services on mobile device is introducing a complex ecosystem in the mobile environment. This research addresses the communication challenges involved in mobile distributed networks and proposes an event-driven communication approach for information dissemination. This research investigates different communication techniques such as synchronous and asynchronous polling and long-polling, server-side push as mechanisms between client-server interactions and the latest web technologies namely HTML5 standard WebSocket as communication protocol within a publish/subscribe paradigm. Finally, this research introduces and evaluates a framework that is hybrid of REST and event-based publish/subscribe for operating in the mobile environment.

ACKNOWLEDGEMENT

I would like express my sincere thanks and gratitude to my academic supervisor Professor Ralph Deters for his guidance, inspiration and funding throughout my study for last two years. I would also like to thank my thesis committee members Professor Julita Vessileva, Dr. Chanchal Roy and Dr. Ramakrishna Gokaraju for their valuable feedbacks on my M.Sc. thesis.

I would like to thank our Graduate Chair Dr. Mark Eramian for his continuous supervision on grad students. I would also like to thank our Graduate Assistant, Ms. Gwen Lanchaster for her endless support and care throughout my studies and for bring us delicious and irresistible handmade cakes.

I would like to thank our University ICT help desk for providing us all kinds of technical supports, answering various questions and most importantly notifying us in advance for every change in the university network that I should be aware of. I would also like to thank Greg Oster, the IT staff member of Computer Science Department for his assistant in acquiring a public DNS name for my lab machine.

I would like to convey my heartfelt thanks to all my colleagues from Multi-Agent, Distributed, Mobile and Ubiquitous Computing (MADMUC) lab for their friendship and valuable feedbacks to my research.

Finally, I would like to thank my dearest family members, my father Qudrat-E-Khuda, my mother Sharifa Begum, my sister and my mentor Dr. Fahmida Farzana, my bother Shoyweeb Faisal and my brother-in-law Dr. Moniruzzaman for their unconditional love, support and inspiration throughout my study. And last but not least, I would like to convey my humble gratitude to the Lord, the Almighty, for His eternal love and blessings that never ends.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CRUD	Create Read Update Delete
CSS	Client Stateless Server
CCSS	Client Cache Stateless Server
EC2	Elastic Compute Cloud
EB	Event Broker
EDA	Event Driven Architecture
GUI	Graphical User Interface
MVC	Model View Controller
MVP	Model View Presenter
MOM	Message Oriented Model
P/S	Publish/Subscribe
PM	Policy Model
REST	Representational State Transfer)
RPC	Remote Procedure Call
ROM	Resource Oriented Model
SOM	Service Oriented Model
SOA	Service Oriented Architecture
WS	Web Service
WSD	Web Service Description

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

In recent years, the growth of mobile devices such as smartphone and tablets has led to an extensive use of mobile applications in almost every sector of our life. The Gartner research [Gartner Report, 2011] forecast 2011 states that the download of mobile apps worldwide had increased by 117 percent from 2010 to 2011 and forecasts an astounding 185 billion downloads from mobile app store by 2014 since the first launch in 2008. The capabilities of these devices in doing more than just making calls as well as sending and receiving text messages has increased the demand for mobile applications in the enterprise as it becomes possible for enterprises to extend their services to the fingertips of numerous consumers. Education, healthcare and business enterprises are some of the sectors where the use of mobile applications is found to flourish in bringing a revolutionary change in the way that data is recorded, accessed, processed and evaluated for use. According to [Ranck, 2010] on “The Rise of Mobile Health Apps”, the current statistics of mobile health (mHealth) apps is over 6,000 in the App store which shows a growing demand for mobile applications in the health domain.

Generally, these mobile applications consume data as Web services from a remote server-based architecture, which is the backbone of most information systems. Today’s information society is built upon collaborative platforms which gathers and shares information across distributed networks. The backbone of these information systems consists of multiple disparate system applications. The growing demand of consumers in accessing services is causing these systems to expand and some of these services can be hosted in the cloud computing environment in order to ensure availability, reliability and scalability in service consumption.

Cloud computing is the era where IT services are outsourced from providers over the internet on pay-according-to-use policy [Lomotey and Deters, 2013]. This ecosystem of bringing disparate platforms together is often referred as a "Distributed System Environment" as illustrated in Figure 1.1.

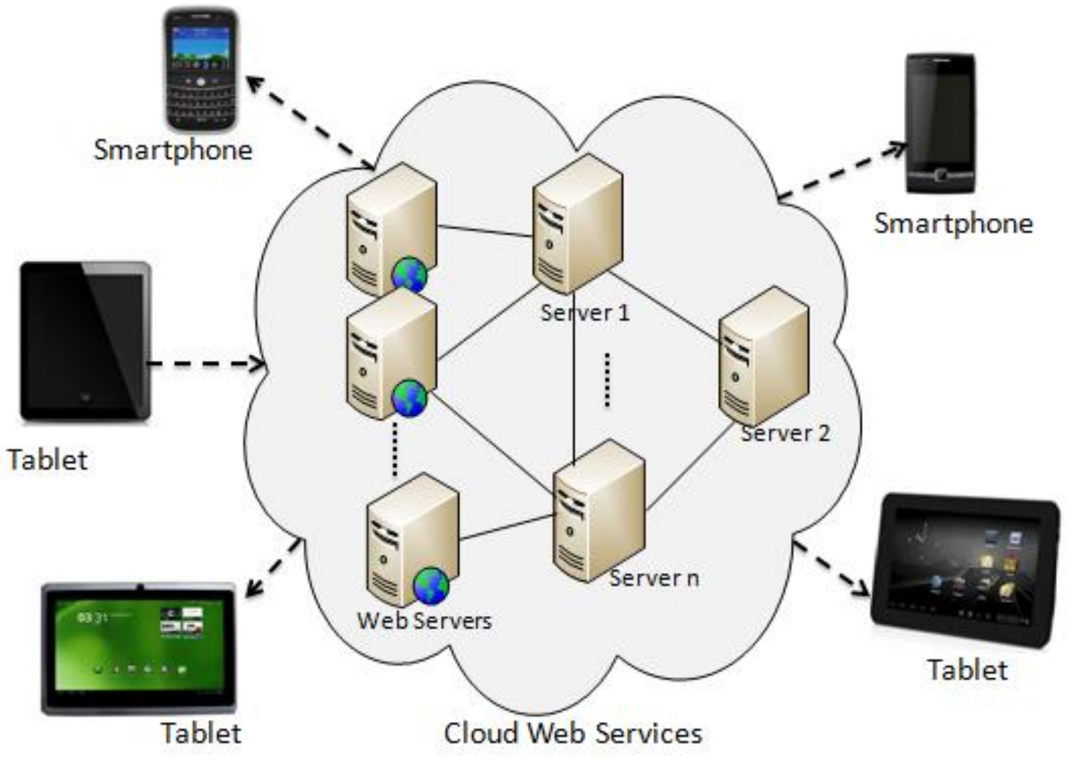


Figure 1.1: Distributed System Environment

With the growing demand of consumer web services and the expansion of systems that forms a gigantic distributed heterogeneous infrastructure, there is an acute need for a framework that can reliably operate in the mobile environment. Since wireless network (e.g. Wi-Fi, 3G/4G) are prone to intermittent connection loss and the system application components can be distributed, mobile applications interacting with the backend servers often face several challenges in providing fast and consistent data delivery. Moreover, the bandwidth fluctuation limitation of wireless connections, which can be attributed to the mobility of users, urges for an efficient dissemination of data.

1.2 Problem Statement

While distributing the system applications provides more flexibility and scalability, it often results into a growing system complexity during services consumption in a mobile environment. One of the major challenges in today's enterprise solution is to ensure integrity among these disparate and distributed system applications which are often connected to legacy systems. In addition to that, mobile devices are becoming an integral part of the growing digital ecosystem and the primary means of accessing IT services. This introduces more challenges to the system when synchronizing the information flow between mobile clients and the distributed system backend. The major challenges while disseminating data over a wireless connection in a mobile environment are as follows,

i. **Unreliable network connection**. Despite the advances in mobile technology, these devices still rely on wireless mediums (e.g. Wi-Fi, Bluetooth etc.) to communicate with other distributed components; these wireless mediums can be unstable especially due to user mobility. As a result, seamless interaction and delivery of information to the mobile devices in a large network becomes challenging [Sutton et al., 2011].

ii. **Higher degree of network latency.** Communication over wireless channels encounters a higher degree of latency that causes delayed information dissemination. As a result, synchronizing resource's state updates on mobile node becomes challenging and mobile users often experience inconsistent view of the application data.

iii. **Limited network bandwidth.** Constraint bandwidth availability is one of the major challenges in wireless communication since Wi-Fi connection is distance sensitive. A WLAN using 802.11b supports 11 Mbps and WLAN using 802.11g supports 54 Mbps. As wireless devices moves farther away from the access point, the performance degrades and the available bandwidth often fluctuates. Moreover, as more wireless devices utilize the connection, mobile users often experience a low performance.

1.3 Research Goals

In addressing the above mentioned challenges in mobile digital ecosystems, this research looks into developing a framework for disseminating data over wireless networks and proposes an architecture that allows system components to independently propagate data (i.e. resource updates) and as they propagate, the eventual consistency technique is employed to synchronize the data (i.e. resource states).

In this regard, my research looks into the Pub/Sub pattern as a mechanism for propagating data close to real-time. Moreover, the emergence Web 2.0 has greatly embraced the RESTful (discussed in detail in the next section) web services [Webber et al., 2010; Fielding, 2000] due to its web compliant API and lightweight solution for resource's state management. Therefore, the proposed framework in this research is a hybrid of REST-based and event-based Pub/Sub that deploys a combination of various client-server interaction modes such as polling, long-polling and server-side pushing. The detail description of RESTful web services and the Pub/Sub design model can be found in chapter 2. The research goals in proposing a novel framework for mobile devices are as follows,

- **Goal 1.** To integrate REST web services within Pub/Sub domain. In that, this research will look into different REST patterns based on Richardson's Maturity Model (RMM) [Fowler, 2010] in disseminating data and understand which of the REST pattern is most suitable for an event-based Pub/Sub system.
- **Goal 2.** To address the above mentioned challenges in wireless network. Hence, the research goal is to reduce network latency, bandwidth usage and also synchronizing resource's state in the face in intermittent connection loss.

Some of the research questions that underlies this study are as follows,

- How fast and efficiently can mobile clients communicate with backend servers?
- How can seamless interaction be facilitated between mobile clients and backend servers in the face of faulty network?

The remainder of the thesis is organized as follows. Chapter 2 reviews some of the key points that this study explored and the existing research works within the identified problem domain. Chapter 3 presents the proposed framework design in addressing the research goals and challenges. Chapter 4 describes the implementation details of the architecture followed by the experiments in chapter 5 designed to verify the framework in accordance with the research goals. Finally chapter 6 concludes the thesis with the contributions of this research.

CHAPTER 2

LITERATURE REVIEW

This section reviews the related concepts and issues in the following order. First in section 2.1, it looks into Pub/Sub design model in disseminating information. Then in section 2.2 it looks into different Web-based communication techniques in integrating Pub/Sub in mobile space. Section 2.3 describes Web services in integrating distributed system components and focuses on REST Web Services in a greater detail. Section 2.4 looks into different software development patterns. Section 2.5 discusses mobile cloud computing in hosting services in the cloud. Finally section 2.6 summarizes the chapter with a discussion of possible solutions in addressing the research problems.

2.1 Pub/Sub System

In the traditional client/server model, a client requesting (pulling) for update information from a server is not efficient as servers encounter tremendous overhead and also not very suitable approach for dynamic information dissemination when dealing with a large distributed network. A communication model that helps in dealing with the information dissemination in a larger scale mobile network is Pub/Sub paradigm [Liu et al., 2010]. In this Pub/Sub architecture, information providers as publishers disseminate information in the form of events and information consumers as subscribers register for events of their own interests. There can be an event broker acting as a middleware which helps in dispatching events to the respective subscribers [Huang, Y., Molina, G., 2001]. Communication in Pub/Sub is inherently asynchronous and transparent in nature as both entities (information provider and subscriber) operate asynchronously through a dispatcher and disseminate state changes to all interested subscribers through one operation. In the basic model of a Pub/Sub system, both providers and subscribers are connected through a set of groups or channels through which subscribers are notified for the events of their interest. Upon receiving event notification, the publisher dispatches the event to the respective subscribers.

2.1.1 Subscription schemes

As subscribers are not interested in all the events that are published by the providers, there are various ways that the subscriber can specify interest for a specific event. These variations have led to different subscription models that are currently seen in Pub/Sub system environments. This section explains two most widely used subscription schemes.

- **Topic-based Pub/Sub scheme.** One of the first generation subscription schemes is the topic-based scheme. In this scheme, subscribers register for notification based on the topic or subject of the events corresponding to a particular group or a set of groups also known as a logical channel [Baldoni and Virgillito, 2005]. Users subscribed to a channel(s) will receive all published events of that channel. The topic-based scheme has been proposed as a solution in many industrial Pub/Sub environments. One of the most mentioned systems is CORBA notification service [Object Management Group, 2002]. Also, among others, TIV/RV, SCRIBE and Bayeux are some of the systems that implement topic-based scheme [Baldoni and Virgillito, 2005]. A drawback encountered in this scheme is its limited expressiveness of the subscribers. A subscriber registers for a subset of events of a topic receives all the published events related to that topic. However, a hierarchical organization of topic-based system has been proposed as a solution to this problem [Eugster et al., 2003].

- **Content-based Pub/Sub scheme.** A more flexible paradigm in the Pub/Sub scheme is content-based subscription. It provides more flexibility to the subscriber by providing more control in subscribing an event based on the actual content of the event. It allows subscriber to impose set of constraints in the form of condition in forming a query on an event notification (also known as filter). Creating a notification using a filter provides subscribers with a more sophisticated way for subscribing events. However, this higher expressive capability in defining subscription on the other hand can be an added challenge in implementing such a scheme since matching publisher's events with subscriber become more complicated and the resource consumption becomes higher [Baldoni and Virgillito, 2005; Eugster et al., 2003]. There are several examples of systems that implement content-based subscription scheme such as Siena [Heimbigne, 2003], Jedi [Cugola, 2001], and Rebeca [Fiege and Muhl, 2000].

2.1.2 Messaging System

A Pub/Sub system is better understood in the domain of a messaging system and also known as Pub/Sub messaging system (Figure 2.1). A messaging system has the capability of managing messages in a way a persistent database is managed by a database system. Messages are coordinated and integrated among the software components as software applications changes over time. Messages are transferred from one machine to another over the unreliable wireless network. The inherent limitations of wireless network makes the messaging system suitable to operate as it repeatedly tries to transmit message until it has been sent. The five steps [Hohpe and Woolf, 2004] in sending messages include – create, send, deliver, receive and process.

The basic concepts in a messaging technology revolve around the key terms of message, channel and routing messages.

Channels – Channels are the virtual pipes that connect senders (publishers) and receivers (subscribers) over the network. Based on how an application needs to communicate, channels are created to facilitate messaging applications in transmitting data.

Messages – Data that are transmitted are wrapped into an atomic packet to form a message. An application must encapsulate the data into a message before in transmits to a channel. Likewise, the message needs to be extracted in the receiver's side in order to process the data.

Routing Messages – Routing is considered as an important concept especially in a large enterprise that requires connecting large number of applications and their channels in transmitting messages. The complexity in routing message depends on the message's final destination as it may needs to go through multiple channels.

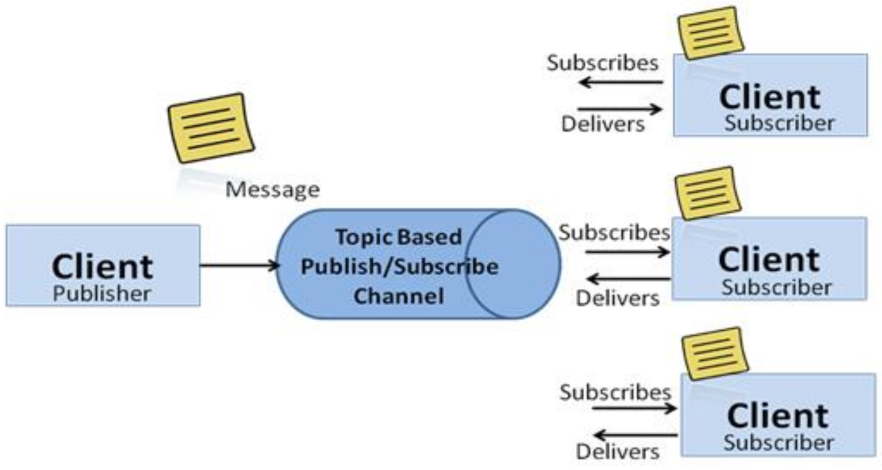


Figure 2.1: Topic-based Pub/Sub Design Pattern

Transmitting data in sending messages back and forth has many added advantages in a distributed application system. Some of the major advantages are –

Asynchronous communication – In asynchronous communication a sender doesn't need to wait for the response to come in order to send the next request. In a messaging system, messages are sent in *send and forget* approach [Hohpe and Woolf, 2004]. Once a message is sent to a message channel, sender does not need to wait for the receiver to receive and process that message, which means sender does not wait for the messaging system to deliver the data. Sender can continue performing the other works once a message is being sent.

Throttling – A problem with messaging in Remote Procedure Calls (RPC) is that the receiver may crash due to the overhead of incoming messages. A messaging system has control on the number of requests to be sent to the receiver to process which saves the receiver from crashing.

However, queuing the request to avoid throttling may cause additional delay for the request senders in receiving response [Hohpe and Woolf, 2004]. This problem is solved by the asynchronous nature request-response of a messaging system.

Reliable communication – Messaging system uses a store and forward style [Hohpe and Woolf, 2004] in providing a reliable delivery of messages. In store and forward style, messaging system first stores the message in the sender's memory and then forwards and stores it again to the receiver's end. While storing the message at both ends can make the system more reliable, forwarding message over wireless connection can be unreliable. However, the repetitive nature in store and forward until the message is received at the receiving end solves the unreliability problem.

2.1.3 Pub/Sub in Mobile Environment

There are several papers that analyze the existing Pub/Sub model mostly on the content-based subscription and suggest more enhanced approaches. These approaches can be adapted into a mobile environment considering mobility issues of Pub/Sub system elements. The main purpose of these researches is to provide a suitable scheme in disseminating information in a mobile network.

[Huang, Y., Molina, G., 2001] proposed a middleware approach for a Pub/Sub implementation and its adaptation into a mobile environment. The authors explains how an event broker as a mediator can facilitate Pub/Sub communication in both centralized and decentralized mobile environment and proposes an algorithm for an optimized wireless network communication. The paper addresses the challenges of mobile networks in terms of network disconnection at any certain point and suggests the replication of users' subscription over multiple event brokers in order to improve the availability and reliability of the system in a mobile environment.

A scalable decentralized peer-based subscription approach implementation of Pub/Sub system has been proposed by [Anceaume et al., 2002]. The study presents a topic-based deterministic information dissemination scheme that provides transparency for publisher and subscriber. A logical orientation scheme in subscription model also ensures a space optimized information dissemination.

Other middleware approaches in Pub/Sub system implementations are seen in the works of [Cugola and Jacobsen, 2002], [Cilia et al., 2003] and [Fiege et al., 2003]. Two key problems that arise in mobile applications in Pub/Sub system that have been addressed in [Cugola and Jacobsen, 2002] are namely scalability, in supporting large number of mobile clients and adapting to application topology as mobile components are subject to change their locations. TOPSS and JEDI are two examples of Pub/Sub systems that address scalability issue by implementing an efficient filtering mechanism at the event broker.

A content-based Pub/Sub middleware approach has been proposed in [Fiege et al., 2003]. The concept of mobility has been segregated into two - physical mobility and logical mobility. Depending on logical mobility, a new approach of 'location dependent subscription' using location-dependent filter has been introduced by author. In addition, the goal of [Caporuscio et al., 2003] is to support mobile client applications in a decentralized Pub/Sub environment where clients are connected to one of the interconnected access points that serve as message routers in a distributed network. The paper implements a ‘mobility support service’ that provides this support to a mobile client by introducing independent mobility service proxies running at the access points of the Pub/Sub system.

