An Interoperable Framework for

API Heterogeneity in Mobile Cloud

Storage Systems

A **PROJECT REPORT**

*Submitted by*

Narendra Kumar.S 2010115087

Pragadheesh Chander.S 2010115094

Santhosh Kumar.M 2010115100

*to*

FACULTY OF INFORMATION COMMUNICATION AND ENGINEERING

***For partial fulfilment of the award of the degree of***

**Bachelor of Technology**

***In***

**INFORMATION TECHNOLOGY**



**DEPARTMENT OF INFORMATION SCIENCE AND TECHNOLOGY**

**ANNA UNIVERSITY, CHENNAI 600 025**

**March 2014**

**BONAFIDE CERTIFICATE**

Certified that this report titled “An Interoperable framework for API heterogeneity in mobile cloud storage systems” is the bonafide work of S.Narendra Kumar(2010115087),S.PragadheeshChander(2010115094),M.Santhosh Kumar(2010115100) who carried out the research under my supervision, for the partial fulfillment of the requirements for the award of the degree of the Bachelor of Technology in Information Technology. Certified further that to the best of my knowledge and belief, the work reported here in does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion.

Place: Chennai **Dr.T.Mala**

Date: 25-03-2014 Project guide

Assistant Professor(Sr.Grade)

Department of Information

Science and Technology

Anna University

Chennai – 600 025

**COUNTERSIGNED**

**Dr. A. Kannan**

PROFESSOR AND HEAD

Department of Information Science and Technology

Anna University, Chennai – 600 025

**ACKNOWLEDGEMENT**

We are extremely grateful to **Dr. A. Kannan**, Head of the Department of Information Science and Technology, Anna University, Chennai 600025, for extending the facilities of the Department towards our project and for his unstinting support.

We express our deep gratitude to our guide, **Dr.T.Mala** for guiding us through every phase of the project. We appreciate her thoroughness, tolerance and ability to share her knowledge with us. We thank her for being easily approachable and quite thoughtful. Apart from adding her own input, she has encouraged us to think on our own and give form to our thoughts. We owe her for harnessing our potential and bringing out the best in us. Without her immense support through every step of the way, we could never have it to this extent.

We express our thanks to the review committee members **Dr. K.Vani,** Associate Professor, **Dr.M.Vijayalakshmi,** Senior Assistant Professor, **Ms. E. Uma,** Assistant Professor and **Mrs.M.Priyadharshini** for their valuable suggestions and critical reviews throughout the course of our project.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

**S.Narendra Kumar S.PragadheeshChander M.Santhosh Kumar**

**ABSTRACT**

Mobile cloud applications are considered as the next generation of mobile applications, due to their promise of bonded cloud functionality that augment processing capabilities on demand, power-aware decision mechanisms that allow to utilize efficiently the resources of the device and their dynamic resource allocation approaches that allow to program and utilize cloud services at different levels (SaaS, IaaS, PaaS). However, adapting the cloud paradigm for mobile devices is still in its infancy and several issues are yet to be answered.

Developing a mobile cloud application involves adapting different Web APIs from different cloud vendors within a native mobile platform. Vendors offer web APIs as an interface that allows programming the dynamic computational infrastructure that support massively parallel computing. Deploying a Web API on a handset is demanding for the mobile operating system due to many reasons like compiler limitations, additional dependencies and code incompatibility.

Another problem with the existing cloud services is that many cloud services offered are distinguishable in terms of specification of protocols and standards due to different cloud vendors who provide them. Their APIs therefore are not compatible with each other. When a developer needs to use various services of cloud provided by different cloud vendors, He must first get to know the protocols and API sets of each vendor. This often becomes a tedious process for most of the developers since the API sets are completely different from one other.

This project involves in developing an interoperable framework to support heterogeneous sets of APIs of multiple storage cloud vendors to define new norms of communication between a mobile platform such as android and different mobile cloud vendors. Such norms of framework should be able to help the developers to connect to different cloud services without the tedious process of learning standards of all cloud vendors.

திட்டப்பணி சுருக்கம்

கைபேசி மேகக் கணிமையின் பயன்பாடுகள், அடுத்த தலைமுறை கைபேசியின் பயன்பாடுகளாக கருதப்படும் காரணம், ஒருங்கினைக்கப்பட்ட மேகக்கணிமையின் பயன்பாடுகள் அளிக்கும் உறுதியும் கருவியின் வலங்களை சிறந்த முறையில் கையாளும் விதமே ஆகும். எனினும் மேகக் கணிமையை கைபேசி கருவிகளுடன் பயன்படுத்துவது இன்னும் குழந்தை பருவத்தில்தான் இருக்கிறது மற்றும் பல கேள்விகளும் விடையின்றி நிலவுகின்றது.