2.1.4 Integrating a Pub/Sub system with mobile web browser

Although different implementations of mobile Pub/Sub systems have different prototypical and standard approaches, the common goal in all of these implementations is achieving an efficient data dissemination strategy. The objective of data dissemination is to transfer dynamic information (state) changes as a consequence of publishing new data and updating existing data from publishers to mobile consumers [Mühl, 2004].

In today’s heterogeneous networks that consists of WiFi, 3G or 4G networks, most of the client consumers in Pub/Sub systems are smartphones and tablets, running native apps or mobile Web apps. From the developers perspective it is a controversial issue when it comes to developing apps for mobile devices. Native apps are developed solely for mobile devices which are accessible via specific device platform such as Android, Blackberry and iOS with a full access capability into the core device features. Mobile Web apps on the other hand provide the platform for single code based solution to be deployed on mobile devices with similar and more improved user experience as native apps. Thus the mobile web app design reduces the cost of building and maintenance of mobile centric applications into half [Perry, 2011]. The mobile browser pattern has become the de facto standard for mobile applications since the Web is everywhere. One key benefit of adopting mobile web methodology is the use of the latest

HTML5 oriented web technology frameworks. Web frameworks such as [PhoneGap, 2012] and [Sencha, 2012] support diverse mobile operating systems and allow mobile web developers to leverage their web technology skills in creating appealing applications. Moreover, these frameworks facilitate dynamic access capabilities to the device native features [Feldman, 2011]. As a result, mobile web applications nowadays are gaining much popularity among the applications developers across several device platforms as well as in Pub/Sub system environment in disseminating information. Two of such strategies are – pull and push. In the pull approach, communication is initiated by information consumer whereas the push approach relies on information producer in initiating the communication [Mühl, 2004]. Several web technologies are found to implement pull and push strategies. Three of such strategies are conceptually known as polling, long-polling and WebSockets.

2.2 Web-based Communication Technique

A real-time web application must receive up-to-date information. When the client browser (consumer) sends HTTP requests to the server (publisher) over a TCP connection, server acknowledges the request and issue a response back to the client.

2.2.1 Polling Technology

Polling is one technique introduced in delivering real time information. In this technique, the client browser sends HTTP requests to the server at a regular time interval and every time the server receives a request, responds back to the client as shown to Figure 2.2 [Hamalainen, 2011]. This approach is suitable in a situation when the server update interval is known to the client so that the client can be synchronized to send request to the server based on the exact interval of message delivery. There is also a growing need for asynchronous communication in collaborative applications where multiple users interact real-time among themselves. To response to this need, the Ajax technique has been introduced which enables web browsers to fetch dynamic information from the server asynchronously using in-built JavaScript functionalities such as XMLHttpRequest [Hamalainen, 2011]. However, although Ajax solves the problem of collaborative communication, its intense communication with the server causes significant overhead especially when using the polling technique [Gutwin at al., 2011]. As it is difficult to predict update interval of message dissemination in real-time application, polling data from the server with a long interval can make the communication slower whereas polling data with a short interval can result in many unnecessary HTTP requests with empty responses which causes lots of unnecessary HTTP responses.

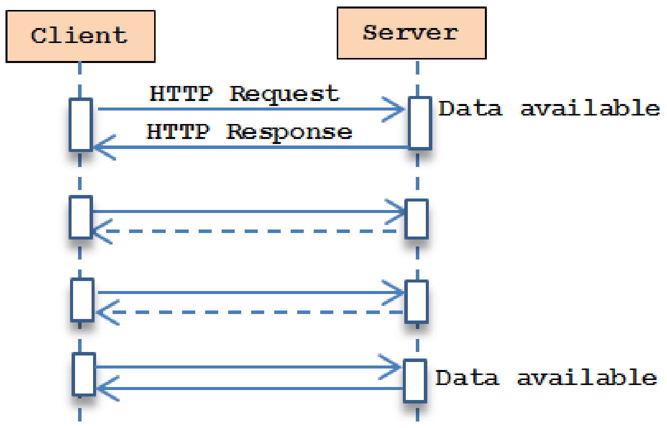


Figure 2.2: Polling Communication Protocol [Hamalainen, 2011]

2.2.2 Long-polling Technology

Long-polling addresses the limitations of polling by avoiding sending request in an interval. In long-polling, as the browser initiates a HTTP connection with a server, the server maintains the connection persistently for a certain period of time and pushes the update message to the client whenever it becomes available [Hamalainen, 2011]. If the update is not available within the set period of time, the server sends an empty response message as it times out and the connection is terminated. The browser then has to re-open another HTTP connection to send the next update request. The long-polling mechanism is depicted in Figure 2.3 In the asynchronous long-polling operation; the server can push update messages to the browser without the client prompting. However, performing long-polling in a groupware application where data is constantly updated will result in no improvement over the traditional polling technique as long-polling throttles the connection with lots of intermediate requests that consumes server resources [Lubbers, 2010]. Bayuex specification defines a Pub/Sub model for Comet [Dionysios, 2008].

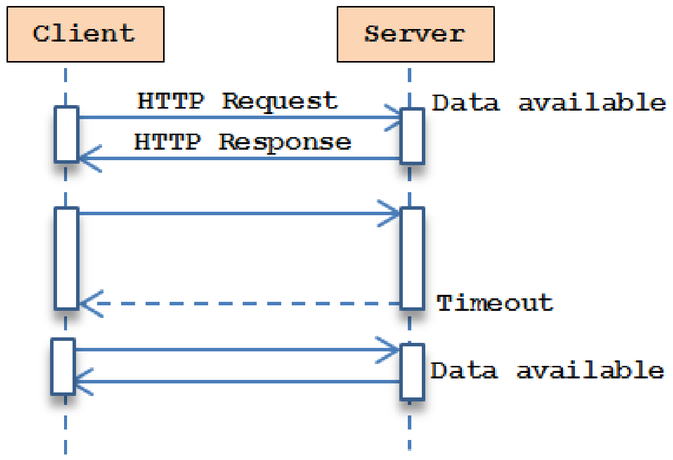


Figure 2.3: Long-Polling Communication Protocol [Hamalainen, 2011]

2.2.3 WebSocket Technology

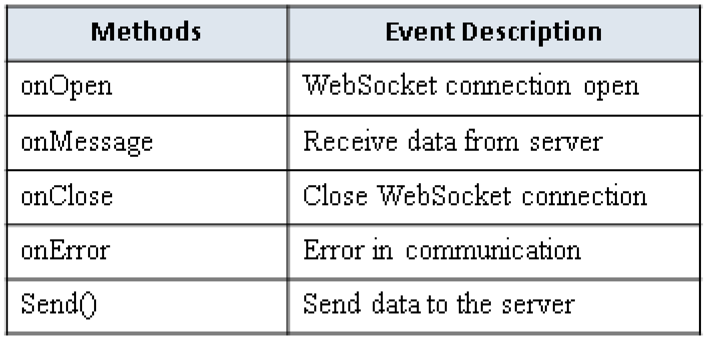
One of the latest web technology concepts introduced in the HTML5 standards as a new approach for the next generation web communication is WebSocket. It provides a full-duplex bi-directional asynchronous communication channel between web browser and web server applications over a single TCP socket per end point [WebSocket.org, 2012]. In addition, it has added the socket functionality to the browser to eliminate many problems of existing technologies. The complete WebSocket standard is the combination of the WebSocket API and the WebSocket protocol.

- **W3C WebSocket API**. The WebSocket API is a draft specification standardized by W3C [WebSocket API, 2012]. The API defines a communication interface between the web application and the browser [Hamalainen, 2011]. The browser must expose the API to the web application so that when initiating a WebSocket connection the application invokes the following API to create a WebSocket object.

```
var WS = new WebSocket (url, [protocol]);
```

Using the object, application then invokes the WebSocket API functions to open and close connection as well as send and receive messages as shown in Table 2.1. Current the browsers that support WebSocket standard are Firefox 6, Google Chrome 16, and Internet Explorer 10 [WebSocket, 2012].

Table 2.1: WebSocket API [WebSocket API, 2012]



Methods	Event Description
onOpen	WebSocket connection open
onMessage	Receive data from server
onClose	Close WebSocket connection
onError	Error in communication
Send()	Send data to the server

- **WebSocket Protocol.** The WebSocket protocol is proposed and standardized by IETF as RFC6455 [WebSocket Protocol, 2011]. The protocol has been designed to improve the existing HTTP connection. Two primary tasks that this protocol performs are establishing connection through handshake and transferring data. Figure 2.4 shows the header fields of the initial handshake between the client and the server.

Handshake from client (Browser request):

```
GET /chat HTTP/1.1
        Host: server.example.com
        Upgrade: websocket
        Connection: Upgrade
        Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
        Origin: http://example.com
        Sec-WebSocket-Version: 13
```

Handshake from server (Server Response):

```
HTTP/1.1 101 Switching Protocols
        Upgrade: websocket
        Connection: Upgrade
        Sec-WebSocket-Accept:s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

Figure 2.4: The WebSocket Handshakes [WebSocket Protocol, 2011]

The initial handshake starts with a HTTP protocol. The client and server then upgrades the HTTP protocol to the WebSocket protocol as shown in Figure 2.5. In the browser request, the GET method indicates the end point of the connection. The WebSocket server uses values from headers sec-WebSocket-Key to calculate a hash value and send it to the client to prove that the handshake was received and sec-WebSocket-accept header field indicates whether or not the server accepts the connection. Once the handshake between the client and the server is successfully established, the connection is ready for data transfer. In the WebSocket protocol, data is composed of sequence of frames which can be of type texts, interpreted as utf-8 text, binary data and control frame. Control frames are texts that are intended for signaling the connection for instance when the connection should be closed.

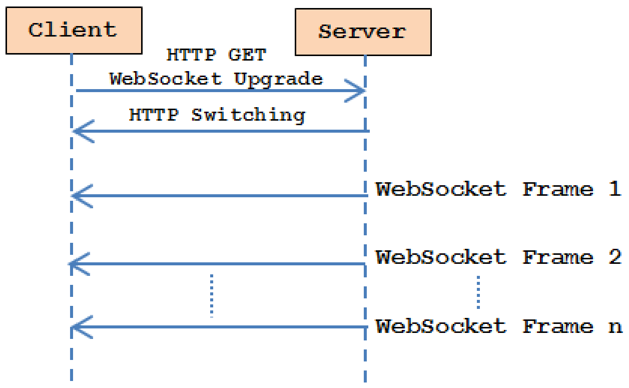


Figure 2.5: WebSocket Communication Protocol [Dionysios, 2008]

Since the WebSocket protocol uses a HTTP compatible handshake, it can also use a HTTP port as well as an underlying TCP protocol for network communications. The URI scheme used by WebSocket protocol is ws: for unencrypted communication that uses port 80 and wss: for encrypted communication that uses port 443. The current protocol version is 13 [WebSocket Protocol, 2011].

2.2.4 WebSocket based Pub/Sub System

Several web-based systems are found nowadays are using the WebSocket API and the protocol as the key implementation tool. A web-based control application using WebSocket is proposed by [Furukawa, 2011] that shows how a WebSocket-based application can be built with just HTML5 without using any add-ons in the web browser. Another work by [Cassetti and Luz, 2011] integrates the WebSocket API into an existing framework to support distributed and agent-driven data mining in an enterprise environment. The work is similar to R-WebSocket [HTML5 Websocket, 2011] except that it implements both the client and the server side interface for WebSocket API. The implementation uses Grizzy framework to provide scalability to the underlying infrastructure.

Young's [Hyuk, Y., 2011] study designs a Mobile Cloud e-Gov system as the part of a pan-government project of Korea. The study uses WebSocket API in order to provide full-duplex real time interactive communication the mobile participants. A WebSocket-based data binding framework known as WebSoDa is proposed in Matthias's work [Heinrich and Gaedke, 2011]. Integrating server-side updates are quite challenging in a distributed network. Matthias's WebSocket standard-compliant framework simplifies this task by efficiently integrating HTML5 declarative binding expression through the WebSocket protocol. Another in-depth study is conducted by Qveflander [Qveflander, 2010] in his master thesis on pushing real time data using HTML5 WebSockets. The study focuses on issues such as scalability and load balancing in finding optimal performance of a client/server application implementing WebSocket to provide real time data. Study results show that, a maximum of 400 clients connecting to a single server consists of Intel core 2 Quad 2.5GHz CPU and 4GB RAM can be handled with an average CPU power of 12.02%.

While addressing several research works, it is also noteworthy to mention the Kaazing WebSocket Gateway [Kaazing, 2012], which is the only enterprise solution available in the market to this date. It provides a complete feature of WebSocket protocol that addresses the key protocol level supports including scalability, availability, security and load balancing.

2.3 Web Services

System applications of a large enterprise solution are often distributed and independent. The continuous growing applications often need to communicate with the legacy system. Hence one of the biggest concern and a widespread need of these enterprises is to deduce solutions for integrating these applications so that they can work together. Among the most common methods of providing web services, the most common are XML-RPC, SOAP and REST.

2.3.1 XML-RPC

RPC stands for Remote Procedure Calls is an inter-process communication mechanism that allows an application to execute a procedure or sub-application that resides in a network address other than its own. It’s a programming style that allows developers to program the call of a remote procedure to design the same way as of a local call and therefore not to concern on network details. Some of the example technologies that follows RPC pattern are namely RMI, CORBA and DECOM. XML-RPC protocol uses XML for invoking remote procedure and receives XML as a return. XML-RPC uses HTTP as the transport protocol. JSON-RPC is a sibling of XML-RPC that uses JSON instead of XML.

2.3.2 SOAP/WSDL

The next standard functionality evolved from XML-RPC is SOAP. The main difference from XML-RPC is that SOAP relies on Web Service Descriptive Language (WSDL) in describing the service. Specification of SOAP is more complicated than XML-RPC. SOAP messages are more structured. It includes an envelope that defines the message and set of encoding rules that express the convention in representing the remote procedure call and their responses. SOAP uses HTTP, SMTP or TCP as the transport protocol.

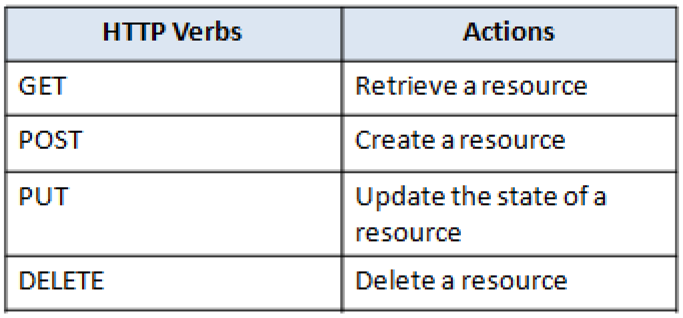
2.3.3 REST

REST is a prominent web service design model first introduced in 2000. REST stands for Representational State Transfer. The term is coined by Roy Fielding as an architectural style with some defined principles in consuming Web services on the Web [Fielding, 2000]. The architectural design of REST can be seen as a virtual state-machine (set of web pages) where the state transition occurs as user progresses through the application which results the application to render the next state as user go to the next page [Sudha and Sujata, 2011]. Anything in the web that we exposed to, whether it is an image or video clip or business process, is considered as a resource. In REST it is said that a resource must have at least one representation and every representation indicates to only one resource.

Resource Representation: Illustrates the view of a resource's state at any instant in time [Webber et al., 2010]. Views are expressed in a machine readable and transferable format such as XML, XHTML, and JSON. Representations are not the same as the resource object; it is the information about the resource object that mediates in accessing that resource. Therefore, in consuming resources, web components using URI exchange the representation of a resource in either of the above mentioned format. Separating the concept of the resource object and the resource information results into flexibility in consuming the resource since the consuming applications and the backend components become more loosely coupled. The two common data representation format used in REST are XML and JSON. How resources are identified, modified or managed are controlled by web services through encapsulating the resource information in XML or JSON based document.

REST Communication: In order to interact with the resources in the Web, REST uses http verbs explicitly provided by HTTP methods which indicate the actions taken on the resource. Following table (Table 2.2) shows the basic http verbs and their mapping to create, read, update and delete (CRUD) operations.

Table 2.2: HTTP Verbs



HTTP Verbs	Actions
GET	Retrieve a resource
POST	Create a resource
PUT	Update the state of a resource
DELETE	Delete a resource

REST Principles: Based on the design principles, RESTful web services can be characterized as follows,

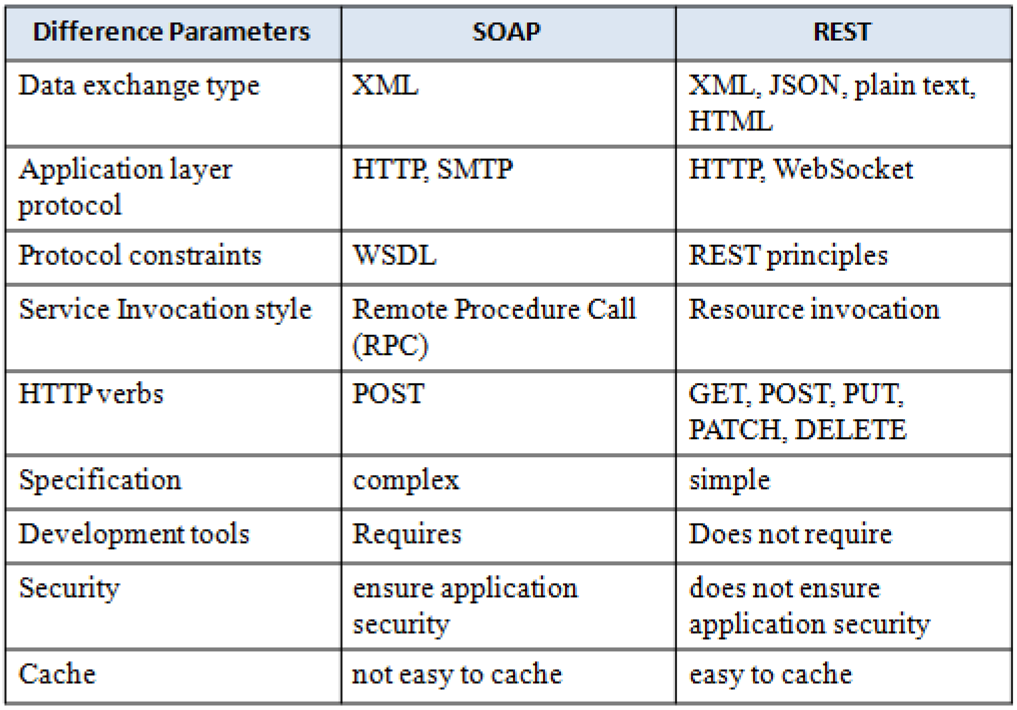
- **Uniform interface.** REST offers a uniform interface in the interaction among the network components and distinguishes itself from other network-based style. Through exploiting small number of verbs (GET, POST, PUT, DELETE), the term ‘uniform interface’ describes how a precise request semantics can be well-defined in meeting the requirements of a distributed application [Webber et al., 2010; Fielding, 2000].
- **Stateless communication.** A RESTful communication is said to be stateless since every client request for a resource representation must provide all necessary information about that resource that make the request comprehensive to the server. As a result, unlike contemporary client-server communication, server does not need to keep any contextual information (state) stored in its storage about the current as well as previous requests. This approach is also known as client-stateless-server (CSS) [Fielding, 2000] style that avoids server from being overloaded with information coming from client request.
- **Scalability and performance.** Since REST architecture uses HTTP as its base protocol; it may often seem less effective for applications where latency and bandwidth are the critical success factor. The synchronous, request-response nature of HTTP may not seem to provide

better performance characteristic. However, due to the specific application semantics and the standard HTTP verbs, caching of response data is possible in REST which provides an immense horizontal scaling with a large amount of throughput [Wang, Q., 2011].

- **Cache.** REST implements cache constraints where the contents can be labeled as cacheable or not-cacheable in order to improve the network efficiency. This labeling is mentioned implicitly or explicitly within the response to a request. A client-side cache is given the right to reuse the data only when the server response is labeled as cacheable. This cache constraint is also known as client-cache-stateless-server (CCSS) style [Jamal, S., 2012].
- **Named Resources.** Resources in a system are named using a URL or an ID that is unique. Anything in the system named as noun is considered as a resource and must have one or multiple representations.
- **Layered components.** REST architecture is composed of layers in order to further improve the behavior for the network based system. Layered approach allows the system to restricting its knowledge within the boundary of a single layer and each layer is allowed to see its immediate layer only. This approach benefits the architecture in implementing shared cache among the intermediaries and in improving the scalability by distributing the loads of services.

The difference between the two most popular Web services to this date - SOAP and REST are as follows.

Table 2.3: Difference between SOAP and REST Web services



Difference Parameters	SOAP	REST
Data exchange type	XML	XML, JSON, plain text, HTML
Application layer protocol	HTTP, SMTP	HTTP, WebSocket
Protocol constraints	WSDL	REST principles
Service Invocation style	Remote Procedure Call (RPC)	Resource invocation
HTTP verbs	POST	GET, POST, PUT, PATCH, DELETE
Specification	complex	simple
Development tools	Requires	Does not require
Security	ensure application security	does not ensure application security
Cache	not easy to cache	easy to cache

2.4 Application Development Patterns

A good architectural pattern in developing software applications can ensure a better performance for resource constraint mobile device. In talking about application design, we often encounter the term ‘MVC’ which is a short form of Model-View-Controller. The concept was first invented at Xerox in 1970s [MVC, 2011] and was first implemented and documented by Trygve Reenskaug [Reenskaug, 1979]. The term has been muted from its origin and obtained different form of ideas and concepts based on its existence in several implementations and often it is recognized as an architectural pattern or architectural style in designing software. Although it is often confused with a design pattern, Fowler [Fowler, 2006] describes MVC as an architectural pattern where different kinds of design pattern can be used. An architectural design that is based on MVC produce a clear abstract framework in the system development process. This provides a clean separation between software components. Decoupling the software components helps developers in designing essential interactive applications with more flexibility and well-organization. Smalltalk-80 is an early implementation that took the concept if separating application logic from user interface [Krasner and Pope, 1988].

MVC is widely used in application's Graphical User Interface (GUI) development [MVC, 2011; Fowler, 2006] and is very important in designing Web application framework. Some of those applications are namely Sencha Touch and Java application frameworks e.g. Struts, Spring, and so on.

2.4.1 MVC Architecture

The classical MVC pattern is used in desktop application. According to the architecture, MVC pattern breaks the code of Web application into three basic parts as shown in Figure 2.6.

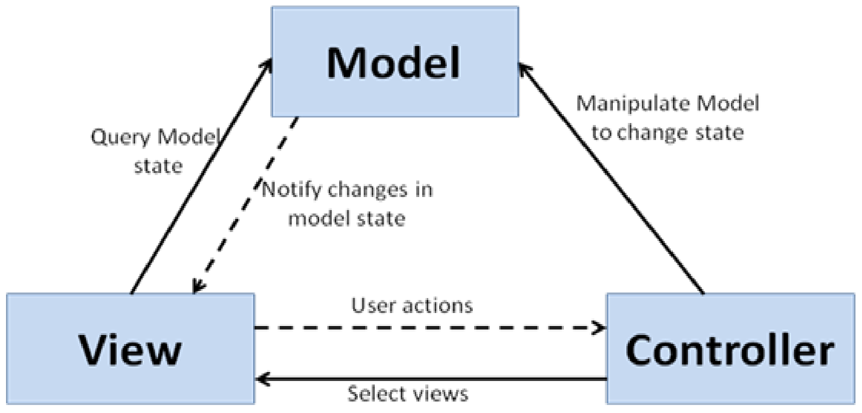

Model
Query Model state
Notify changes in model state
Manipulate Model to change state
View
User actions
Select views
Controller

Figure 2.6: Architectural design of Model-View-Controller [Gulzar, 2002]

- ***Model.*** In MVC, *model* represents application's data and also the business logic in accessing and manipulating those data [Gulzar, 2002]. It presents the current state of the application that resides in the model object. *Model* usually groups the data and the related operations that can be executed on those data in order to provide services. Services of model are exposed through model's interface methods that are generic to support various types of client in accessing or updating data. *Model* is not concerned with user interface or the controller layer that an application requires. However, as being one of the components of MVC, *model* is not completely isolated. When any state change occurs in the *model,* it typically notifies its associated viewer or observer of the change.

- ***View.*** *View* is responsible for rendering data from the model and forward user inputs to the controller. It manages the display of different types of information obtained from model component i.e. *view* is modifiable of its own as it encapsulates the presentation semantics and adapts with several types of client's user interface. *View* also acts as a presentation filter based on the current state of the information in model. For example – changing an image caption in the *model* can be executed by 'Edit' *view* where user can select a specific image caption in the metadata of the form [Osmani, 2012].
- ***Controller.*** Controller is the request handler that acts as an intermediary between *models* and *views*. It is responsible for updating the *model* when user manipulates the *view* and updating the *view* when there is change in the *model* [Osman, 2012]. It translates user actions and updates the *model* accordingly. Controller need to be designed to handle various types of user inputs.

The idea of MVC framework is having a single controller that controls the application based on the requests or arguments. An argument may define an event, invoking a *model* or a usual GET request in the web application. This concept of separating the *view* from *model* or separating controller from the view causes more decoupled, more flexible and maintainable application code. Different users can now participate and collaborate on application *model* which makes the application re-usable and cost effective.

2.4.2 Model-View-Presenter (MVP)

An evolved version of MVC is MVP, stands for Model-View-Presenter that focuses on improving the presentation logic/UI logic. The concept was originated in the early 1990s at a company named Taligent [Osmani, 2012].

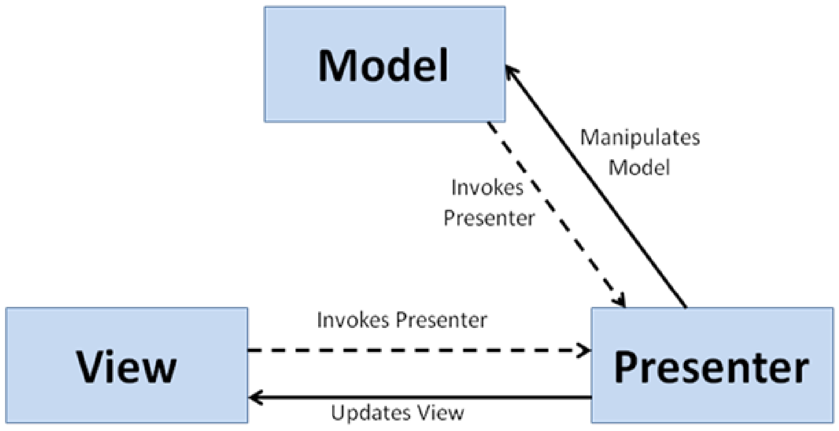
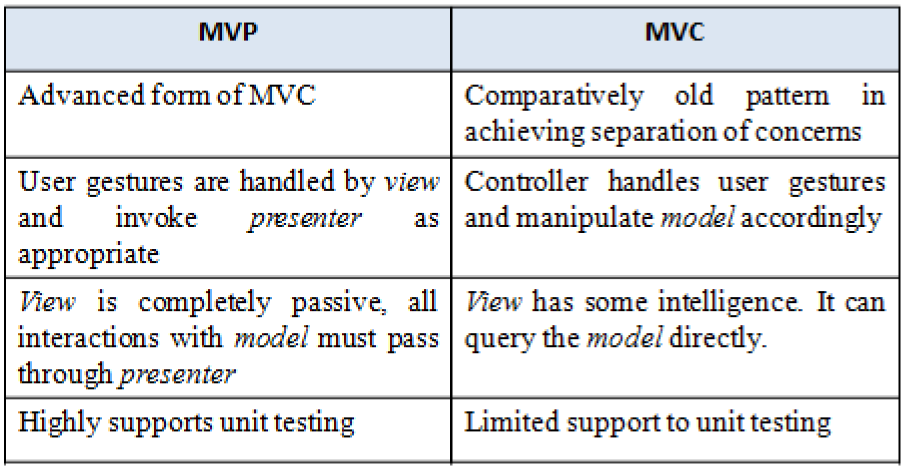


Figure 2.7: MVP Design Pattern [Osmani, 2012]

- ***Model.*** The model in MVP defines the data to be displayed or acted upon from the domain model or by accessing data (Figure 2.7).
- ***View.*** View is responsible to display data from model and also to route them in the presenter layer to act upon.
- ***Presenter.*** Presenter binds model to view by retrieving data from model and presents them to view.

Unlike MVC, the *Presenter* component in MVP contains the user interface business logic of the *View*. Communication between *View* and *Presenter* thus happen through a view interface. As the UI logic of the *View* is dedicated to the *Presenter*, a direct request from *Presenter* to *View* becomes possible. *Presenter* can trigger the *View* updates without visiting though the *View* component. This is often considered as a reason in taking MVP pattern most suitable for web-based architecture [MVCsharp.org, 2012]. The separation of concern in presentation logic helps *Presenter* to ignore implementation details of the *View* and only concern on the method to invoke of the *View* interface. This feature of MVP provides a higher level of abstraction which made it a successor to MVC. Moreover, the design pattern facilitates the developers for the unit testing of their programs. Differences between MVC and MVP are as follows.

Table 2.4: Difference between MVC and MVP design pattern [Emmatty, J.T., 2011]



MVP	**MVC**
Advanced form of MVC	Comparatively old pattern in achieving separation of concerns
User gestures are handled by *view* and invoke *presenter* as appropriate	Controller handles user gestures and manipulate *model* accordingly
View is completely passive, all interactions with *model* must pass through *presenter*	*View* has some intelligence. It can query the *model* directly.
Highly supports unit testing	Limited support to unit testing

2.4.3. Event-driven Programming

The traditional web application supports sequential flow of data where user had to fill a form and submit before showing the html content on the page. With the advent of AJAX, the modern UI of MVC/MVP supports event-driven style of data flow. User's action such as a button click or screen tap or screen swipe is sensed by the controller/presenter and performs some logics before viewing the data. These events to be processed need to pass through a dispatcher and managed by event handlers (Figure 2.8). As the stream of events arrives, the job of dispatcher is to determine the event type and pass it to the handler that can handle events of that type.

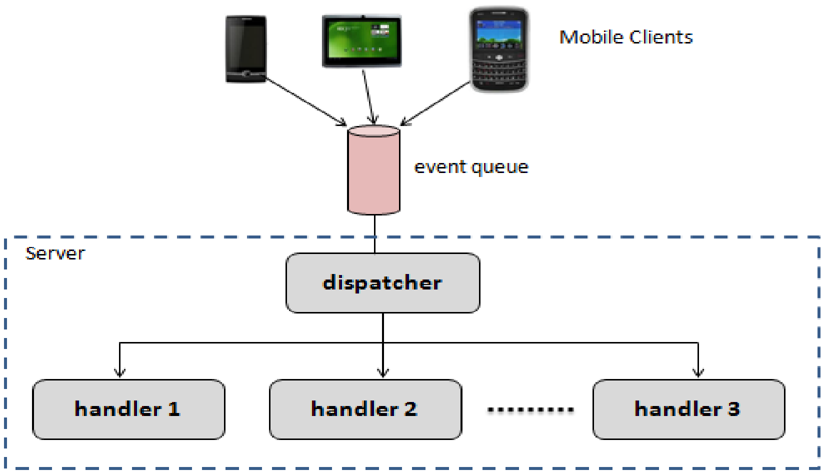


Figure 2.8: Event-driven programming style [Stephen, F., 2006]

In a client-server interaction, dispatcher and the event handlers may reside in the server side as shown in Figure 2.8. In that case, events from client's requests are queued up before transmitting them to the server to be processed. In event-driven, programs are like multiple individual modules that can be triggered based on the event types. The program is designed as a continuous loop that keeps listing for event and calls the event handler (also known as callbacks) that matches the event type.

2.5 Cloud Computing

Cloud offers a new paradigm in computation and an evolution of information of information technology where user's resources such as memory, storage are hosted in the remote centrally located datacenters instead of physically placing them at user's location. The datacenters consists of hardware and software that provides access to the general public for the services as pay-as-you-go manner also known as Public Cloud. On the other hand, the internal datacenters of a business or an organization which is not accessible by general public is referred as Private Cloud [Armbrust et al., 2009] The concept of cloud computing was evolved in order to achieve improvement over the existing internet computing. Ubiquitous broadband and wireless network, reducing storage cost are some key driving forces behind cloud computing. Based on the architectural design, cloud is mainly defined into three layers as shown in Figure 2.9.

Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a Service (IaaS)

Figure 2.9: Layered view of Cloud Computing [Hoang et al., 2011]

- ***Infrastructure as a Service (IaaS)*** is a provision model where the service vendors outsource hardware equipment, storage, network components in a usage-based pricing [Foster, 2008; Hoang et al., 2011]. Example of IaaS cloud services are Amazon EC2 (Elastic Compute Cloud) and Amazon S3 (Simple Storage Service).
- ***Platform as a Service (PaaS)*** is a service delivery model that allows cloud users to test the existing applications or build, test and deploy their own applications with some restrictions in the tools and programming language supported by the service [Foster, 2008; Hoang et al., 2011]. Google App Engine (GAE) and Microsoft Windows Azure, and Force.com are some examples PaaS providers.

- ***Software as a Service (SaaS)*** is software distribution model that allows users to access software applications hosted by SaaS providers in pay-as-you-go manner. Users can use the applications remotely over the internet without installing them in their local machine. Google Apps, Microsoft Office 365, Facebook, Twitter are some of the pioneered examples of SaaS applications.

The foundation of cloud computing is seen as a remarkable way in consuming web services in resource poor of mobile device by offloading resource intensive computation and data storage outside the device into resource rich remote machines [Ashik et.al., 2012] [Kazi, R.; Deters, R. 2013a]. The major advantages of MCC are thus seen in offloading computation and data storage. Computing in the cloud also provides scalable hosting of IT backend services. Several approaches have been proposed by myriads of research studies for the effectiveness of offloading techniques. Since the wireless signal may attenuate due to device mobility, these studies offer a notion of dynamic offloading that is said to be feasible in such network environment. [Chun and Maniatis, 2009] offers a cloud infrastructure that seamlessly offload execution from mobile device to a replicate copy of mobile application software running in the virtual cloud server. This approach of migrating computation from a device to a device replica gives mobile user an illusion of using powerful, feature rich device and also known as CloneCloud. Similar approach is proposed by [Satyanarayanan et al., 2009]. This study proposed to locate the cloud service software on a nearby resource-rich computer(s) called *cloudlets* that is well connected to the internet as well as to the mobile users. The approach of bringing the cloud virtual machine close to the mobile users is considered latency optimized in terms of latency and data transfer cost. In offloading mechanism, a fine grain offloading approach has shown in MAUI system [Cuervo, 2010] where instead of offloading on the whole application software, which methods to be executed remotely are decided in the runtime and thus saves energy and increase the battery life of mobile devices. [Cao et al., 2009] provides an ad-hoc cloud infrastructure where mobile devices host web services and expose their computation power to other mobile peers on the network.

Combining cloud computing and RESTful Web services provides a new paradigm of mobile computing. [Christensen, 2010] in his research specifies REST as a suitable architectural platform that lends itself well in consuming cloud Web services in resource constraint mobile device.

2.6 Summary

Integration of distributed system applications has always been a challenge for enterprise solution. Network connections are not always reliable and sometimes they suffer from intermittent connection loss and also slow connection. Applications in these systems are different in terms of the programming language and the environment where they operate. Therefore a change in the system integration is inevitable that can keep pace with the internal change of the system. Over time, developers have proposed different approach in integrating system in a distributed environment.

From the literature review it can be concluded that the channel based Pub/Sub is an ideal model for a distributed system where applications are disparate and dispersed over the network. The space decoupling nature of Pub/Sub enabled mobile applications and the interacting parties who use these applications to be anonymous and independent from each other. Publisher can publish events at any time without blocking themselves and subscribers are notified asynchronously through a callback. Publisher doesn't hold any reference of subscriber which let the publisher to publish events even when the subscriber is disconnected. This decoupling in production and consumption explicitly removes dependencies among the interacting participants and increases the scalability. The communication in Pub/Sub is asynchronous that well adapts with the distributed environment such as mobile environment.

On the other hand, Web services have been a great solution in integrating distributed and disparate system applications [Kazi, R.; Deters, R., 2013b]. Due to clear semantics and uniform interface and its supportability for different message formats, REST Web Services has become the most suitable approach in consuming services in mobile environment. REST avoids the single access point in consuming services and thus increases the service scalability. Reviewing the challenges in mobile distributed environment and the proposed solutions, this research attempts to address the following open issues;

- How can we build a RESTful Pub/Sub system in mobile environment?
- How much the system needs to comply with REST and Pub/Sub features to call it RESTful Pub/Sub?
- And because of operating in mobile environment, how can we ensure a system that is fault-tolerant and yet efficiently disseminate information?

The summary of some of the concepts that has been reviewed in this chapter has been categorized in Table 2.5.

Table 2.5: Summary of literature review

Reviewed Concepts in Literature addressing research challenges	
Pub/Sub System	- Model in disseminating information as event messages [Liu et al., 2010, Baldoni and Virgillito, 2005, Cugola and Jacobsen, 2002, Huang, Y., Molina, G., 2001] - Topic and content-based subscription scheme [Baldoni and Virgillito, 2005, Cugola and Jacobsen, 2002] - Pub/Sub in Mobile Environment [Huang, Y., Molina, G., 2001.], [Anceaume et al., 2002], [Cugola and Jacobsen, 2002], [Fiege et al., 2003] - Implementing Pub/Sub system on mobile browser [Mühl, 2004]
Web-based Communication technique in Information dissemination	- Strategies – pull and push [Mühl, 2004] - Polling Technique [Hamalainen, 2011], [Gutwin at al., 2011] - Long Polling [Heimbigne, 2003], [Hamalainen, 2011] - WebSocket [WebSocket API, 2012], [WebSocket Protocol, 2011], - Web Socket in Pub/Sub system [Furukawa, 2011], [Cassetti and Luz, 2011], [Hyuk Y., 2011], [Heinrich and Gaedke, 2011], [Qveflander, 2010]

Web Services	- WS Architecture [W3C 2004] - WS model [Sudha and Sujata, 2011], [W3C, 2004] - RESTful WS [Webber et al., 2010], [Fielding, 2000]
Cloud Computing	- Cloud Computing [Armbrust et al., 2009], [Foster, 2008], [Hoang et al., 2011] - Mobile Cloud Computing [Chun and Maniatis, 2009], [Satyanarayanan et al., 2009], [Cuervo, 2010], [Christensen, 2010]

CHAPTER 3

DESIGN AND ARCHITECTURE

The chapter looks into different REST patterns in event dissemination in accordance to the challenges mentioned in problem statement (section 1.2) and then propose a framework that is adopted for mobile clients to consume RESTful Web Services within an event-based Pub/Sub domain. The proposed framework is designed in three main layers as shown in Figure. 3.1.

3.1 Proposed RESTful Pub/Sub Framework

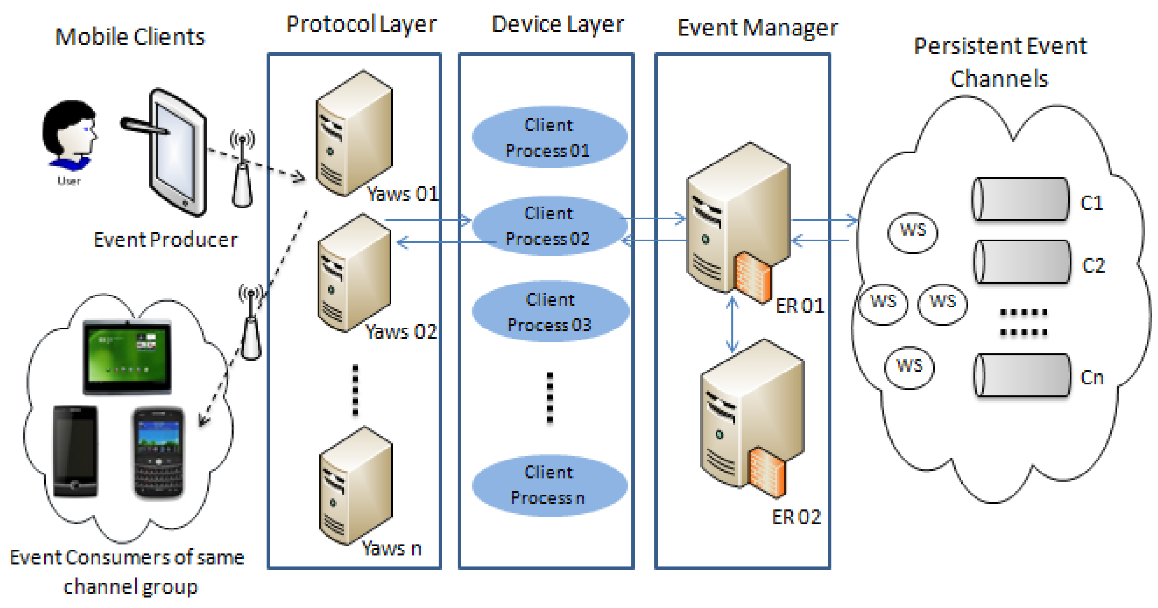


Figure 3.1: RESTful Pub/Sub System Framework

The front-end of the framework represents mobile clients who are publishers and/or subscribers of data at the Web Service (WS) channels. The backend of the framework contains Web servers as Protocol layer and Device layer, Event Manager and the cloud hosted Web Services channels. The Web servers and Event Manager act as a proxy layer between mobile clients and WS channels. Since we adopt a Pub/Sub model, data are disseminated in the form of events. Similarly, a mobile client that publishes events is known as the Event Producer (EP) and subscribers of these events are labeled as the Event Consumer (EC). However, an event consumer can be an event producer and vice versa. In this framework, topic-based persistent event channels were adopted. In topic-based persistent event channels, event producer publishes events to a specific channel topic and the event consumers show their interests for events by registering to a specific channel topic.

Event channels are collections of events represented by the event topic. In the Pub/Sub model, events are published using a single input channels that splits into multiple output channels to multicast the events to each subscriber. In the application-level, mobile client applications include User Interface (UI) layout, the business logic, and the model for managing a local storage. A stub component in the client model interacts with the skeleton of the server application. The persistent event channels are fronted with the Event Router component that takes the responsibility of multicasting events to the mobile subscribers. The layered view of the proposed application-level architecture is shown in the Figure 3.2.

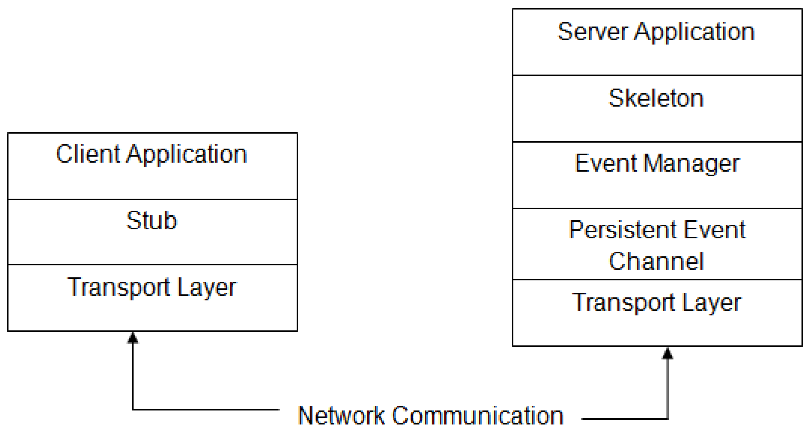


Figure 3.2: Layered view of the architecture

In Figure 3.2 above, the client application includes a UI layout, the business logic and the local storage capability. The client stub provides the functionalities of the backend server on the local device. On the contrary, the skeleton on the backend server describes the functionalities of the server application. The actual implementation of the skeleton is done at the persistent event channel. Further, the Event Manager works as an intermediary between the skeleton and the persistent event channel. All message exchanges between the client device and the remote server takes place over the standard TCP/IP transaction layer.

3.2 Event Dissemination patterns based on Richardson's REST maturity level

According to the Richardson's Maturity Model (RMM) [Fowler, 2010], a RESTful dissemination of data can take four different patterns based on REST Web Service's maturity level also known as the glory of REST.

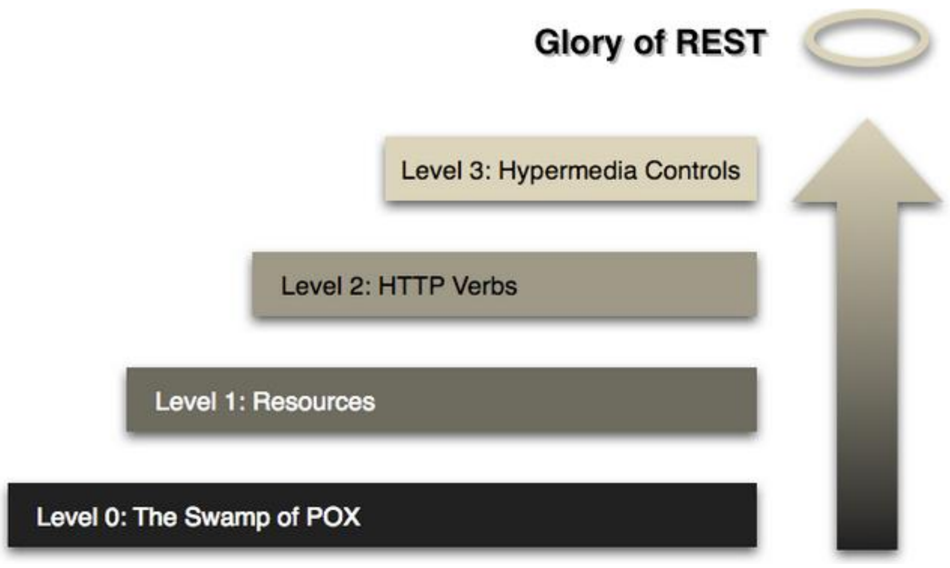


Figure 3.3: RESTful maturity levels by Leonard Richardson [Fowler, 2010]

In the context of the proposed framework in this thesis, the patterns are hereby discussed as follows;

3.2.1 Pattern A: Using HTTP POST (Level 0)

Event-dissemination of this pattern follows level 0 of the RMM. In this pattern, services are exposed using one URI; and consumers can access the URI using a single HTTP POST method. This is similar to SOAP based WS where requests are sent to one URI and XML payloads are exchanged between the sender and receiver. According to this pattern, an event publish request to a channel looks as follows;