கைபேசி மேகக் கணிமையின் பயன்பாடுகள் பல்வேறு மேகக் கணிம விற்பனையாளர்கள் பல்வேறு இணையதள ஏபிஐகளை தழுவி உருவாக்கப்படுகிறது. மேகக் கணிம விற்பனையாளர் இணையதள ஏபிஐகளை, டைனமிக் கணக்கீட்டு உள்கட்டமைப்பையும் அதன் வாயிலாக பெருத்த இணை கணினி முறைகளையும் அனுமதிக்கும் ஒரு இடைமுகமாக விளங்குகிறது. இணையதள ஏபிஐகளை கைபேசியில் பயன்படுத்தல் கைபேசியின் இயக்கு தளத்திற்கு கடினமான காரியமாகும்.

தற்போதைய மேகக் கணிமையின் சேவைகளின் மற்றொரு சிரமம் என்னவென்றால் பல மேகக் கணிமையின் சேவைகள், வேறுபட்ட மேகக் கணிம வழங்குனரின் வேறுபட்ட தரங்களாலும் நெறிமுறைவிவரக்குறிப்பு காரணமாகவும் வேறுபட்டு நிற்கிறது. அவற்றின் ஏபிஐகள் ஒன்றுக்கொன்று ஒத்ததாக இருக்காது. ஒரு பயனர் மேகக் கணிமையின் சேவைகளை பயன்படுத்த விழைந்தால் முதலில் அந்த விற்பனையாளர்க்கு ஏற்ற நெறிமுறைகளும் ஏபிஐகளை கற்க வேண்டிய சூழ்நிலை ஏற்படுகிறது. இது பயனருக்கு மிகுந்த சிரமத்தை ஏற்படுத்துகிறது.

எங்களது திட்டம், வெவ்வேறு மேகக் கணிம வழங்குனரின் வேறுபட்ட ஏபிஐ தொகுப்புகளை ஒன்றாக பயன்படுத்த பரிமாறக்கூடிய தளத்தை ஏற்படுத்தி அதன் மூலம் ஏன்ட்ராய்ட் போன்ற கைபேசி தளத்திற்கும் பல்வேறு மேகக் கணிம வழங்குனருக்கும் இடையே ஒரு புது விதமான தொடர்பை உண்டாக்குவதே ஆகும். அப்படிப்பட்ட பரிமாறக்கூடிய தளம் பயனர்கள் பல்வேறு மேகக் கணிமையின் சேவைகளை எளிதில் பயன்படுத்த உதவும். அது பயனர் ஒவ்வொரு மேகக் கணிம வழங்குனரின் நெறிமுறைகளும் ஏபிஐகளை கற்க வேண்டிய சூழ்நிலையை முற்றிலும் அகற்றிவிடும்.

**Table of contents**

|  |  |  |
| --- | --- | --- |
| Chapter No | Title | Page no |
|  | **ABSTRACT(ENGLISH)**  **ABSTRACT(TAMIL)** | iv  v |
|  | **LIST OF FIGURES** | x |
|  | **LIST OF ABBREVATIONS** | xi |
| 1 | **INTRODUCTION**  1.1 SCOPE OF THE PROJECT  1.2 METHODOLODY  1.3 DEFINITION  1.4 CONTRIBUTIONS OF THE  PROJECT  1.5 OVERVIEW OF THE REPORT | 1  1  1  2  3  3 |
| 2 | **LITERATURE SURVEY**  2.1CLOUD COMPUTING  2.2 MOBILE CLOUD COMPUTING  2.3 CLOUD INTEROPERABILITY  2.4 API HETEROGENEITY  2.5 LIMITATION OF EXISTING WORK | 4  4  4  5  7  7 |
| 3 | **SYSTEM DESIGN**  3.1 SYSTEM ARCHITECTURE  3.2 DETAILED DESIGN  3.2.1 Manager  Interoperability Engine  Session Management  Cache Management  3.2.2 Adapter File system  3.2.3Mobile API set | 9  9  10  11  11  12  13  13  13 |
| 4 | **IMPLEMENTATION**  4.1 MOBILE CLIENT  4.1.1 API setup  4.1.2 Cloud Authentication  4.2 SERVER SETUP  4.2.1 Service Deployment  4.2.2 Session management  4.2.3 Redis Cache  4.2.4 Adapter File System  4.3 CLOUD SERVICE | 14  14  14  14  15  15  15  15  16  16 |
| 5 | **RESULTS AND EVALUATION**  5.1OUTPUT  5.1.1 Upload to cloud service  5.1.2 Download from cloud  5.1.3 List files on Cloud  5.1.4 Search files on Cloud  5.1.5 Transfer files between clouds  5.2 PERFORMANCE  Performance Evaluation | 17  17  17  18  18  18  18  19  19 |
| 6 | **CONCLUSION AND FUTURE WORKS** | 21 |
|  | **REFERENCES**  **APPENDIX A: CODE IMPLEMENTATION**  **APPENDIX B: SCREENSHOTS** | 22 |