```
POST /channelService HTTP/1.1
{
 "event_type":"channelPublishRequest";
 "event_date":"20-01-1013";
 "channel_topic":"c1" ;
 "event_message": "…data…"
}
```

The server response for a successful request will be as follows,

```
HTTP/1.1 200 OK
[message headers…]

{
"event_type":"channelPublish";
"channel_topic": "c1";
"event_version":"event_v1";
}
```

All these requests are sent to the single URI /channelService. Details of the requests are served in the message body.

3.2.2 Pattern B: Using HTTP GET or POST (level 1)

Event dissemination of this pattern is based on level 1 of the RMM. In this pattern, a service is exposed as many logical resources with unique URIs contrary to single resource/service of level 0 (pattern A). A request is sent either using HTTP POST and/or HTTP GET. An event publish request to a channel looks as follows;

```
POST /channel/c1 HTTP/1.1

{
 "event_type":"channelPublishRequest";
 "event_date":"20-01-1013";
 "event_message":"…data…"
}
```

The server response of a successful request will be as follows,

```
HTTP/1.1 200 OK
[message headers….]

{
  "event_type":"channelPublish";
  "event_version":"event_v1"
}
```

In this pattern, operations can be performed using HTTP POST. Sometimes HTTP GET is used in addition to HTTP POST. However, HTTP verbs do not strictly follow HTTP rules or REST constraints in this pattern. As a result, the verb "GET" can be misused in a way that can cause a service to change its state.

3.2.3 Pattern C: Using HTTP CRUD Operations (level 2)

Services in this pattern host numerous URI-addressable resources. Unlike level 0 and 1 of the RMM, coordinating interactions in this pattern utilizes all the HTTP verbs (GET/retrieve, POST/create, PUT/update, DELETE/delete) in performing the CRUD operations. A response message in this communication utilizes the http status code. A channel publish request in this dissemination pattern looks as follows,

```
POST /channel/c1 HTTP/1.1
{
 "event_type":"channelPublishRequest";
 "event_date":"20-01-1013";
 "event_message":"…data…"
}
```

The response to a successful request looks as follows,

```
HTTP/1.1 200 OK
[message headers….]

{
  "event_type":"channelPublish";
  "event_version":"event_v1"
}
```

3.2.4 Pattern D: Using Hypermedia (level 3)

Pattern D is similar to pattern C in a way that it utilizes all the HTTP verbs in performing the CRUD operations except that it also utilizes the hypermedia element of the HTTP stack of the Web technology in the response message. A published request according to this pattern will look as follows;

```
POST /channel/c1 HTTP/1.1
{
 "event_type":"channelPublishRequest";
 "event_date":"20-01-1013";
 "event_message":"…data…"
}
```

The response of this request looks as follows,

```
HTTP/1.1 200 OK
[message headers….]

{
  "event_type":"channelPublish";
  "event_version":"event_v1";
  "link":{
          "rel": "/linkers/channel/channel_topic/eventMessages";
```

```
            "url": "/channel/c1/eventMessages/event_version/";
          };
  "link":{
            "rel": "/linkers/channel/channel_topic/eventDelete";
            "url": "/channel/c1/event_version";
          }
}
```

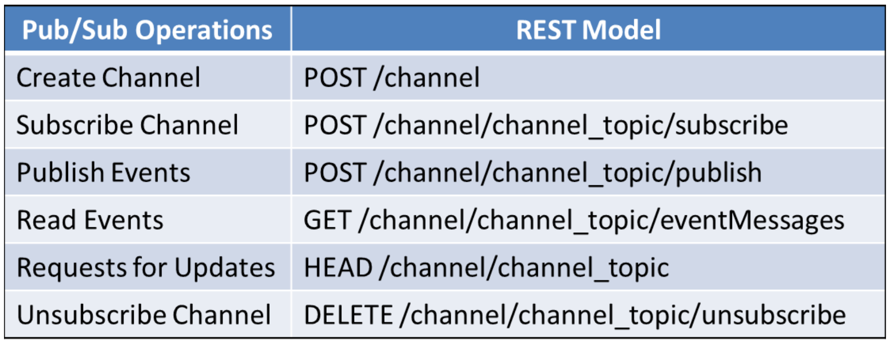
From these four patterns of event-dissemination based on the RMM it can be observed that consuming services in pattern A and B requires service requesters to know the exact location of the service or resources.

3.3 Modeling Pub/Sub operations in REST according to the RMM level 3

Consuming services in a Pub/Sub framework can be challenging when complying with REST features described in chapter 2. This section describes how interactions can take place in REST-based manner in the proposed Pub/Sub based framework. Interactions between Web services and the service consumer are described in terms of major functionalities provided by the Pub/Sub service.

Table 3.1 shows how operations of a Pub/Sub model can be mapped into REST services.

TABLE 3.1: REST representation of Pub/Sub operation



Pub/Sub Operations	REST Model
Create Channel	POST /channel
Subscribe Channel	POST /channel/channel_topic/subscribe
Publish Events	POST /channel/channel_topic/publish
Read Events	GET /channel/channel_topic/eventMessages
Requests for Updates	HEAD /channel/channel_topic
Unsubscribe Channel	DELETE /channel/channel_topic/unsubscribe

- **Creating a Channel.** Channel creation is accomplished by using the HTTP POST request. An event publisher when creating a channel uses the host's Channel service to create the channel. Figure 3.4 shows the interaction between the event publisher and the backend server.

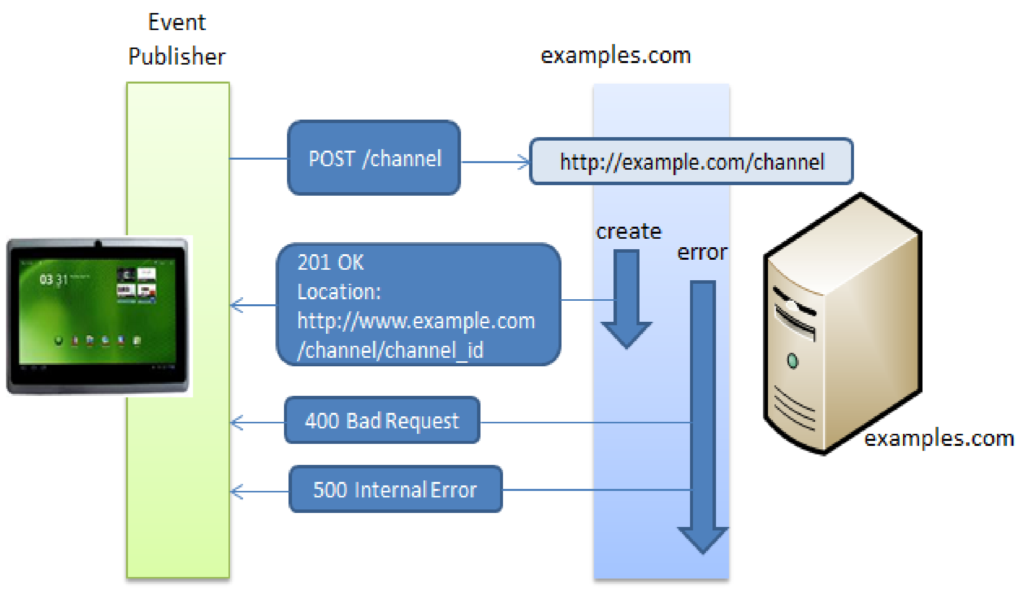


Figure 3.4: Message flow while creating a channel

The following shows the network-level view of a request-response in creating a channel;

```
Request:

  POST /channel HTTP/1.1
  Host: www.example.com

  Response:

  HTTP/1.1 201 Created
  Location: http://www.example.com/channel/channel_topic

  {
   "event" :
     "link" :{
```

```
              "rel" : "/linkrels/channel/event_publish"
              "url" : "/channel/channel_topic/eventmessage"
              }
      "link" :{
              "rel" : "/linkrels/channel/subscription"
              "url" : "/channel/channel_topic/subscription"
              }
}
```

- **Subscribing to a Channel.** The usual procedure of subscribing to a channel in Pub/Sub domain is creating a SUBSCRIPTION method that is invoked upon client's subscription event. In this REST Pub/Sub framework, channel subscription is handled by the HTTP POST request. In order to obtain an existing channel address, an Event Consumer (EC) first sends a GET request to the service host as follows,

```
Request:
GET /channel HTTP/1.1
Host: www.example.com

Response:
  HTTP/1.1 200 OK
  Location: http://www.example.com/channel/channel_topic
  {
   "event" :{

     "link" : {
               "rel" : "/linkrels/channel/subscription"
               "url" : "/channel/channel_topic/subscribe"
              }
     "link" :  {
                "rel" : "/linkrels/channel/event_publish"
                "url" : "/channel/channel_topic/eventmessage"
               }
            }
  }
```

The URI relation in the response message tells how the resource can be manipulated. Using the URI received from the response, the event consumer sends subscription request as follows,

```
Request:

POST /channel/channel_topic/subscription HTTP/1.1
Host: www.example.com
{
     "subscriber_id" : "deviceid_001"
}

Response:

HTTP/1.1 200 OK
Location: http://www.example.com/channel/channel_topic/subscription
```

EC is subscribed to the channel using its device ID. In case of unsubscribing from the channel, EC uses the same URL location to DELETE its subscription interest using HTTP DELETE like,

DELETE /channel/channel_topic/subscription?subscriber_id= "deviceid_001" HTTP/1.1

- **Publishing Event Messages to a Channel**

Event messages can be published by both the event publisher and event publisher to the subscribed channel using HTTP POST. A publish request when sent to the URI is received as a response in both the CREATE and SUBSCRIBE operation. The request of a publish operation in a network-level view looks as follows,

```
Request:
  POST /channel/channel_topic/eventmessage
  Host: www.example.com

  {
   "event" :{
             "data" : {
                        "...message…"
                       }
            }
  }
```

A PUBLISH operation is followed by a Notification message that is delivered to the subscriber. Instead of using HTTP POST, notification is sent by invoking the NOTIFY method when a resource is added into the channel group. According to [Thomas, et al., 2012], using HTTP POST leaves possibility that a malicious subscriber could substitute its own notification services with another vulnerable services notification system. A notification message contains the name and the creation time of a resource.

3.4 Backend System Architecture

The backend server is responsible for hosting Pub/Sub Web Services. Web Services enables clients to create event channels (event groups) and publish events to the channel, subscribe to the channel(s) of their interests, be notified for resource updates of the channel and also unsubscribe from the channel. The system architecture takes a centralized topic based Pub/Sub model. The major functional components of the framework backend are shown in Figure 3.5 and discussed below.

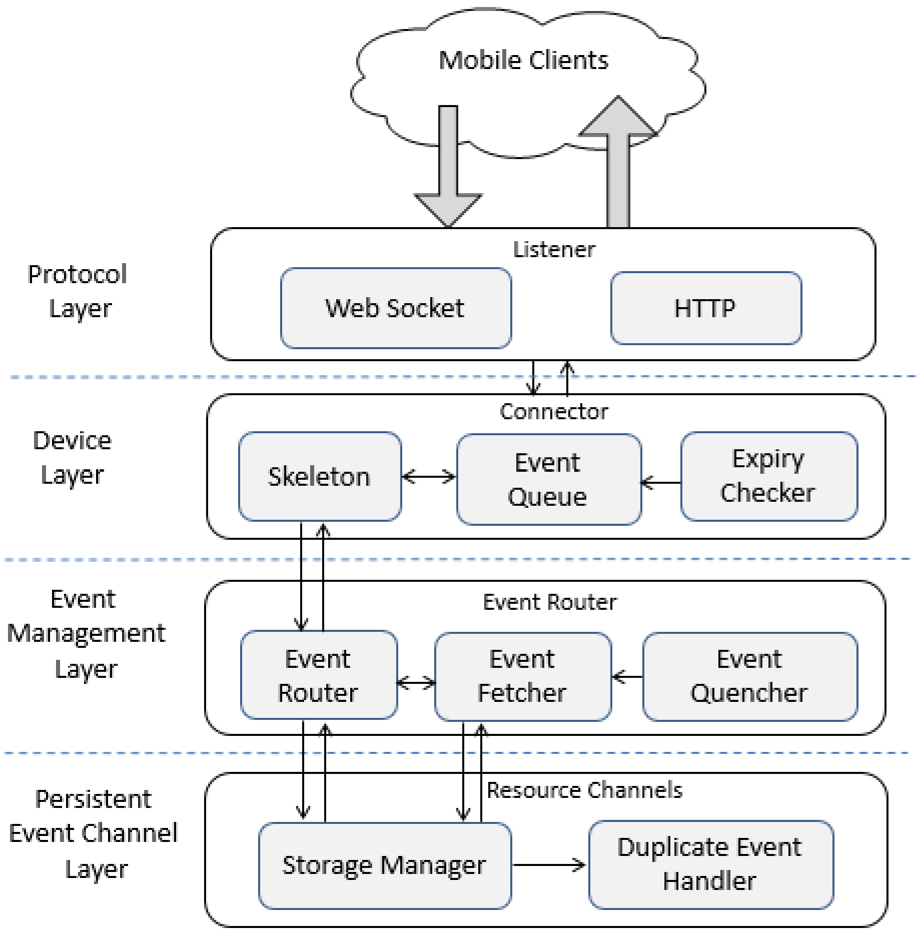


Figure 3.5: Pub/Sub Backend System Components

a) Protocol and Device Layer. When an event is published in the event channel, it needs to be propagated as an update notification among respective subscribers. A published event is composed of event type, $e\textit{type}$; published time, $e\textit{timestamp}$ and event messages, $e\textit{message}$ (payload).

$$\text{Event, } e = \{e\textit{type}, e\textit{timestamp}, e\textit{message}\}$$

A published event is received by the Listener before it is transferred to the Event Manager (EM) process. It contains separate request handler for compatible transport mechanism. The expected transportation mechanism is the standard HTTP connection and/or WebSocket connection. Since mobile clients are using different types of device platforms, the embedded browser of native device application may not support either of this connection at any given time. To provide device transportation compatibility, a Listener process manages the request handlers for both HTTP and WebSocket.

The device layer is responsible for redirecting client requests to the web services for appropriate operation execution using the connector process. This helps mobile consumers to maintain a presence at the proxy when they are disconnected and thus resume the interaction with backend once the connection has been restored. The skeleton component of device layer provides the interface layer for Pub/Sub service, describing the functionalities that the service provides.

The Event Queue (EQ) component of the device layer buffers event update notifications received from Event Manager. It also handles duplicate event notifications to cope with network inconsistency. Event notifications are buffered in the queue until it has been propagated to the client device in FIFO style. An event is persistently removed from the queue once it is delivered to the consumer.

Notifications in the Event Queue might become obsolete when event consumer is disconnected for relatively a long period of time. An event that is too old than the expected event longevity, need to be discarded from the event queue. The Expiry checker in the layer does a periodical checking in the event queue to ensure that no event notification in the queue is obsolete. Device layer is also stores event data into the process storage based on their notification IDs.

b) Event Manager (EM). The Event Manager is responsible to route event notifications to all the users who are subscribing to the channel group. Once an event is published to the persistent event channel, Event Manager invokes the Event Fetcher (EF) to fetch the list of all subscribed users of that channel. Consequently, the Event Router (ER) is invoked to actually send event notifications to the users from the subscription list. Dissemination of event updates takes a broadcast approach in delivering data to all currently active subscribers.

The Event Manager is also responsible to discard published events that arrives and does not match with the existing channel groups. An unmatched event is discarded when they are received at Event Manager. According to Huang's paper [Huang, Y., Molina, G., 2001], this approach is also known as event quenching. Discarding unmatched events considered to be advantageous as it does not require Event Manager or any of its replica (if any) to attempt transmitting irrelevant data to the persistent event channel over the network. Moreover, accomplishing this task at Event Manager also reduces computational workload at Event Channels.

c) **Persistent Event Channel (PEC).** The Persistent Event Channel handles consumer's request for subscribing to the channel, unsubscribing from the channel, publishing event messages to the channel and also delivering event from the channel.

Event Channels maintain persistent data storage for event messages published by event producer. All published event requests are sent to duplicate event handlers to check for duplicate event messages to avoid network connection delay. This can be done by checking the event ID that has been assigned by event producer's application. An event with unique event ID is stored in the channel storage persistently. Each event in the channel is uniquely identified by its URL. And thus each event resource can be accessed my consumer by sending http requests using the standard http verbs such as HEAD (meta-data), GET (read), PUT (replace), POST (create and write).

3.5 The Mobile Client Framework

In this architectural framework, mobile clients are thin clients such as smartphone and tablets. Applications for these devices are responsible to register themselves to a particular channel group or group of channels based on the channel topic by consuming the Pub/Sub web services hosted in the code. Once a device registers itself, it continues to receive event notifications for any updates made in the persistent channel. In order to provide code flexibility and interoperability, the client side application is designed following the Model-View-Presenter (MVP) pattern as shown in Figure 3.6. In this design pattern, the Presenter acts a mediator between the Model and the View components. A stub component of the backend server is hosted in the Model. The stub is responsible for all incoming and outgoing transactions. Once an event update arrives at the stub, the latter passes the event to the View’s logic through the Presenter to be displayed on interface layout. Likewise, event messages produced by client actions (e.g. button click) are passed to the stub through the Presenter which then transmits the data to the backend server.

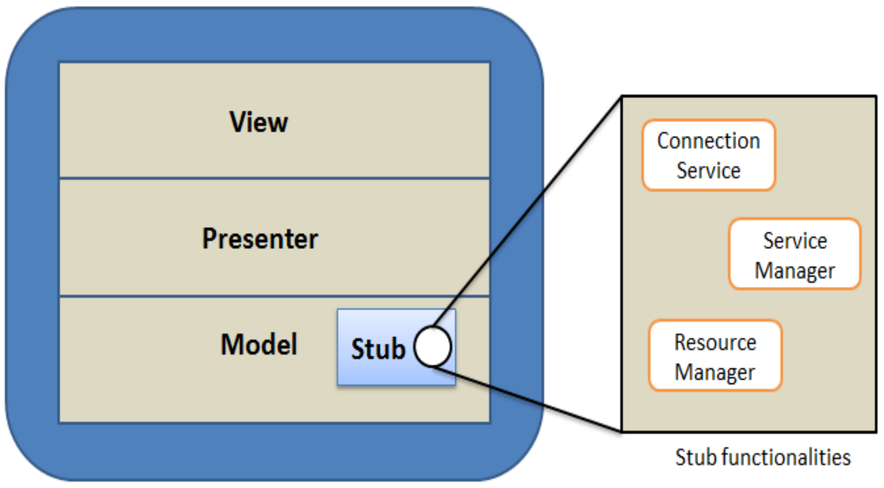


Figure 3.6: Mobile Client Architecture

The *Model* component of the client application is designed to contain a persistent storage for event notifications. Moreover, it contains a queue for unpublished events; events that are produced by the client actions but could not be delivered due to the connection loss. These unpublished events are removed from the queue once they are delivered to the backend server. All interactions between the *Presenter* and the *Model* take place though the stub. The major functionalities of a stub are as follows;

a) Connection service. The stub is responsible to connect mobile application to the proxy server. Whether the communication should take place over WebSocket connection or should it be http polling are decides by the stub.

b) Service Manager. The stub provides the same interface of the remote cloud hosted Pub/Sub web services. It binds client's application to the remote web services over Web. It also enables client applications to invoke the consecutive functionalities of the remote web services such as subscribing to Channel, publishing data, retrieving data or unsubscribing from channel in a way as if calling to local functions. All event messages generated by these actions are encoded into JSON format before they are transmitted between client and proxy.

c) **Resource Manager.** The stub is responsible to store update notifications to the local storage when it arrives from proxy. States of the stored event notifications are used to check for event updates at the proxy when a client application reconnects after an intermittent connection loss. Stub also checks for the unpublished events in the queue once after every connection establishment.

3.6 Update propagation over unreliable wireless network

The decoupling nature of event service in a Pub/Sub model does not require event producer and event consumer to hold any reference about each other. In other words, they do not have to actively participate to the event service at the same time since event production and consumption does not happen in the same main flow of event service [Eugster et al., 2003]. Hence the event producer is not blocked while producing event and subscribers of the event receive asynchronous event notifications. Data dissemination in this model is delimited while operating over an unreliable wireless network. In a case when network between mobile clients and backend messaging system is unavailable (as shown in Figure. 3.7), data needs to be stored persistently in order to provide guaranteed delivery.

In this framework, the guaranteed delivery is ensured by storing events in a NoSQL database in the cloud hosted channel where event keys are the event identifier/GUID/timestamp and event values are data that comes with the event messages.

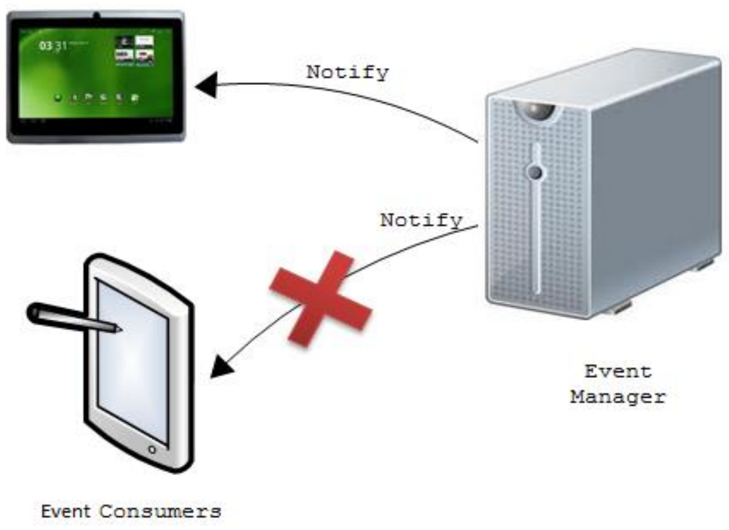


Figure 3.7: Update notification over intermittent Wireless connection

This limitation of inconsistent notification is avoided by having the consumer issue a new request to the main Channel resource. When a consumer reconnects to the network first sends a HTTP HEAD request to Event Manager Service to check if there is any updates available. HEAD request therefore contains the latest update notification version viewed by consumer and checks with the current version of the Channel resource. When notification version at the consumer matches the Channel version indicates that there have not been any updates in the Channel resource. If it does not match, Event Manager responds consumer with the resources that have been published in a later time than the received version (timestamp). Figure 3.8 shows the consumer-server interaction when requesting for new updates.

Request:

```
HEAD /channel/channel_topic/
Host: www.example.com
If-None-Match: "12:13:2013
```

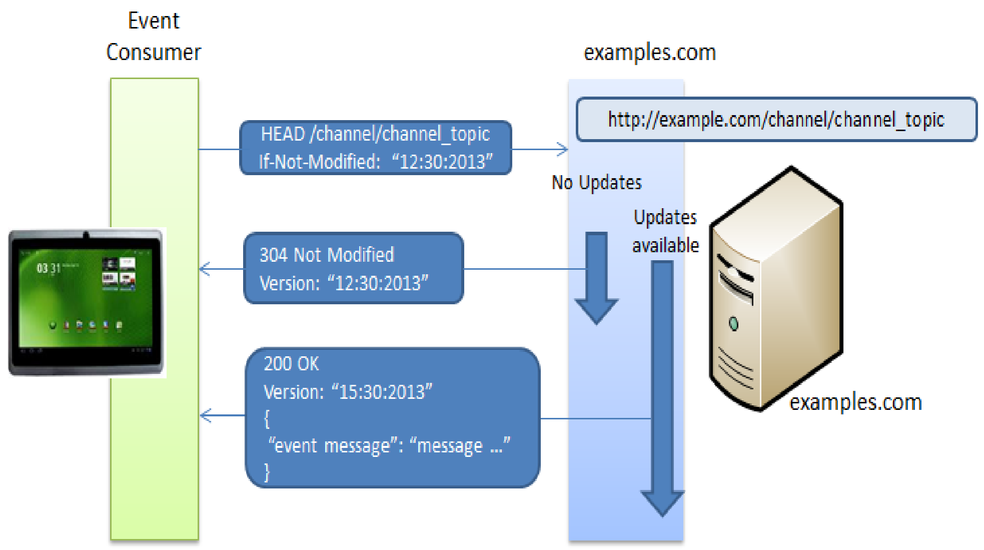


Figure 3.8: Message flow when requesting for event update

All published events are buffered in the event queue at the consumer's Web Server proxy channel until they are delivered. An event consumer's proxy channel is invoked on the arrival of an event message which is then delivered to the consumer using the consumer specific callback application. Our proxy channel offers a durable subscription that saves event messages for offline subscribers and helps subscribers to synchronize already received event states/event identifiers with the event state of the proxy channel buffer when they reconnect to the system.

In case the messaging system is down, the mobile clients maintain an event queue that buffers all unpublished events until they are delivered to the messaging system when the system is up and running. Generally, an event is considered to be delivered when an acknowledgement is received. Our mobile consumers are idempotent meaning that receiving of same message multiple times does not change client's state of the received events. When the ACK is not received, consumer's proxy resends the event message to the client application. In that scenario, the event buffer of the client application is used to detect and eliminate duplicate events.

3.7 Summary

In this chapter, a Pub/Sub model based architecture has been proposed in disseminating data that models client-server messaging into REST-friendly manner. Due to unavoidable facts of wireless network, this architecture describes possible solutions while dealing with intermittent connection loss of mobile consumer. The key points are as follows,

- It becomes challenging to comply with REST features when maintaining consumer’s subscription state information for future notification of resource updates. This contradiction has been addressed by explicitly issuing subscriber’s state management service to the Event Manager. In this way, event publishers can keep themselves free from consumer’s state information.
- A combination of push and push based interaction fine-tuned each other in fault-tolerant system. Since backend server pushes notification to consumer without any knowledge of notification version consumer is currently holding, consumer polling for event updates can be beneficial in keeping himself/herself synchronized with the main Channel resource.
- And finally, it is very important to choose the right pattern of communication for disseminating events knowing the factors involved such as message payload and network strength. A right communication pattern in disseminating events can significantly improve system’s performance in processing cost and network load.

CHAPTER 4

IMPLEMENTATION

This chapter describes how the proposed architecture is deployed from the design perspective of the mobile client, the middleware, the event broker and the persistent data storage. A client-side application is developed and integrated with the server backend. Details of the implementation of each component of the framework are described below;

4.1 Pub/Sub Backend Implementation

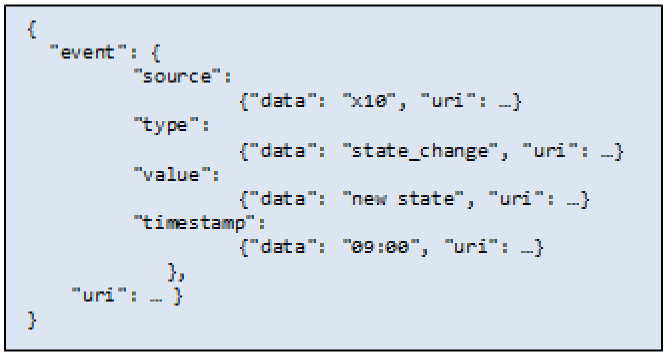
The architecture proposes server backend that is based on Pub/Sub pattern. The backend server nodes consist of a middleware server and a persistent storage server with Pub/Sub brokering system as a front end. The mobile client establishes connection to the middleware server and though it communicates with the persistent database in a RESTful manner.

4.1.1 Middleware Implementation

The middleware component connects mobile applications to the Pub/Sub channels. It is implemented in Erlang/OTP [Larson, J., 2008], a high concurrency oriented functional programming language that supports large number of concurrent actor like activities, called Erlang processes.

The middleware is designed to support RESTful like communication. For every subscribed channel, middleware maintains a temporary data storage that needs to be synchronized with the Channel data storage every time an update has been made at the Channel component. Temporary storage is built as ETS, a temporary storage that can store data in the runtime of Erlang system. This storage is also used for caching purposes and provides a DELETE operation since the messages in the temporary storage needs to be trimmed off after a certain period time.

Communication between the proxy server and mobile application goes through Yaws 1.94 server (an Erlang based http server) that supports both http and WebSocket connection. Since this implementation relies on WebSocket connection, all communications between mobile apps and middleware arrives at the WebSocket listener component that resides just in front of the middleware. All communications take a message-oriented approach where the messages are constructed in a JSON format (Figure 4.1).



```
{
  "event": {
         "source":
                  {"data": "x10", "uri": …}
         "type":
                  {"data": "state_change", "uri": …}
         "value":
                  {"data": "new state", "uri": …}
         "timestamp":
                  {"data": "09:00", "uri": …}
            },
    "uri": … }
}
```

Figure 4.1: JSON Data format

4.1.2 Event Broker and Channel Implementation

Channels are designed to be the persistent data storage for the proposed architecture. Channels are exposed by an Event Brokering (EB) system component. Both Event Broker and Channel are implemented in Erlang. The Channel interface is RESTful compliant and provides three basic functionalities to the Event Broker – Create, GET and POST. Mobile client has access to the only GET and POST method of a Channel component. Messages in Channel data store are never deleted as they are kept for persistency. Addition to the data store, Channel maintains a list of subscribers who has subscribed to the channel. Subscriber list is updated whenever a client has joined to or unsubscribed from the channel.

Channel data storage is built as DETS table, persistent disk storage in Erlang system that store data as objects in a file. When a POST operation is made in the channel, Event Broker is responsible to fetch the subscriber list and the complete data from Channel's data store and broadcast the channel content to all connector processes of the middleware that falls within the subscription list.

Storing data into either persistent data storage (DETS) or temporary storage (ETS) avoids data duplication. Data structure of the stored data in both DETS and ETS table is a tuple that takes an element as its key. When a data is being stored, a lookup is performed into the respective table and the key of the existing tuple is matched with the key of the data to be stored. Every tuple in the table contains a unique key. Any storing attempts made to the storage that has same key of an existing tuple will not be stored. This is to avoid data duplicity. It is essential to avoid duplicate data in order to synchronize data in both Channel and Proxy component and the client side storage and to offer an eventual consistency throughout the system components. The following pseudo code shows the steps in storing data into DETS and ETS.

```
   %% handling data duplication when publishing event and
notifying event subscribers %%

   When received publish(event e) from node x
       If event_id matches existing match{key, value}pair
      {
        discard event
        return "duplicate key error"
      }
       Else
      {
       insert into DETS insert()
       invoke notify()
       fetch Subscriber_list()
       broadcast event to the Subscribers
      }
```

Figure 4.2: Pseudo-code for storing data in ETS and DETS

4.2 Mobile Client Implementation

The client side mobile application framework is designed and implemented on Android 4.0 Ice Cream Sandwich OS [Android, 2012]. The application is running on both Android Web View (device embedded browser) with open source framework that provides supportability in accessing device features (such as PhoneGap) and on desktop browser such as Google Chrome.

The client application is designed in MVC pattern as shown in Figure 4.3 – a UI component that views device stored or server pushed data on the device embedded browser, a Model component that manages device caching and a queue in storing unpublished client requests (while disconnected) and a Controller component that intermediates' between UI and Model.

UI component passes the callbacks to the controller so that controller can reply to the View when there is an update from the backend. A stub element locates inside Model that handles communications between mobile client and backend servers and also responsible to update Model caching and inquiry Model queue every time client establishes a connection to the backend. Client side components are developed using the latest web technology such as HTML5, JavaScript and CSS. In order to improve UI layout and facilitate event mechanism, web technology framework such as jQueryMobile (v 1.2) is used.

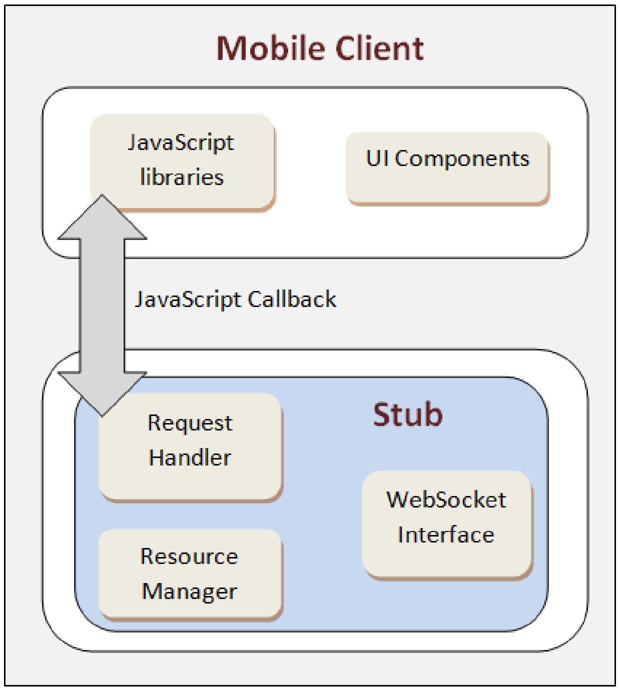


Figure 4.3: Mobile application framework

4.2.1 Client-side Storage

In implementing client side storage like the model queue and model caching, the browser embedded Web SQL database (relational database) is used. Web SQL database features of device embedded browser are obtained using PhoneGap Library [PhoneGap, 2012], an open source framework that leverages the latest web technologies of HTML, CSS and JavaScript and provides access to device embedded features.

4.2.2 Client-side Communication Interface

A mobile client sends asynchronous requests to the server and the responses from server are pushed back to client application. Over the years, several web technologies that have been developed to send asynchronous requests to the web browser are namely Ajax and Pushlet. Some of the recent technologies includes WebSocket, server-side-event, XMPP and Bayeux. The client-server interaction in this implementation exploits WebSocket connection. The client-side API for WebSocket provides four functionalities as follows.