**LIST OF FIGURES**

**FIGURE NO TITLE PAGE NO**

2.1 Architecture of UCI 6

2.2 Example of Orchestration 6

3.1 Architecture of Interoperable 9

Framework

3.2 Overall interaction between components 10

5.1 No. of requests vs. Time graph 19

5.2 Size vs. Time graph 20

B.1 File uploads to cloud service 33

B.2 File downloads from cloud service 33

B.3 List files on cloud services 34

B.4 Search files on cloud services 34

B.5 Transfer between cloud services 35

**LIST OF ABBREVATIONS**

AFS – Adapter File System

API – Application Program Interface

IE – Interoperability Engine

MCC – Mobile Cloud Computing

MCM – Mobile Cloud Middleware

REST – Representational State Transfer

SOAP – Simple Object Access Protocol

TP – Transaction Process

**CHAPTER 1**

**INTRODUCTION**

Mobile Cloud Computing (MCC) is arising as a prominent research area that is seeking to bring the massive advantages of the cloud to the constrained smartphones and to enhance the telecommunication infrastructures with self-adaptive behavior for the provisioning of scalable mobile cloud services. MCC focuses on the benefits that can be achieved by the mobile resources when a mobile operation such as data storage or processing is dele-gated or offloaded to the cloud. These benefits include extended battery lifetime, improved storage capacity and increased processing power, thus enriching the mobile applications along with the mobile user experience. Moreover, MCC focuses on finding an optimal configuration of a mobile cloud infrastructure in order to handle the oscillating telecommunication loads, to facilitate the process of deploying services without managing the underlying technology and to reduce operational and provisioning costs.

Developing a mobile cloud application involves adapting different Web APIs from different cloud vendors within a native mobile platform and combining those distinguishable specification of protocols and standards of different cloud vendors into a framework to support heterogeneous sets of APIs.

* 1. **Scope of the project**

This project is targeted for mobile platforms such as Android to provide users with a one stop solution to access the cloud storage services from one single point of contact. This project is intended for the mobile application developers for easy access of files across various cloud services without the need to understand the infrastructure or API definition of the service providers.

This project is initially developed with 2 working cloud vendors (Google drive and DropBox) and provides developers with basic functionalities but can be extended to support other existing and newly ventured open source cloud vendors with all functionalities.

* 1. **Methodology**

Proposed system is a framework which basically acts between the mobile device and cloud services for service delegation.

A framework created will be able to support heterogeneous sets of APIs of multiple storage cloud vendors .The framework hides the complexity of dealing with multiple cloud providers by abstracting the APIs from different clouds in a common operation level so that the service functionality of the framework can be added based on combining different cloud services.

When a mobile application tries to delegate a mobile task to a cloud, the user click sends a notification in the form of user-request. The user-request includes the cloud platform on which the operation to be performed. Once receiving the request, it is processed by the central Manager for creating the adapters that will be used in the transactional process with the clouds. When the user-request is forwarded to the Manager, it first creates a session assigning a unique identifier The user request is then handled by the Interoperability-Engine and suitable actions are performed pertaining to user-request.

* 1. **Definition**

**TP Handler:** TP handler handles the incoming request from various devices for suitable service invocation and sends the response back to the deice

**API setup:** API setup is a module which sets up the configurations so as to communicate with cloud provider. The setup is established by creating adapters for the cloud service.

**Session management:** Creates a session for every request made by the user by assigning a unique identifier. The session is cleared on completion of the request.

**Cache management:** This module caches the frequently requested files which cuts out the redundant communication with the user device and also avoids time delay

**Transactional space:** Transaction space is a transient storage space. It is used to store the files of request temporarily. Files storage is a normal file system with type, size, directory and other metadata such as date created, modified, last accessed.

**Adapter file system:** Adapters are executable interfaces.It Consist of the client ID and secret key that are required to setup a standard communication to that particular cloud vendor.Once the adapters are executed, the requested function is looked up and executed.

* 1. **Contribution of the project**

This project involves adapting different Web APIs from different cloud vendors within a native mobile platform and combining those distinguishable specification of protocols and standards of different cloud vendors into a framework which helps the developers to connect to different cloud services without the tedious process of learning standards of all cloud vendors and also additional cloud vendors can be added to current framework with ease

* 1. **Overview of the report**

The overview of the report is explained as follows. Chapter 2 discusses the Literature Survey, the limitations of the existing work and the objective of the project. Chapter 3 explains about the architecture, detailed design and all the modules in detail with the activity diagrams. Chapter 4 lists the implementation details of various modules with their input and output. Chapter 5 summarizes the report and presents directions for future work.

**CHAPTER 2**

**LITERATURE SURVEY**

Creating homogeneity in API involves understanding the concepts of cloud computing and mobile cloud computing, their differences and similarities, API structure and cloud interoperability. The review of literature in these areas is given in this chapter.

**2.1 Cloud Computing**

Cloud computing is a new computing paradigm that allows users with different computing demands to access a shared pool of configurable computing resources (e.g., servers, network, storage, database, applications and services). Many commercial cloud providers have emerged in the past 6-7 years, and each typically provides its own cloud infrastructure, APIs and application description formats to access the cloud resources, as well as support for service level agreements (SLAs)[8].

The three service models of cloud, *SaaS* (Software as a Service), *PaaS* (Platform as a Service) and *IaaS* (Infrastructure as a Service) on different levels provide computational services to the consumers with essential characteristics such as on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service.

**2.2 Mobile Cloud Computing**

Mobile cloud computing is the combination of cloud computing and mobile networks to bring benefits for mobile users, network operators, as well as cloud computing providers. The ultimate goal of MCC (mean of MCC is Mobile/Cloud Computing) is to enable execution of rich mobile applications on a plethora of mobile devices, with a rich user experience. “Mobile Cloud Computing at its simplest refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just smartphone users but a much broader range of mobile subscribers [3].

Concept of Mobile cloud computing has several limitations and several factors such as limitations in storage and processing power, and thus need a change in infrastructure. These limitations are overcome by offloading the process to the cloud vendors [4].

Despite of the limitations of Mobile cloud computing, it has several advantages such as dynamic provisioning, scalability and ease of integration [3].

**2.3 Cloud Interoperability**

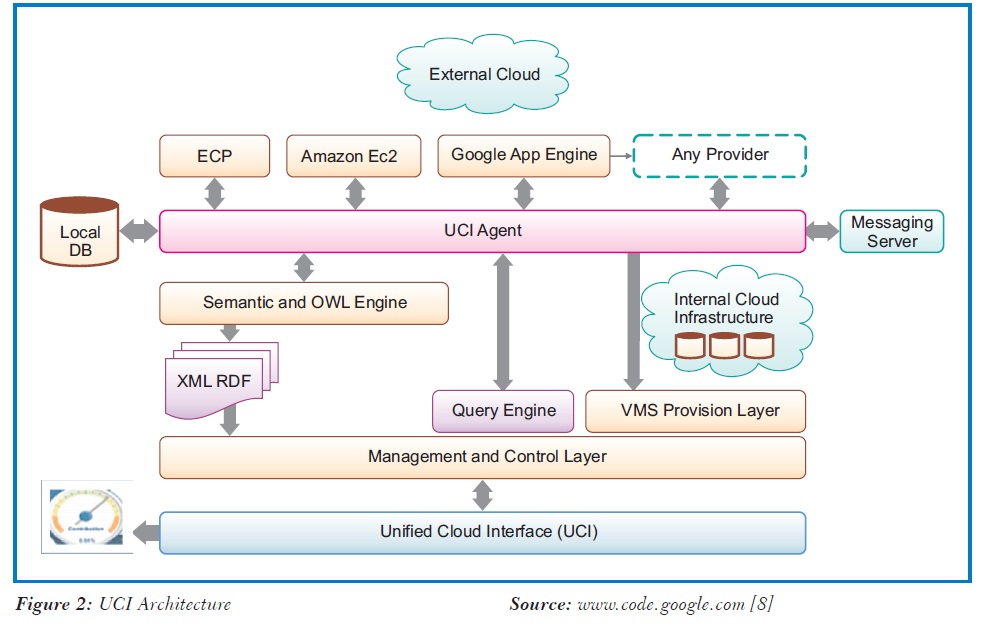
With increase in number of cloud service providers and each having their own infrastructure which the consumer must follow, it becomes impossible to follow generic set of protocols to use all the cloud vendors and creates vendor lock-in [1].

In the cloud Ecosystem, the rapid development of cloud computing in the absence of early standards beside fragmented business policies have created various types of heterogeneity that give rise to new problems such as vendor lock-in. Vendor lock-in problem is a state when code and data cannot be moved from one cloud to another because of dissimilarities among underlying architectures and programming languages. Though the problem might be attractive in business competition, it causes several challenges particularly data integrity, interoperability and portability. Hence, cloud-users become vulnerable to problems like reliability if service provider stops serving request and data extraction from the hosting servers and price hike [8].

To address this issue, various Cloud interoperability techniques are being drafted and implemented.

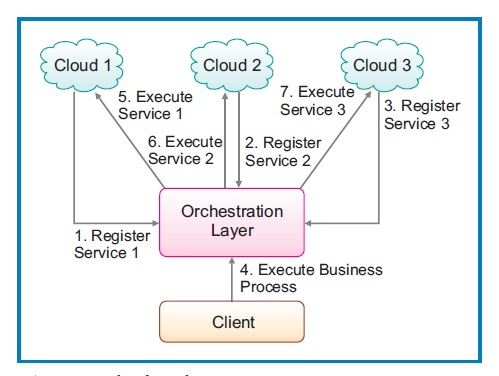
Unified Cloud Interface/Cloud Broker is a approach wherein Cloud computing vendors have formed a common platform — cloud computing interoperability forum (CCIF) — to address the problem of cloud interoperability and standardization. The purpose of CCIF is to discuss and come up with a common cloud computing interface. CCIF is planning to come up with a *unified cloud interface* [1]*.*

Features of UCI (unified cloud interface) include unifying various clouds API and abstract it behind an open and standardized cloud interface. Thus a key driver of the unified cloud interface (UCI) is to create an API about other APIs. The purpose of cloud broker is to serve as a common interface for the interaction between remote platforms, networks, systems, applications, services, identity and data.