```
                 //creates a WebSocket instance
        var myWebSocket = new WebSocket (url, [protocol]);

                      myWebSocket.onOpen(){
          //establishes WebSocket connection with server
                                }

                    myWebSocket.onMessage(){
           //receive all incoming messages from server
                                }
                     myWebSocket.onClose(){
          //closes connection between client and server
                                }
                     myWebSocket.onError(){
    //invokes when there is an error occurred in the
                                   connection
                                }
                                }
```

When client wish to send a message to the server, it simply calls `send()` function.

```
myWebSocket.send(){
          //deliver a message to the backend
}
```

4.3 Summary

This chapter describes the technologies and the techniques used in the implementation of the proposed system. The implementation is divided into two parts – mobile client framework implementation and backend server implementation. Section 4.1 describes three layers of mobile client framework with a broader emphasis on the implementation of model component which includes stub in managing client-side caches and communication with server backend. The backend server implementation is described in section 4.2

CHAPTER 5

EXPERIMENTS

In this chapter, the proposed system is evaluated in accordance with the research challenges stated in Chapter 1, aiming to study the system performance under different scenarios. The experiments analysis and evaluation serve to demonstrate the framework's feasibility in various event dissemination patterns and also to identify the best performing scenario.

5.1 List of Experiments

Table 5.1 summarizes the proposed experiments that relate to the research challenges.

Table 5.1: Lists of proposed experiments

Experiments	Experiment Goals
Update Propagation Test	To observe the perceived delay in propagating event updates with different size of message payloads
Client App Performance Test	To test the client application portability and performance in the JavaScript environment on the mobile client and desktop browser, as well as Erlang desktop client
System Overhead Test	To observe system overhead in propagating events over different communication protocols
Synchronization Test	To observe perceived delay in synchronizing event updates from backend persistent channel as well as device layer/connector.
Bandwidth Consumption Test	What is the throughput of sending data over different communication protocol

5.2 Experiment Setup

The three major components in this experiment setup include mobile users (event producer and consumer), Pub/Sub Proxy layers (Protocol Layer, Device Layer and Event Manager), Pub/Sub Persistent Event Channels as shown in Figure 5.1.

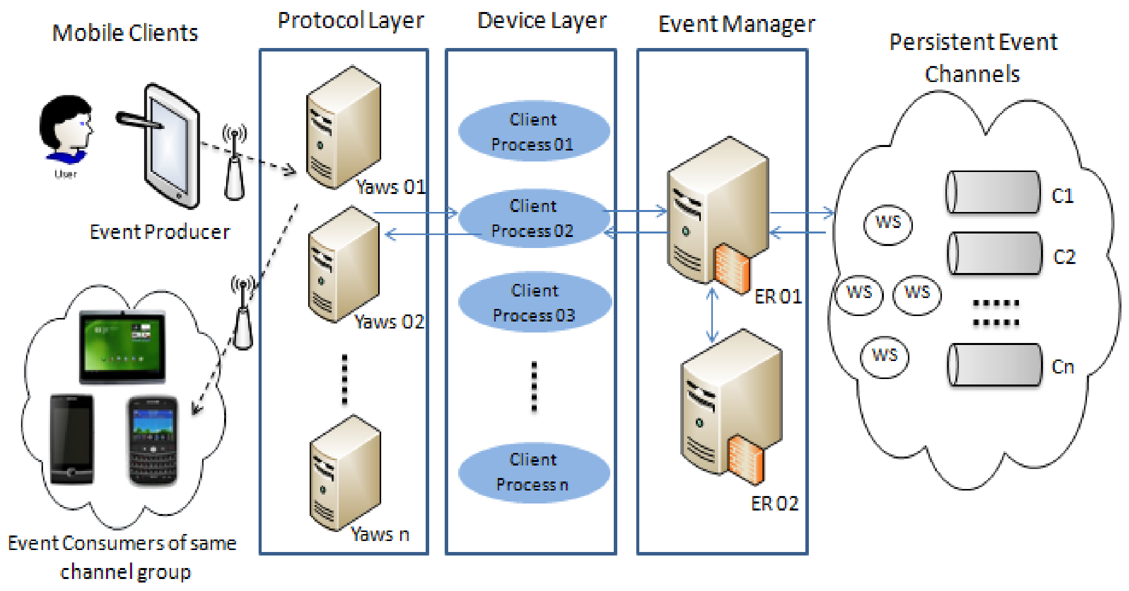


Figure 5.1: Overall scenario of the system

- Mobile client: Mobile clients are running on ASUS Transformer Prime tablet. The device specifications are shown in Table 5.2

Table 5.2: Hardware specifications of the mobile device

Hardware	Specification
System	Android™ 4.0 Ice Cream Sandwich OS
Processor	NVDIA® Tegra® 3 Quad-core CPU
Memory	1 GB
CPU Speed	1.3 GHz

- Pub/Sub proxies: A Windows 7 desktop machine is used to host Pub/Sub proxy layers. Table 5.3 summarized the hardware specifications.

Table 5.3: Hardware specifications of Pub/Sub proxy layers

Hardware	Specification
System	64-bit Windows 7 Professional
Processor	Intel® Core ™ i5-2400 CPU
Memory	16.0 GB
CPU Speed	3.10 GHz

- Pub/Sub Persistent Event Channels: A Windows 8 desktop machine is used to host Pub/Sub event channels. The hardware specification is shown in Table 5.4.

Table 5.4: Hardware specifications of Pub/Sub Persistent Event Channels

Hardware	Specification
System	64-bit Windows 8 Enterprise
Processor	Intel® Core ™ i5 CPU
Memory	4.0 GB
CPU Speed	3.20 GHz

5.3 Experiment 1- Update Propagation Test

This experiment calculates the time it takes in propagating a resource update message in the form of a notification within the proposed architecture. The resource consumption time (i.e.

accessibility) includes the time difference between an event gets published; and is received by a mobile consumer. Since the higher level of REST (level 3 as mentioned in chapter 3) includes Hypermedia in the response, the response message generated is larger comparing to the lower levels of REST i.e. a large message payload needs to be propagated when the higher level of REST (level 3) is followed. Therefore, the experimental parameters chosen for this experiment are summarized in Table 5.5.

Table 5.5: Experiment parameters for Update Propagation test

Dissemination Pattern	With and without event message
Event message payload	5kb
	10 kb
	50 kb
Update Notification payload	2 kb

5.3.1 Experiment Scenario

The time spent on propagating a resource update is the Round Trip Time (RTT) calculated at the publisher's end upon receiving the published resource. Event notification and the event message are sent to the subscribers based on server-side push as shown in Figure 5.2. In the first scenario, event message of different message payloads are published to the Pub/Sub persistent channel. Upon receiving the published events, Event Broker generates an event notification of 2 kb and pushes an accumulation of event message and the update notification to the mobile consumers. In second scenario, Event Broker pushes only the update notification.

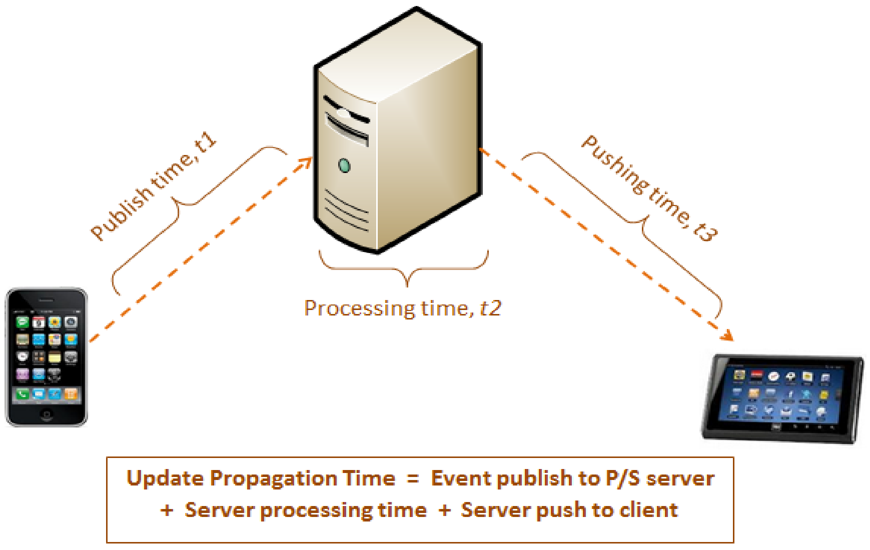

Publish time, t1
Processing time, t2
Pushing time, t3
Update Propagation Time = Event publish to P/S server + Server processing time + Server push to client

Figure 5.2: Time Delay in Resource Update Propagation

5.3.2 Result and discussion

The experimental results are shown in Figure 5.3 and 5.4. The result in Figure 5.3 shows the time it takes to propagate 5 kb, 10 kb and 50 kb of event messages from mobile publishers to the Event Router and then a summation of event message and update notification from the Event Router to the mobile consumer. The result shows an increase in the propagation time as the message payload increases. A similar increase in message payload to time ratio is experienced in Figure 5.4. However, the propagation time is much faster in Figure 5.4 compared to Figure 5.3 since the latter scenario does not include event message.

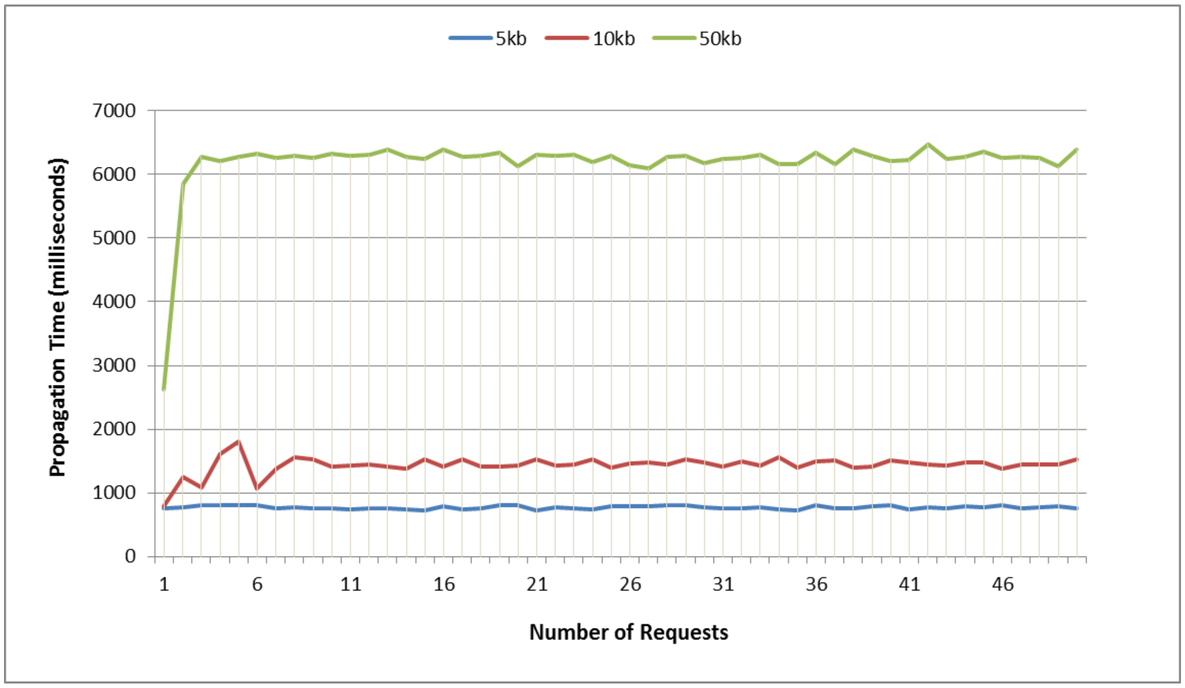


Figure 5.3: Propagation time (with event messages)

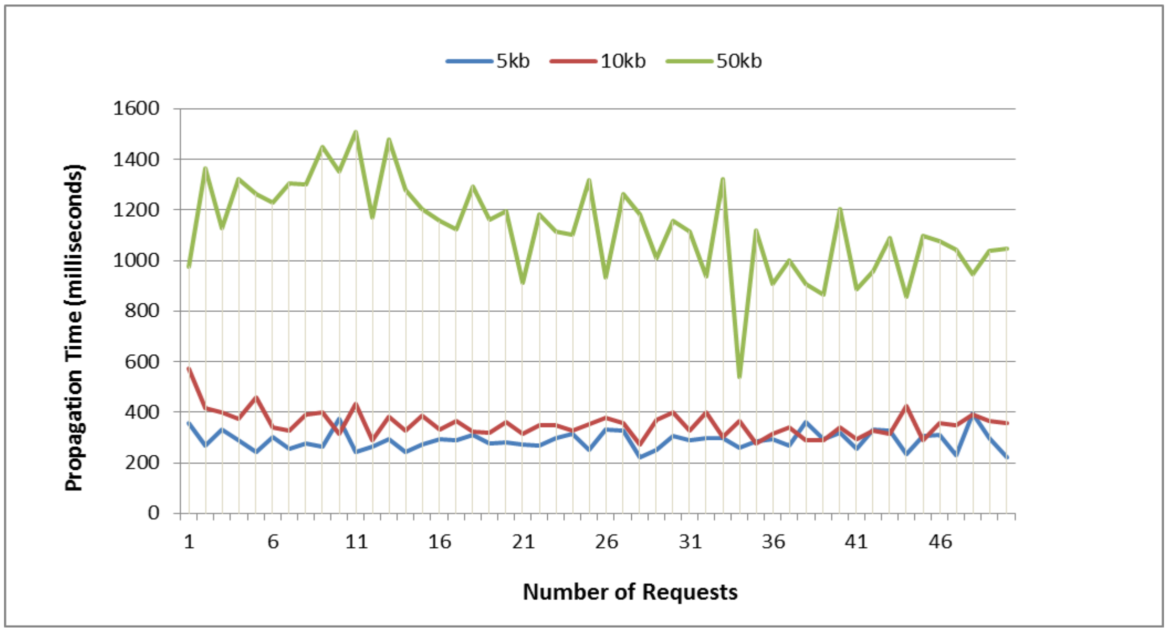


Figure 5.4: Propagation time (without event messages)

The result in table 5.6 shows the average, maximum and standard deviation time of update propagation of both scenarios. From table 5.6, it can be inferred that propagation time for 10 kb of payload is 4.06 times faster without the event messages being pushed to the consumer than only notification is pushed. It can be inferred that in a scenario where the published message is larger, broadcasting only the update notification can be a faster choice.

Table 5.6: Result of update propagation test

Update propagation pattern	Message Payload	Time in milliseconds		
		Average	Maximum	Standard Deviation
With Event Message	5kb	769.1	811.2	24.81
	10kb	1433.3	1799.8	143.4
	50kb	6201.2	6478.4	499.9
Without Event Message	5kb	288.5	396.80	38.2
	10kb	353.4	574.6	52.4
	50kb	1127.4	1509.4	185.6

Furthermore, an extrapolation on Figure 5.4 shows that initially the update propagation time for 50 kb of message payload is much longer. After the first 30 samples, the update propagation time was observed to maintain an average time of 1115.6 milliseconds throughout the experiment. Since every update propagation requires an equal amount of computation starting during the event publish and update receive at the mobile consumer's end, experiencing a longer propagation time can be attributed to the network instability.

5.4 Experiment 2 – Client App Performance Test

The purpose of this experiment is to observe the system's performance in request/response on different client application platforms. In this experiment, three different application platforms that have been tested are Erlang client, JavaScript Desktop browser and device embedded browser. Each of this platform establishes WebSocket connection to its backend system.

5.4.1 Experiment Scenario

In this experiment, 5 kb of event messages has been published from the initial sender to the Persistent Channel and 1 kb of event messages has been pushed to mobile clients by Event Router. As the event message propagates from sender to the receiver, the Round-Trip-Time (RTT) has been observed.

5.4.2 Result and discussion

Among the three client applications, the best performance is observed on the Chrome browser running on Desktop. The result in Figure 5.5 shows that the average RTT on Android browser is 212.8ms when it is 119.4ms on Erlang client (1.8 times faster than on Android) and 61.3ms on Chrome browser (3.5 times faster than on Android). The average, maximum and standard deviation of RTT on chosen client platforms are shown in table (Table 5.7).

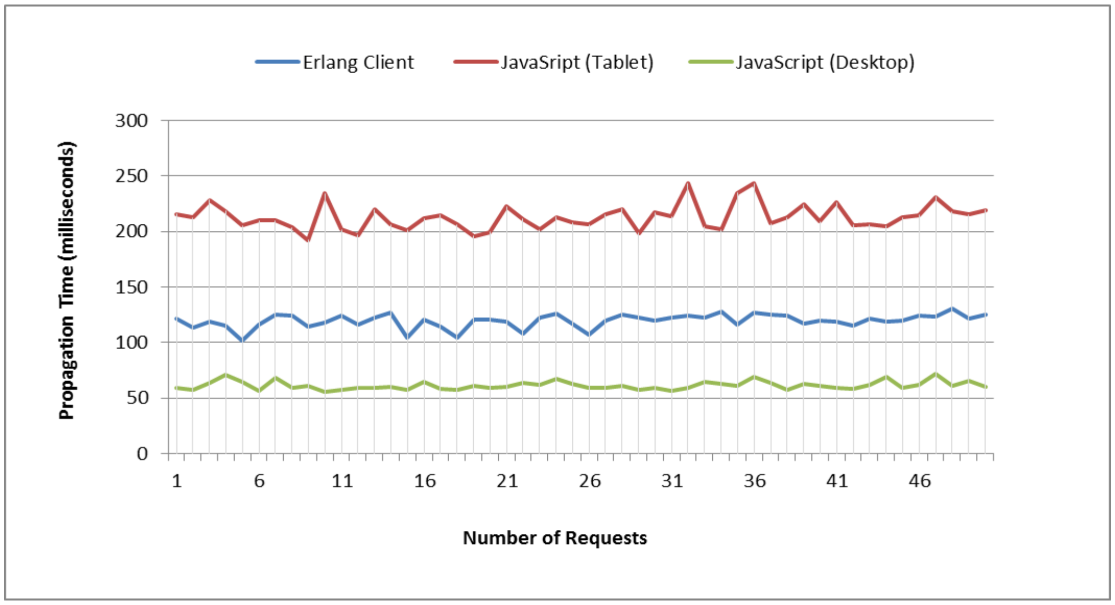


Figure 5.5: RTT per request (multiple client platforms)

Table 5.7: Result of client application platform performance test

Client Platform	Time in milliseconds		
	Average	Maximum	Standard Deviation
Erlang Desktop	119.4	130.2	6.2
Javascript on Chrome Browser	61.3	72	3.8
Javascript on Android	212.8	243.4	11.5

One possible reason that the app on Android WebView performs slower than Chrome browser is because WebView is linked to the Android application layer written in Java. For every activity in WebView for example JIT (just-in-time) compilation of JavaScript, the callback function is invoked. Moreover, the integration of an external framework in the application such as PhoneGap might have added an additional execution time which in turn causes performance deterioration.

5.5 Experiment 3 - System Overhead Test (Protocol Overhead)

This test is conducted to observe the amount of overhead the chosen dissemination approaches introduces on the system in terms of latency in consuming a resource from the Persistent Channel. The chosen approaches include client pull over HTTP Ajax and server push over WebSocket. The purpose of this test is to observe the time difference and identify which approach performs better in event dissemination.

5.5.1 Experiment Scenario

In this experiment, the event update message is stored in the persistent channel. The experiment is conducted in two scenarios. In the first scenario, mobile consumer who are subscribing to a channel are configured to pull for event updates from the channel every 2 seconds. In the second scenario, as event updates arrives at Persistent Channel, Event Router pushes the update to the subscriber's end i.e. update propagation does not require any requests arriving from the subscribers. Both of these scenarios have been shown in the Figure 5.6.

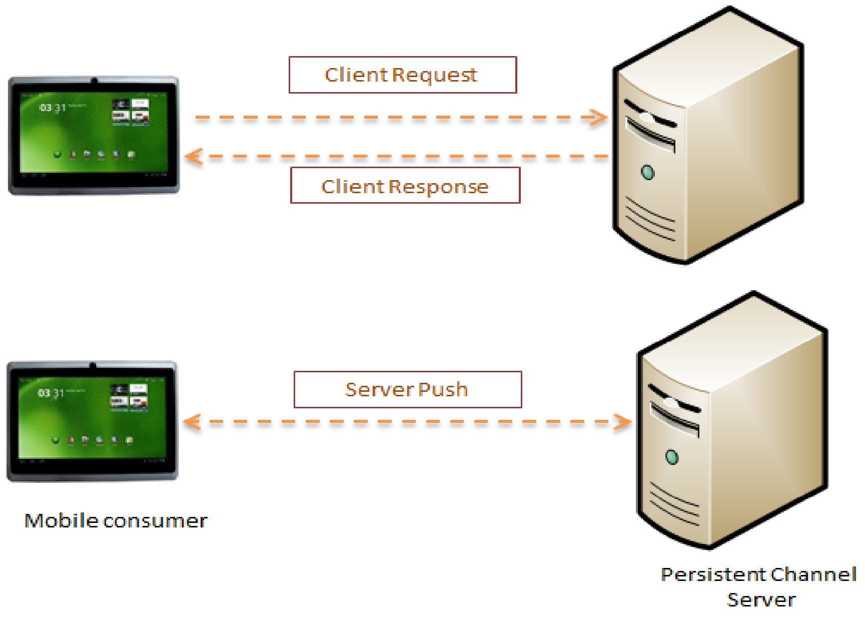


Figure 5.6: Client pull (synchronous and asynchronous) and server push

5.5.2 Result and discussion

The result of client pull and server push is shown in the Figure 5.7. The graph shows the time for individual update propagation (50 samples) obtained from an average of five iterations where the size of each event message is 10 kb. From the graph, it can be observed that, time consumption in first scenario where the message propagates from event publisher to the server and having server send update to the subscriber as a response for update request takes much longer time comparing to the time of propagating event from publisher to the server and having server push the update to the subscriber of the channel. Time in event consumption is observed almost 1.5 times faster in server push scenario compared to client pull.

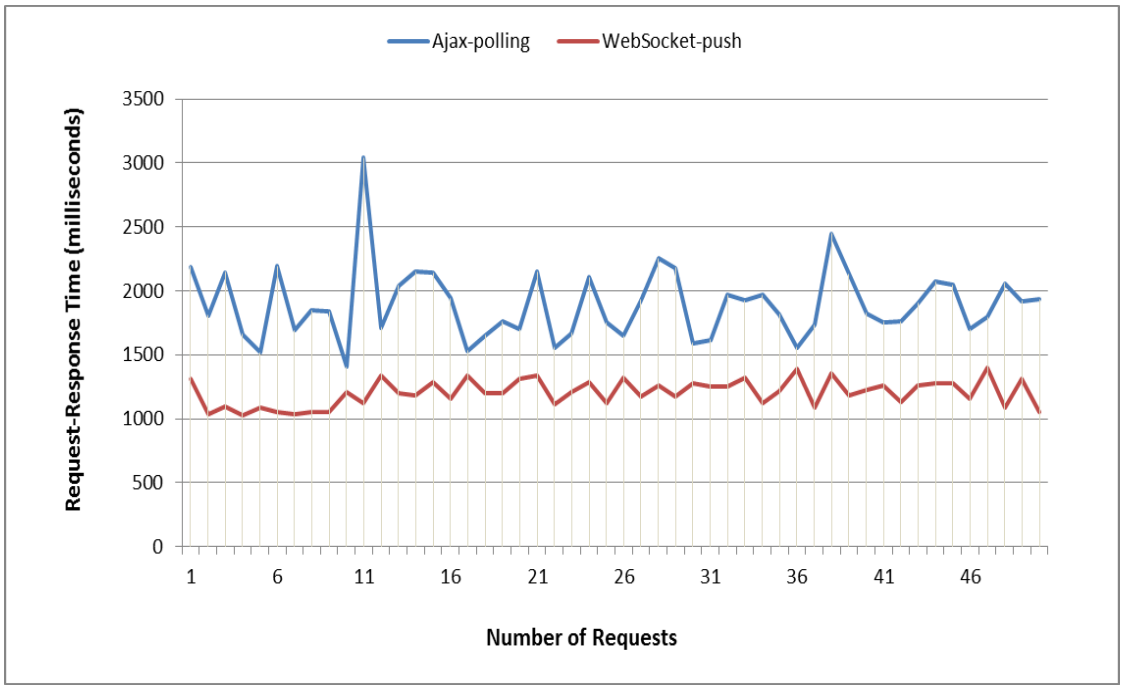


Figure 5.7: Response time per request over http polling and WebSocket

The average, maximum and standard deviation time (in milliseconds) for disseminating event messages from publisher to the consumer over two dissemination approaches have been shown in the table (Table 5.8) below.

Table 5.8: Result of system overhead test

Dissemination Pattern	Time in milliseconds		
	Average	Maximum	Standard Deviation
Ajax-polling	1894	3042.2	282.3328
Server Push	1204	1396.8	104.6415

A possible reason that Ajax-polling takes longer time than the server push is that client pulling interval is set to every 2 seconds. Any update that arrives right after the client pull, will take almost 2 seconds for client to receive the updates. However if the update arrives closer to the end of 2 seconds pulling interval then the propagation time difference between client pull and server push are very close except the fact that message overhead is higher in Ajax pull (around 634 bytes) compared to WebSocket header (around 6 bytes) which adds an additional time latency in event propagation.

5.6 Experiment 4 – Resource State Synchronization Test

A framework that is designed to run part over heterogeneous network for example in this case, part over wireless network and part over LAN, one problem that arises in accessing resources from a far node is the routing overhead. In the proposed framework of this research, a client process is maintained for each individual subscriber at the device layer where the resources are stored temporarily. If the client process is not maintained at the device layer then the alternative approach in synchronizing client side resource would be sending request for updates at the Persistent Channel which is multiple hops away from the clients. Therefore consumer's resource state can be synchronized from two different locations – Connector process of the device layer and the Persistent Event Channels. Hence, the purpose of this experiment is to observe system's performance difference in maintaining and not maintaining a client process at the device layer.

5.6.1 Experiment Scenario

In conducting the experiment, a resource has been published at the Persistent Channel. In first scenario, a client process with a temporary storage is maintained, hence the published resource has been pushed to the Connector by Event Router and client resource is synchronized with the backend resource at the device layer as shown in Figure 5.8. In the second scenario, published resource is made available to only Persistent Channel. Hence client application is configures to synchronize its local resource at the Persistent Channel.

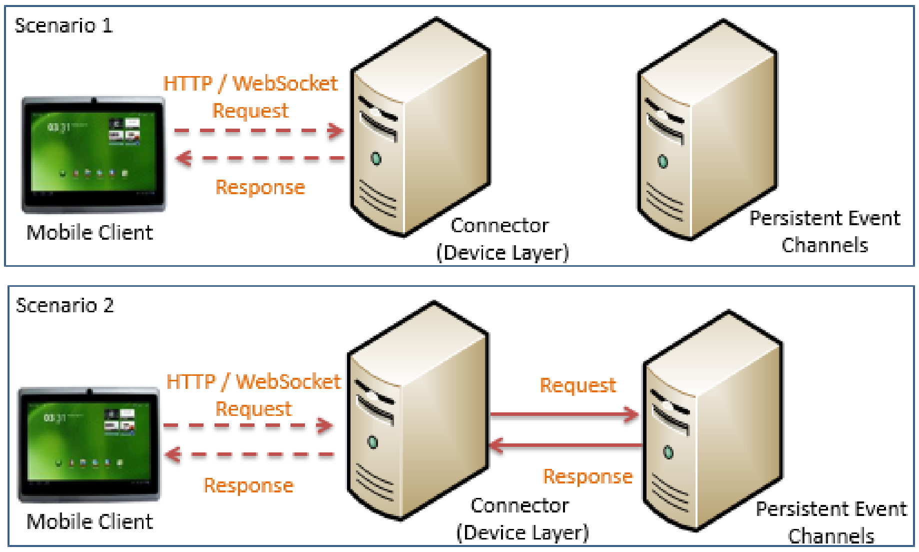