**Fig 2.1 Architecture of UCI**

Another approach to cloud interoperability is *Enterprise Cloud Orchestration Platform.* This approach suggests an orchestration layer between every cloud and the consumer and formation of InterCloud. InterCloud means a federation of all kinds of clouds. All these clouds will be full of applications and services. It will not be possible to use these without some type of orchestration. A single management platform is provided to conveniently manage multiple clouds that facilitates businesses to migrate deployments. It helps businesses to manage and scale cloud deployments as well as facilitate application migration and management across multiple clouds.



**Fig 2.2 Example of Orchestration**

**2.4 API Heterogeneity**

Application Programming Interface (API) is an interface supplied by OS vendors or service providers that allows an application written in a high-level language to access specific data or functions from the API distributor. Therefore, APIs play an important role in delivering a rich experience to mobile users.

Numerous cloud vendors provide different services with custom-build policies, infrastructures platforms, and APIs that make the cloud landscape heterogeneous. Such variations cause interoperability and portability as a major challenge in cloud computing. There is a notion that business competition also diversifies cloud providers with their heterogeneous frameworks, exacerbating heterogeneity on the cloud side [8]**.**

On the cloud side, the majorities of cloud providers develops and deploy their own proprietary APIs to describe syntax of specific operations to be utilized by their clients. A drastic growth in the number of cloud providers has created a huge silo of different APIs that intensifies the difficulty of developing applications due to interpreting semantics of data and operations. This outlook, results in API variation intensifying interoperability and portability issues**.**

The UCI approach for cloud interoperability is more intriguing because unlike orchestration it needs only minimal or no support from the cloud providers and has a programmatic point of contact. The general approach is to create a new set of API combining various APIs of cloud providers.

This cross-platform API is language and platform-agnostic application programming interface for using multi-Cloud resources, Designing a generic agent framework for embodying various stakeholders, e.g. Cloud vendors and their resources, Cloud users of various types, and collection of modules that can be used to adapt the agents to support needed functionalities [2].

The cross-platform API allows programmers to easily develop auto-scalable cloud applications with properties such as generic resource abstraction which intends to hide the actual particularities of each Cloud provider or distributed solution, under a simple common interface. Exploiting this abstraction, the user can easily switch from one provider to another without modifying the source code [2].

Delegation of tasks from mobile devices to cloud by using the cross-platform API suggested above enables increasing the processing limit of the mobile Applications. Computational/storage delegation has become a common operation that is supported by any mobile platform through various mechanisms (e.g. Web sockets, REST-based requests, etc.)[5].

The Offloading process proceeds as follows, In asynchronous process, when a mobile application sends a request to access a cloud service, the handset immediately gets a response that the transaction has been delegated to remote execution in the cloud, while the status of the mobile application is sent to local background so that the mobile device can continue with other activities. Once the process is finished at the cloud, an asynchronous message about the result of the task is sent back to the mobile, so as to reactivate the application running in the background, and thus the user can continue the activity.

**2.5 LIMITATION OF EXISTING WORK**

Cloud interoperability discussed above is still under research and has only few existing working prototypes. These existing works focus on creating a new standard for all cloud services for cloud interoperability but creating a new standard for already well developed cloud services is an impossible task since it demands changing the their protocols.

This chapter explained the existing works and their limitations. The next chapter discusses the detailed design and architecture of the project proposed.

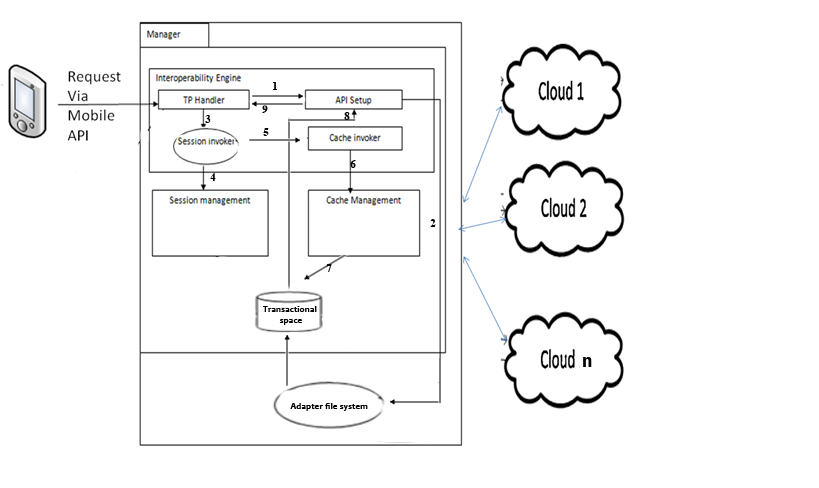
**CHAPTER 3**

**SYSTEM DESIGN**

This chapter discusses the design methodology adopted to connect user device with different cloud services. First the overall methodology adopted is explained, followed by the detailed design of the components.

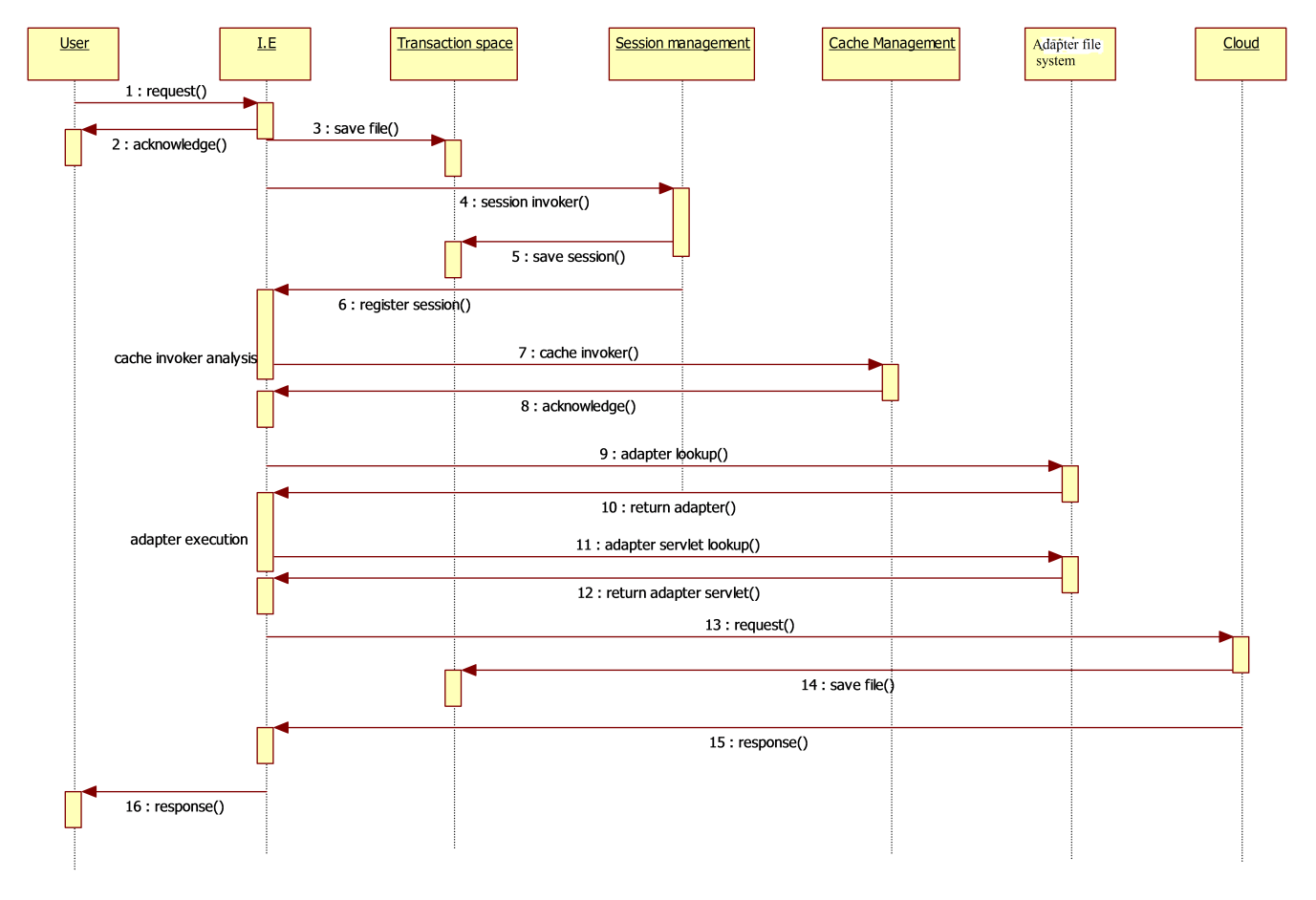
**3.1 SYSTEM ARCHITECTURE**

Figure 3.1 represents the overall system architecture. Every cloud services are distinguishable in terms of specification of protocols and standards due to different cloud vendors who provide them. Mobile client just offloads its request to the manager and the manager takes care of the cloud interoperability using web APIs and credentials given by the client.

** Fig 3.1 Architecture of interoperable framework**

**3.2 DETAILED DESIGN**

The Fig 3.2 represents the overall interaction between components. The user makes a request from mobile device, the request is fetched by an interoperability engine which then proceeds to transactional space(if file storage is necessary). Session and Cache management is handled. After initiating a session and initializing the cache, an adapter lookup is established. The respective adapter servlet is called, based on the user request to cloud service. The service is then granted and the response is sent back to the user. Further each module is illustrated in detail.

****

**Fig 3.2 Overall interactions between components**

**Modules in the System:**

* + 1. **Manager**

**Interoperable engine**

**TP Handler module:**

Interoperable Engine, IE handles the request from the Mobile device using Mobile API set and requests to the system would be based on the REST protocol. Request would contain information such as *Name of the Service provider, Service requested, Configuration parameters, Files, Cloud credentials*. Eg: (Google drive, download, Android 4.2.0, final.doc, username , password).It then handles the incoming request to the system and retrieves the information from the request. After receiving the request it sends an acknowledgement message. The acknowledgement to the mobile device is sent using the asynchronous notification. This notification is sent as push notifications from the Google cloud messaging service. After acknowledging the request, the file is stored in the transactional space.