Figure 5.8: Synchronizing client resource state from Connector (device layer) and Persistent Event Channels

5.6.2 Result and discussion

The results from the experiment is graphically presented in Figure 5.9. The graph shows the synchronization time for 50 individual requests. Each synchronization time plotted on the graph is an average time of five iterations. A resource of size 5kb has been synchronized between client's local storage and the backend storage based on client's current resource id. Results shows that the average time required to synchronize the resource from device layer is 228.5 milliseconds while it is 588 milliseconds if synchronized from the Persistent Channel Layer which is 2.6 times (157.3 %) slower. Hence, maintaining a client process in a closer proximity of the client device can result in a better performance in synchronizing data in a distributed framework.

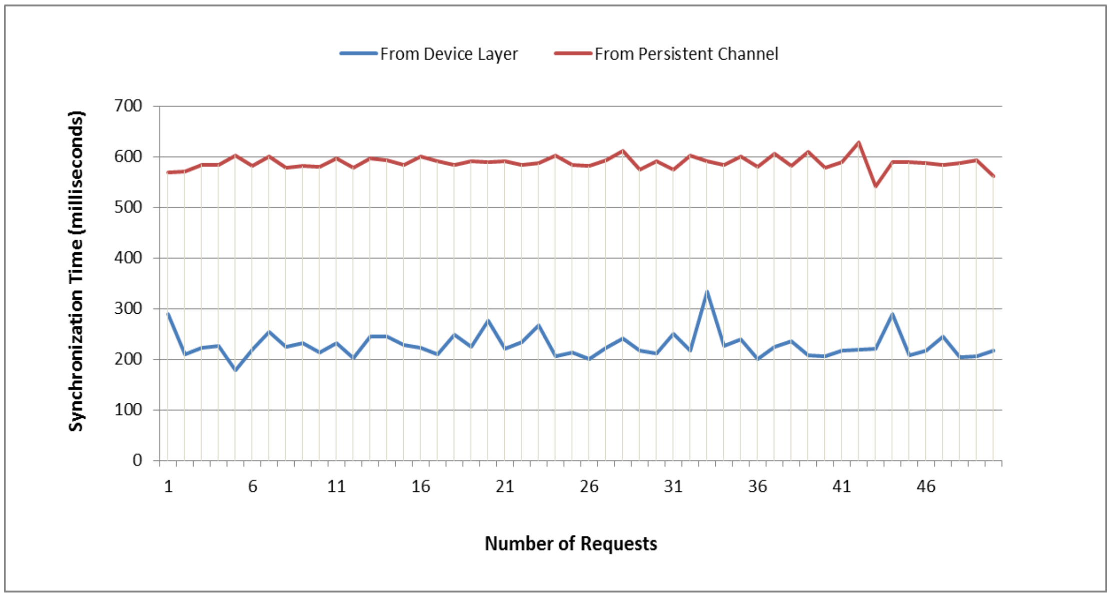
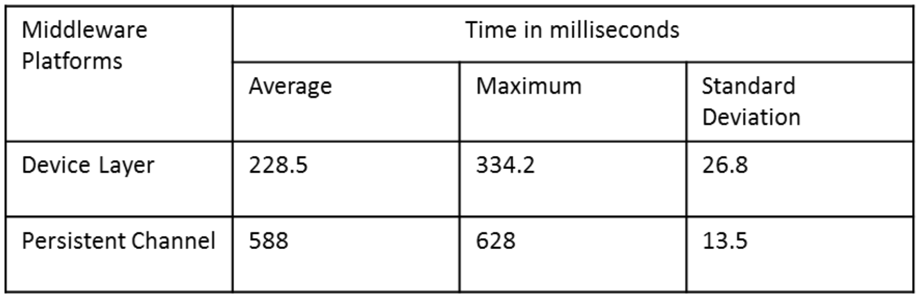


Figure 5.9: Response time per request from the device layer and from Persistent Channel

Table 5.9 shows the average, maximum and standard deviation time (in milliseconds) for synchronizing event messages of 5kb payload from device layer as well as Persistent Channel layer.

Table 5.9: Result of State Synchronization test



Middleware Platforms	Time in milliseconds		
	Average	Maximum	Standard Deviation
Device Layer	228.5	334.2	26.8
Persistent Channel	588	628	13.5

5.7 Experiment 5 - Bandwidth Consumption Test

This experiment analyzes the bandwidth consumption over wireless network in disseminating resource updates to the corresponding clients. The purpose of this experiment is to compare the throughput of update dissemination over traditional client pull approach with the server push based data dissemination in Pub/Sub paradigm. The experiment investigates the technique that helps in efficiently consuming available bandwidth by avoiding unnecessary network traffic in communication network. As the updates are propagated from Pub/Sub server to clients, bandwidth is calculated at server’s end for every incoming and outgoing interaction.

5.7.1 Experiment Scenario

In this experiment, a similar scenario of System Overhead test (Experiment 2) has been adopted (Figure 5.6). This experiment is conducted in two phases. In first phase, client app is configured to send resource update request at a constant rate (i.e. every 2 seconds). Upon receiving the client request, Pub/Sub server responds with an update notification of 2kb of message payload and the updated resource. In case there is no update available, sever acknowledge the requester with a message “No update is available”. In second phase, Pub/Sub server pushes the updated resource to the subscriber without subscriber prompting for the update.

5.7.2 Result and discussion

Figure 5.10 shows the throughput in kilobyte/second for individual resource propagation in client pull and server push approach of event dissemination. In this experiment, 10kb of data has been transferred between mobile client and server. The average throughput obtained over http polling is 5.8 kb/s when the average throughput over WebSocket is 8.6 kb/s. Bandwidth consumption over WebSocket results in at least 1.5 times higher compared to http polling.

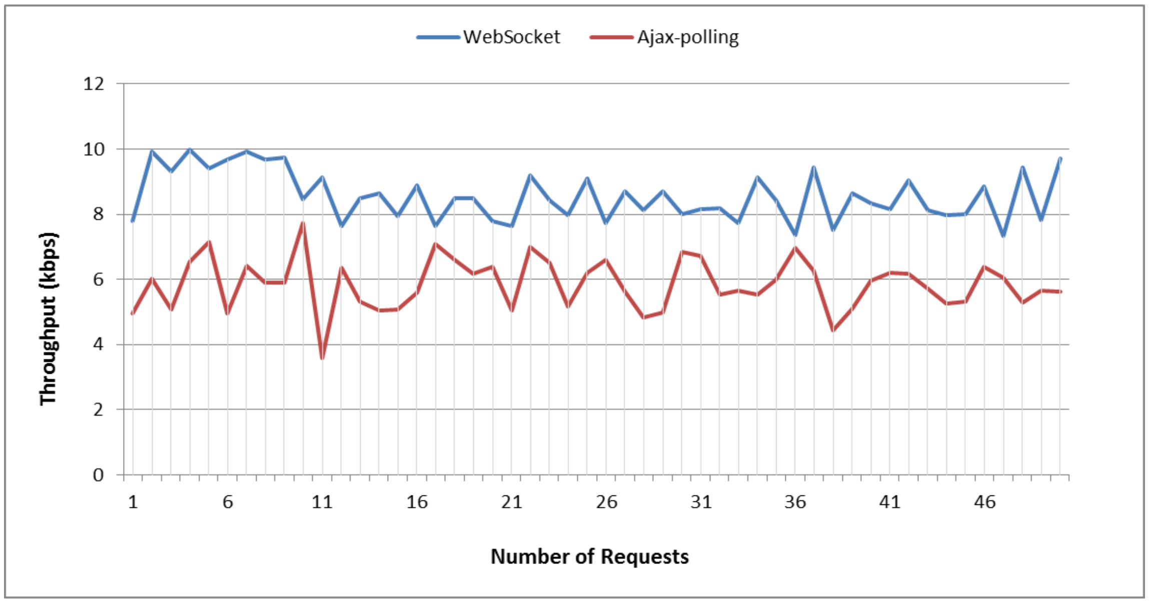


Figure 5.10: Throughput per request over http polling and WebSocket

Table 5.10 shows the average, maximum and standard deviation (kb/s) throughput over http polling and WebSocket.

Table 5.10: Result of bandwidth consumption test

Dissemination Pattern	Throughput (kb/s)		
	Average	Maximum	Standard Deviation
Ajax-polling	5.8	7.7	0.79
Server Push	8.6	9.9	0.75

The reason server push consumes bandwidth more efficiently is because WebSocket has a smaller overhead (around 6 bytes) compared to http-polling (around 634 bytes) and therefore data propagation time is comparatively lesser in WebSocket push.

5.8 Summary

The proposed experimental design is conducted to evaluate system's performance based on the perceived network latency while consuming web services on mobile device over the wireless network, system overhead introduced due to the adopted communication channel protocols at the application layer and also the bandwidth consumption in terms of throughput (kb/s) over wireless communication network.

From the results on update propagation it has been observed that as the message payload increases, the message dissemination time also increases and in continuation to that disseminating an increased payload of event messages delays the delivery of update notification to the corresponding subscribers if the update notification includes the event message itself (Figure 5.3 and 5.4). Therefore, if relying on the upper level of RESTful Web Services (level 3) in an event-based Pub/Sub system, a suitable approach for disseminating event in mobile environment would be sending only the update notifications to the mobile clients and delivering the updated resource upon client requests.

The time for client side resource synchronization with the backend resource can be reduced if a temporary storage of the resource is maintained for each individual client in a location that is closer to the client. The conducted experiment on client app resource state synchronization demonstrates 2.6 times faster synchronization time with Connector at the Device layer compared to the Persistent Channel at far backend server (Figure 5.8). The distance of resource location as well as the additional computation of proxy layers can add extra latency in accessing the resource.

The system’s performance in event dissemination based server push and traditional client pull scenario shows a dramatic performance difference in the proposed framework. In a server push scenario, update message is sent to the server and server pushes the update to the subscriber. In client pull server sends request to the subscriber upon receiving subscriber request. Experimental results show a dramatic performance improvement (almost 1.5 times faster) in WebSocket push-based event dissemination over the traditional client pull approach. Moreover, transferring data over WebSocket channel results into higher throughput (kb/s). Result shows 1.5 times greater throughput of data transfer on WebSocket connection compared to Ajax http polling connection.

In conclusion, the experiments have shown the potential of the proposed framework in successfully disseminating events and help demonstrate framework’s feasibility within mobile environment due to its adoption of push-based event dissemination using the lightweight and scalable RESTful Web Services.

CHAPTER 6

CONCLUSION

6.1 Summary

The proliferation of mobile devices is bringing a dramatic change in mobile digital ecosystem and resulting into a distributed and heterogeneous system that includes several platforms, computer languages and different IT technologies. As a matter of fact, integrating system applications in overly distributed system has become challenging and a major concern for today's enterprise service providers of information system. Moreover, mobile devices use wireless channel as a standard access media in receiving services which involves the challenges of propagating data over unreliable network such as network latency, limited bandwidth and intermittent connectivity and hinders data propagation in close to real-time and synchronizing them across the framework. To overcome these challenges, in this research we proposed a hybrid of REST-based and Pub/Sub event based framework to provide reliable event dissemination in mobile environment [Kazi, R; Deters, R., 2013c].

This thesis has begun with the background information and motivation behind the research work, problem statement and research goals that the work expects to achieve from this study. With respect to the problem statement and research goals, this research looks into different architecture models such as topic-based Pub/Sub model as one of the current enterprise application integration technique and also system interaction style. Research also explores one of the Web Service techniques namely RESTful Web Services as a promising technology to reach interoperation in heterogeneous environment. Different types of data dissemination techniques has also been studied in this research such as traditional client-pull approach over Ajax http connection and the server-push approach of event dissemination over WebSocket in a distributed system. Apart from the backend framework design, research also focuses client application design and explored some of the standard design frameworks such as MVC and MVP in making application components more loosely coupled. Based on the reviewed literature, the research proposes an architecture model that is suitable to operate in mobile environment. Our proposed architecture shows possible integration between RESTful Web Service and Pub/Sub model and defines the interaction protocol. Nevertheless, proposed architecture acknowledges intermittent connectivity issues in its framework design.

A prototype of the architecture has been implemented in this research. The backend architecture is built in Erlang, a concurrency oriented programming language that ensures server scalability and reliability in providing services to a large number of mobile clients. The client-side application is developed based on MVP architecture pattern using JavaScript and some external JavaScript libraries. The backend system components rely on a message-based communication style and the event dissemination approach between backend server and mobile subscribers relies on server-push approach contrary to the traditional client-pull approach.

The proposed framework design is evaluated through conducting experiments on network latency in propagating and synchronizing events and bandwidth consumption to observe system`s performance.  Experiment results demonstrates system`s improvement in push-based event dissemination over the traditional client-pull event dissemination.

In conclusion, this research proposed a RESTful Pub/Sub framework for integrating distributed system components in mobile space and efficiently disseminating data over wireless network. The proposed framework is designed to achieve faster and reliable data dissemination.

6.2 Research Contributions

The research contributes in the domain of Web Services based event dissemination in Pub/Sub domain as follows;

- Analyzes different patterns of RESTful Web services within Pub/Sub domain for disseminating consumer data, hence provide interaction protocol.
- Studies the latest Web communication technologies and different data dissemination patterns to address the challenges of network latency in mobile environment.
- The use of Web frameworks such as jQuery, jQuery Mobile and PhoneGap enhance the deployment of cross platform mobile application.
- Proposes a solution for traditional pull-based architecture by adopting WebScoket as a communication protocol.
- Provides a platform for Pub/Sub communication on mobile environments.

6.3 Limitations and Future Studies

The proposed framework suffers from following limitations;

- The backend implementation of the proposed framework is Erlang platform specific which does not support tools that are written in other programming language. Developers are bound to write platform specific actions and requires to have an extensive knowledge on the language platform. Hence application development is expensive.
- The Proposed framework uses third-party API for WebSocket communication protocol. A self-developed WebSocket connection would provide developers a greater control in event dissemination such as configuring the buffer size of the communication channel.

- Moreover, the current research does not adopt large scale deployment on real devices. The current deployment only includes two tablet devices simulating as both mobile publisher and mobile subscriber. Hence, systems performance in terms of scalability on a regular wireless environment with large user group is unknown. It would be great to deploy the framework on a large scale and assess the impact on the proposed service.

This research will like to explore the following features as future studies of this research that could be added to the existing framework to achieve greater performance improvement.

- **Decentralized Pub/Sub system.** The current Pub/Sub framework is based on centralized event brokering system that relies on a single event broker. The centralized event broker keeps record of all active subscriptions in the system. When an event is published, event broker invokes its notification method and delivers the update notification to the subscription user`s list that it currently holds. If the event broker is down then the event dissemination within the framework will be compromised hence relying on a single event broker increases the vulnerability of the entire system because it limits the system by the capacity of a single server. Hence adopting decentralized Pub/Sub model [Huang, Y., Molina, G., 2001] is a promising line of work. In decentralized approach, the system consists of M number of event brokers each responsible for a portion of N total subscription and hence responsible to deliver event updates to its own active subscription user`s list. Besides the decentralized approach, peer-to-peer (P2P) support [P. Triantafillou and I. Aekaterinidis, 2004] can help building a large-scale distributed system where every connected device can act as client and/or server and form a completely decentralized, self-organizing and scalable system.

- **Maintaining a User Profile.** The proposed framework is based on topic-based subscription scheme where users subscribe to events of a channel based on the channel topic or subject. However, subscription mechanism can be improved by introducing a subscription scheme based on the actual content of an event which provides more granularity in event subscription through offering a fine filtering mechanism on events. In this mechanism, maintaining a user profile can be useful in defining filtering rules in event subscription [I. Podnar et al., 2002]. Nevertheless, the proposed framework uses a flexible queuing policy where the notifications are buffered until the subscriber reconnects. A more complex and granular queuing policy would buffer undelivered notifications based on the subscriber defined propertied such as priorities and expiry dates of event channels.