**Session invoker**

The request is received by TP handler. It transfers the control to the session management module passing the request parameters. After initiating a session, it acknowledges the process so as to move on to the cache management module.

**Cache invoker**

Once the session is initiated, this module, cache invoker, is invoked. This module analyses the request on whether the file requested is to be cached or not, analysis can be based on various factors such as number of times requested, number of times appeared in different vendors. After analyzing it invokes the cache management module.

**API Setup**

This module is invoked after receiving the request. Name of the cloud vendor and required function is obtained from request parameter. This module sets up the configuration parameters so as to communicate to the cloud provider. This set up is done by creating adapters for the required cloud vendor. After creating the adapters it calls the respective adapter servlet and the corresponding function.

**Transaction space**

Transaction space is a transient storage space storing the progress of the requests. It is used to store the details of the session and the files of request temporarily. Files storage is a normal file system with type, size, directory and other metadata such as date created, modified, last accessed.

**Session management**

Session management is invoked by the session invoker module of the Interoperability engine. The request parameters are obtained from the interoperability engine and it creates a session for every request made by the user. Each session consists of a  *Unique session key, User OS configurations and cloud credentials*  . Unique session key is auto generated for further reference. OS configuration and cloud credentials of a particular user are sent along with the request and are transferred while invocation. These sessions are stored in transaction space and are cleared upon the completion of the request.

**Cache management**

Cache management module is invoked by the cache invoker module of the interoperability engine. The File to be cached is identified from the cache invoker and this module caches the frequently requested files. For example, files that are to be uploaded to many clouds for backup can be cached. It also reduces the redundant communications to the mobile device. It is implemented using Redis. Redis is an in-memory key-value store for small chunks of arbitrary data.

**3.2.2 Adapter file system**

Adapters are executable interfaces. It consist of the client ID, secret key that are required to setup a standard communication to that particular cloud vendor. It is stored in a separate file system. Once the adapters are executed, the requested function is looked up and executed. It receives input objects (like filename in case of upload or download) and then it calls the function with given parameters. After successful execution of request it returns the response back to the API setup

**3.2.3 Mobile API set**

Android client application is created to access the REST services deployed. The user sends the request and the request parameters are identified by TP Handler. Based on the request, the service is invoked and after completion of the processes the client receives the acknowledgement.

Detailed description set of each and every module implementation is discussed in Chapter 4.

**CHAPTER 4**

**IMPLEMENTATION**

**4.1 MOBILE CLIENT**

The client application developed in android accesses the server through URL link. The client sends the request from mobile device through mobile API set. The request to the system would be based on REST protocol. The user must also specify the service required from the particular cloud vendor. For every service requested there will be a unique URL. The client application delegates the input parameter to the web service which is deployed over the server.

**4.1.1 API setup**

This module gets all the input parameters from the request made. This module sets up the configuration so as to communicate with cloud vendor. The setup is completed by creating adapters for particular cloud service. Before creating adapters the user must be authenticated to use the cloud service.

**4.1.2 Cloud authentication:**

For every service requested by the user, the user must be authenticated in order to access the particular cloud service. A token will be generated for every service invocation by the user. The authentication process is done by the cloud vendor itself. After passing the authentication process the task is carried out in the server.

**4.2 SERVER SETUP:**

The server is set to accept the incoming request and based on the request parameter the suitable tasks are performed. The server is deployed over the World Wide Web which provides different URLs for different service invocation.

**4.2.1 Service deployment:**

Each service consists of a servlet which handles the incoming request from the client application. The servlet calls the particular service requested for the task completion. The service is deployed over the following URL

http://www.middleware.narendrakumar619.cloudbees .net

**4.2.2 Session management:**

It creates a unique session key/token which saves the user configurations like *Device, OS, Locality , IP address* . This token validates every users request made to access the cloud storage. The token is autogenerated by cloud vendor and are cleared on completion of the request.

**4.2.3 Redis Cache management:**

Redis is an in-memory key-value store for small chunks of arbitrary data. It cuts out redundant communication to the mobile devices. It gets the input parameters. It caches the frequently requested file in the server so that when user request for the particular file, it need not be fetched from cloud whereas the one that resides on server will grant the request.

**4.2.4 Adapter file system:**

Adapters are executable interfaces. It consist of client ID, secret key that are required to setup a standard communication to particular cloud vendor. The adapters are called on request from the user . It gets the input objects(like filename in case of upload or download). After getting input parameters the requested function is looked up and executed. It returns the response back to the API setup.

**4.3 CLOUD SERVICES**

File management in cloud storage services requires the developer verification from the cloud vendor. An application is set up on the cloud vendor so that it allows communication from local clients to access the server.

**4.3.1 Application setup to use cloud service**

A cloud application is set up on cloud vendors ( Dropbox and Google Drive) such that it provides access to developers to use their cloud storage. After setting up the application the clients shall perform file management on cloud services. The cloud application set up is transparent to the user clients.

The next chapter discusses the results and performance evaluation of the implemented project.

**CHAPTER 5**

**OUTPUT AND RESULTS**

**5.1 OUTPUT:**

The API provides features as,

Upload a file to google drive and dropbox.