- **N-Screen Application Framework.** Supporting N-screen application in Pub/Sub framework is another future direction of this research that can be looked into to improve our proposed framework. In Pub/Sub system, subscriber may use multiple devices and subscribed to an event channel from each of his/her device. In this scenario, resources are shared among multiple devices with separated screens [Zhang, 2012] i.e. visibility of a subscribed event resource may have device preferences based on the user profile. This approach of using N-screen application provides more flexibility in integrating user`s device with Pub/Sub system. However, dealing with N-screen subscriber application requires consistent user experiences across multiple devices irrespective of device platforms and hence require efficient resource state synchronization technique.
- **Mobile Web Service Provisioning.** One of the major trends of distributed system network and also a future direction of this research is the emergence of mobile terminals

as Web Service providers also known as Mobile Hosts [Srirama, S. et al., 2006]. When lot of research focuses on provisioning Web Services from resource constraint mobile device, some research works sees the potential of using smart and more powerful mobile devices with sufficient speed as the service delivery node in a peer-to-peer settings. By using light weight Web Services such as RESTful, web services can be easily deployed on these devices [Lomotey and Deters, 2012]. This approach provides greater integration and interoperability among mobile devices.

REFERENCES

ABI Research 2011. 2.1 Billion HTML5 Browsers on Mobile Devices by 2016 says ABI Research. Oyster Bay, New York.

Available: http://www.abiresearch.com/press/21-billion-html5-browsers-on-mobile-devices-by-2011. Retrieved January 11, 2012.

Anceaume, E., Datta, A.K., Gradinariu, M., Simon, G. 2002. Publish/subscribe scheme for mobile networks, in: Proceedings of the ACM Workshop on Principles of Mobile Computing 2002, pp. 74–81.

Armbrust, M. et al. 2009. Above the clouds: A Berkeley view of cloud computing, Dept. Electrical Engineering and Computer Sciences, University of California, Berkeley, Tech. Rep. UCB/EECS, vol.22(6), pp. 931-945.

Ashik, K., Kazi, R., Deters, D., 2012, "Supporting the Personal Cloud", IEEE Asia Pacific Cloud Computing Congress 2012, Shenzhen, China, November 14-17, 2012.

Android, 2012. Introducing Android 4.0. Available: http://www.android.com/about/ice-cream-sandwich/ Retrieved on March 15th, 2012.

Baldoni, R. and Virgillito, A. 2005. Distributed event routing in publish/subscribe communication systems: a survey. Technical Report TR-1/06. The Computer Journal, vol.50(2), pp.444 -459.

Cugola, G., Nitto, E., Fuggetta, A. 2001. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. IEEE Transactions on Software Engineering, 27(9):827-850.

Cugola, G.,Jacobsen, H. 2002. Using publish/subscribe middleware for mobile systems.Mobile Computing and Communications Review 6(4): 25-33.

Cilia, M., Fiege, L., Haul , C., Zeidler , A., Bunchmann , A. 2003. Looking into the past: enhancing mobile publish/subscribe middleware, In *Proc. of the 2nd intl. Workshop on Distributed Event-based Systems*, 2003.

Caporuscio, M., Carzaniga, A.,Wolf , A. 2003. Design and Evaluation of a Support Service for Mobile, Wireless Publish/Subscribe Applications.IEEE Trans. Software Engineering 29(12): 1059-1071.

Cassetti, O., Luz, S. 2011. The WebSocket API as supporting technology for distributed and agent-driven data mining. Available: http://www.scss.tcd.ie/~casseto/NGDM11-websockets.pdf. Retrieved February, 15, 2012.

Chun, B. G., Maniatis, P. 2009. Augmented Smartphone Applications Through Clone Cloud Execution, in Proceedings of the 12th Workshop on Hot Topics in Operating Systems (HotOS XII), May 2009.

Cuervo, E. et al. 2010. MAUI: Making Smartphones Last Longer with Code Offload, in Proceedings of the 8th international conference on Mobile systems, applications, and services (ACM MobiSys '10). San Francisco, CA, USA: ACM, 2010, pp. 49–62.

Cao, Y., Jarke, M., Klamma, R., Mendoza, O., Srirama, S. 2009. Mobile Access to MPEG-7 Based Multimedia Services, in 2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware. Taipei, Taiwan: IEEE, 2009, pp. 102–111.

Christensen, J.H. 2009. Using RESTful web-services and cloud computing to create next generation mobile applications, Proceeding of the 24th conference on Object oriented programming systems languages and applications - OOPSLA '09, New York, New York, USA: ACM Press, p. 627-634.

Dionysios, G. 2008, HTML5 Web Sockets vs. Comet and Ajax. Available: http://www.infoq.com/news/2008/12/websockets-vs-comet-ajax. Retrieved December 17th 2011.

Emmatty, J.T. 2011. Differences between MVC and MVP for Beginners. Available: http://www.codeproject.com/Articles/288928/Differences-between-MVC-and-MVP-for-Beginners Retrieved on June, 01 2013.

Eugster, P., Pascal, A., Guerraoui, R., Kermarrec, A. 2003. The many faces of publisb/subscribe, ACM Computing Surveys (CSUR), v.35 n.2, p.114-131.

Fiege, L., Muhl, G. 2000. Rebeca Event-Based Electronic Commerce Architecture, Available: http://www.gkec.informatik.tu-darmstadt.de/rebeca. Retrieved March 20th 2013.

Fiege, L.,Gartner , F.C.,Kasten , O., Zeidler , A. 2003. Supporting Mobility in Content-Based Publish/Subscribe Middleware, p.103-122.

Feldman, D. 2011. Adventures in HTML5: Mobile Webkit performance optimization.

Available: http://operationproject.com/2011/adventures-in-html5-mobile-webkit-performance-optimization/#.T4INMvtSTeF. Retrieved December 17th 2011.

Furukawa, Y. 2011.Web-based Control Application Using Websocket, ICALEPCS2011, p.673-675.

Fowler, M. 2006. GUI Architecture. Available: http://martinfowler.com/eaaDev/uiArchs.html Retrieved February 25th 2012.

Fielding, R. T. 2000. Architectural styles and the design of network-based software architectures. PhD Dissertation. Dept. of Information and Computer Science, University of California, Irvine. 2000.

Foster, I. et al. 2008. “Cloud Computing and Grid Computing 360-Degree Compared,” Grid Computing Environments Workshop (GCE '08), 2008, p. 1-10.

Fowler, M. 2010. Richardson Maturity Model: Steps toward the glory of REST. Available: http://martinfowler.com/articles/richardsonMaturityModel.html Retrieved October 10th, 2012.

Franklin, M., Zdonik, S. 1996. "Dissemination-based Information Systems". IEEE Data Engineering Bulletin, Vol. 19 No. 3. P. 21-28.

Gartner, 2011. Gartner says Worldwide Mobile Application Store Revenue Forecast to Surpass $15 Billion in 2011. Retrieved March 25th, 2012.

Gutwin, C., Lippold , M., Nicholas , T. C. 2011. Real-time groupware in the browser: testing the performance of web-based networking. CSCW 2011, p. 167-176.

Gulzar, N. 2002. Fast Track to Strut: What it does and how. Available: http://media.techtarget.com/tss/static/articles/content/StrutsFastTrack/StrutsFastTrack.pdf Retrieved on February 13th, 2012.

Heimbigne, D. 2003. Extending the Siena Publish/Subscribe System, Technical Report CU-CS-946-2003, University of Colorado at Boulder, p. 1-16.

Hohpe, G., Woolf, B. 2004. Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, Boston. 2004.

Huang, Y., Molina, G., 2001. Publish/subscribe in a Mobile Environment. In Proceedings of MobiDE. P. 27–34.

Hamalainen, H. 2011. HTML5: Websockets, http://www.mendeley.com/download/public/1174531/4717378065/1ea8b2b5a59d2aa918b768616d647121279cfb73/dl.pdf. Retrieved on February 15th, 2012.

HTML5 Websocket implementation for the R language 2011. Available: http://www.asymptotix.eu/news/html-5-websocket-implementation-r-language Retrieved on January 20th, 2012.

Hyuk, Y. 2011, Mobile Cloud e-Gov Design and Implementation Using WebSockets API, 204-211, FutureTech. Retrieved on January 20th, 2012.

Heinrich, M., Gaedke, M. 2011. WebSoDa: A Tailored Data Binding Framework for Web Programmers Leveraging the WebSocket Protocol and HTML5 Microdata. ICWE 2011, p. 387-390.

Hoang, T. D., Chonho, L., Niyato, D., Wang, P. 2011. A Survey of Mobile Cloud Computing: Architecture, Applications and Approaches, Wireless Communications and Mobile Computing, Wiley Journals. ISSN 2229-5518.

Jamal, S. 2012. “Combining caching with a cloud hosted proxy to support mobile consumers of RESTful services”, M.Sc. Thesis Submitted to the College ofGraduate Studies and Research, Department of Computer Science, University of Saskatchewan, Saskatoon, Canada. November, pp. 1-85. 2012.

jQueryMobile. Available: http://jquerymobile.com/ Retrieved on December 20th 2011.

Kaazing 2012. Available: http://kaazing.com/products/kaazing-websocket-gateway. Retrieved on January 20th 2012.

Krasner, G. E., Pope, S. T. 1988. A Description of the ModelView-Controller User Interface Paradigm in the Smalltalk-80 System, Journal of Object Oriented Programming, 1(3), Aug.–Sep. 1988, pp. 26–49.

Kazi, R.**;** Deters, R. 2013a, "A Cloud-hosted Hybrid Framework for Consuming Web Services on Mobile Devices", The Third International Conference on Selected Topics in Mobile and Wireless Networking, Montreal, Canada. August 19-21, 2013.

Kazi, R.; and Deters, R. 2013b, "RESTful dissemination of healthcare data in mobile digital ecosystem (DEST 2013), Menlo Park, California, July 24-26, 2013.

Kazi, R.; and Deters, R. 2013c, "A Dissemination-Based Mobile Web Application Framework for Juvenile Ideopathic Arthritis Patients", International Symposium on Network Enabled Health Informatics, Biomedicine and Bioinformatics (Hi-Bi-Bi 2013), Niagara Falls, Canada, August 25-28, 2013.

Larson, J., "Erlang for concurrent programming," Queue, vol. 6, no. 5, pp. 18–23, 2008.

Lomotey, R.K.; Deters, R. "Reliable Consumption of Web Services in a Mobile-Cloud Ecosystem Using REST", *2013 IEEE 7th International Symposium on Service Oriented System Engineering (SOSE),* On page(s): 13 – 24, vol., no., pp.13,24, 25-28 March 2013

Lomotey, R.K., Ralph, D., Using a Cloud-Centric Middleware to Enable Mobile Hosting of Web Services, Procedia Computer Science, Volume 10, 2012, Pages 634-641, ISSN 1877-0509, 10.1016/j.procs.2012.06.081.

Liu, C., Liu, Y., Ma, X., Gao, J. 2010, An Application scheme of publish/subscribe system over clustering Mobile Ad Hoc Networks. P. 1-4.

Lubbers, P., Greco, F. 2010. HTML5 Web Sockets: A Quantum leap in Scalability for the Web. Available: http://soa.sys-con.com/node/1315473. Retrieved on January 20th 2012.

Mühl, G., Ulbrich, A., Herrmann , K., Weis, T. 2004. Disseminating Information to Mobile Clients Using Publish-Subscribe. IEEE Internet Computing 8(3): 46-53.

MVC 2011. MVC – Model View Controller. Available: http://molecularsciences.org/zend/mvc_model_view_controller Retrieved on June 15thth 2012.

MVCsharp.org. The Basics of MVC and MVP. Available: http://www.mvcsharp.org/Basics_of_MVC_and_MVP/Default.aspx Retrieved on June 15th 2012.

Object Management Group 2002. CORBA notification service specification, version 1.0.1. OMG Document formal/2002-08-04. 2002.

Osmani, A. 2012. Developing Backbone.js Applications. Building better JavaScript applications. Available: http://addyosmani.github.com/backbone-fundamentals/#mvc-mvp Retrieved on June 15th 2012.

Osmani, A. 2012. Learning Javascript Design Patterns. Available: http://addyosmani.com/resources/essentialjsdesignpatterns/book/ Retrieved on February 15th 2012.

Perry, R. 2011. Hybrid Mobile apps take off as HTML5 vs native debate continues. http://venturebeat.com/2011/07/08/hybrid-mobile-apps-take off-as-html5-vs-native-debate-continues/ Retrieved on January 17th 2012.

Podnar, I., Hauswirth, M., Jazayeri, M., Mobile Push: Delivering Content to Mobile Users. In Proceedings of the International Workshop on Distributed Event-Based Systems in conjunction with the 22nd International Conference on Distributed Computing Systems, 2002.

PhoneGap. 2012. Available: http://phonegap.com/ Retrieved on March 20th 2012.

Qveflander, N. 2010. Pushing realtime data using html5 Web Sockets. Master's thesis, Umea University – Department of Computing Science. 2010.

Ranck, J. (2010). The Rise of Mobile Health Apps. October 2010. Available: http://pro.gigaom.com/2010/10/report-the-rise-of-mobile-health-apps/. Retrieved on October 30th 2012.

Reenskaug, T. MVC XEROX PARC 1978-79. http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html, 1979.

Sutton, P., Arkins, R., and Segall, B. 2001. Supporting Disconnectedness –Transparent Information Delivery for Mobile and Invisible Computing. In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01), Brisbane, Australia. p. 277-285.

Sencha 2012. Available: http://www.sencha.com/products/touch. Retrieved on March 15th 2012.

Sudha, N., Sujatha, S. 2011. Integrating Soa and Web Services. River Publishers, Aalborg, Denmark. River Publishers, 2011.

Satyanarayanan, M., Bahl, P, Caceres, R., Davies, N. 2009. The Case for VM-Based Cloudlets in Mobile Computing, IEEE Pervasive Computing, vol. 8(4), pp. 14–23.

Srirama, S., Jarke, M., Prinz, W.: Mobile Web Service Provisioning. In: Int. Conf. on Internet and Web Applications and Services (ICIW06), IEEE Computer Society (2006)

Stephen, F. 2006. Event-Driven Programming: Introduction, Tutorial and History. Available: http://eventdrivenpgm.sourceforge.net/ Retrieved on December 20th, 2012.

Thomas, E., Benjamin, C., Cesare, P., Raj, B. 2012. SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST. Prentice Hall, August 10, 2012.

Triantafillou, P., Aekaterinidis, I., Content-based publish/-subscribe over structured p2p networks. 1st International Workshop on Discrete Event-Based Systems, 2004.

Wang, C., Li, Z. 2004. A Computation Offloading Scheme on Handheld Devices. Journal of Parallel and Distributed Computing, Vol. 64, No.6, pp. 740-746. Retrieved on March 20th, 2012.

Wang, Q. 2011. "Mobile Cloud Computing", M.Sc. Thesis Submitted to the College of Graduate Studies and Research, Department of Computer Science, University of Saskatchewan, Saskatoon, Canada. November, pp. 1-80, 2011.

WebSocket.org 2012. Available: http://www.websocket.org/ Retrieved on March 15th, 2012.

WebSocket API 2012. Available: http://dev.w3.org/html5/websockets/ Retrieved on March 20th, 2012.

WebSocket 2012. Available: http://en.wikipedia.org/wiki/WebSocket Retrieved on March 20th, 2012.

WebSocket Protocol 2011. Available: http://tools.ietf.org/html/rfc6455 Retrieved on March 20th, 2012.

W3C 2004. Web Services Architecture. Available: http://www.w3.org/TR/ws-arch/#id2260892 Retrieved on March 20th, 2012.

Webber, J., Parastatidis, S., Robinson, I. 2010. REST in Practice, O'Reilly Media. Retrieved on March 20th, 2012.

Zhang, X. (2012). "N-Screen Application Framework", M.Sc. Thesis Submitted to the College of Graduate Studies and Research, Department of Computer Science, University of Saskatchewan, Saskatoon, Canada. November, pp. 1-85.