Download a file from google drive and dropbox.

List files of google drive and dropbox.

Search files from google drive and dropbox.

Transfer file from google drive to dropbox and vice versa.

The service resides at the url,

http://www.middleware.narendrakumar619.cloudbees .net

**5.1.1 Upload to cloud service:**

The service is requested through the following URL, the input parameters like (FileName,CloudName,Code) are obtained from the user for validation. Once the user is authorized the selected file will be uploaded to the cloud storage.

<http://www.middleware.narendrakumar619.cloudbees.net/rest/cloud/upload>

**5.1.2 Download from cloud service:**

The service is requested through the following URL, the input parameters like (FileName,CloudName,Code) are obtained from the user and the file is searched in the cloud storage. The file is downloaded. If the file is not available in the storage, it displays 404 error.

http://www.middleware.narendrakumar619.cloudbees.net/rest/cloud/download

**5.1.3 List files on cloud:**

The input parameters like (CloudName,Code) are obtained from the user and all the files in the cloud storage are listed.

<http://www.middleware.narendrakumar619.cloudbees.net/rest/cloud/list>

**5.1.4 Search files on cloud:**

The service is requested through the following URL, the input parameters like (FileName,CloudName,Code) are obtained from the user and the file is searched in the cloud storage and the result is displayed.

<http://www.middleware.narendrakumar619.cloudbees.net/rest/cloud/search>

**5.1.5 Transfer files between cloud service:**

The service is requested through the following URL, the input parameters like (FileName,FromCloudName,ToCloudName,FromCode,ToCode) are obtained from the user. The subsequent file is transferred to the required cloud.

<http://www.middleware.narendrakumar619.cloudbees.net/rest/cloud/transfer>

**5.2 PERFORMANCE EVALUATION**

**5.2.1 Number of request Vs Time graph:**

The number of incoming request is compared to time on mobile and desktop platform. Time is calculated from system clock i.e. the system time before service invocation and after sending response to mobile client is noted and their difference is compared with number of incoming request which can be viewed from cloud vendors console.

Time(sec)

No of request

**No of request vs. Time graph**

From the graph depicted , we can conclude that service invocation through mobile takes more time when compared to desktop invocation. The delay is due to mobile platform has more number of communications than desktop. From the graph we can also analyze that the graph rises at the same rate for desktop whereas for mobile it varies.

**5.2.2 Size Vs, Time graph:**

The size of the file in transaction is compared to time on mobile and desktop platform. Time is calculated from system clock i.e. the system time before service invocation and after sending response to mobile client is noted and their difference is compared with size of the file which can be obtained from file properties parameter.

Time(sec)

Size(Mb)

**Size vs. Time graph**

From the graph depicted , we can conclude that service invocation through mobile takes more time when compared to desktop invocation but it merges at a point where the file size exceeds 3 Mb, it is because for a larger file the byte code conversion is same in mobile and desktop and the only time variation is due to additional communication in mobile. From the graph we can also analyze that the graph varies for desktop and mobile. For a larger file, the time taken for transaction is almost equivalent.

The project conclusion and future works and features that can be added are discussed in chapter 6.

**CHAPTER 6**

**CONCLUSION AND FUTURE WORKS**

The end product is a framework which basically serves as a bridge between the mobile device and cloud services for service delegation.

The mobile cloud application involves adapting different Web APIs from different cloud vendors and combining those distinguishable specification of protocols and standards of different cloud vendors into a framework. The framework is currently implemented with Dropbox and Google Drive and also it provides feature to add additional cloud vendor in near future.

The implementation so far has addressed the issue of API heterogeneity, which is an initial step towards the solution of cloud interoperability. The future work has to focus on implementing orchestration layer between the cloud and the end user for the federation of clouds.

**REFERENCES**

1. A V Parameswaran and Asheesh Chaddha , Cloud Interoperability and Standardization - SETLabs Briefings, 2009.
2. Dana Petcu, Marian Neagul , Building an Interoperability API for Sky Computing - 2011.
3. Hoang T. Dinh, Chonho Lee and Dusit Niyato , A Survey on Mobile Cloud Computing: Architecture, Applications and Approaches.
4. Huber Flores, Satish Narayana Srirama and Carlos Paniagua , A Generic Middleware Framework for Handling Process Intensive Hybrid Cloud Services from Mobiles - Institute of Computer Science Distributed System Group, 2011.
5. Huber Flores, Satish Narayana Srirama , Mobile Cloud Middleware – elseiver, may 2013.
6. Peng Xiao and Yanping Zhang , CS-Mobile: A Cloud-based Distributed Storage Middleware for Mobile Devices - International Journal of Smart Home, 2013.
7. Zhizhong Zhang, Chuan Wu and David W.L. Cheung , A Survey on Cloud Interoperability: Taxonomies, Standards, and Practice , march 2013.
8. Zohreh Sanael, Saeid Abolfazli, Abdullah Gani , Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges – IEEE communication survey, 2012

**Appendix A**

**CODE IMPLEMENTATION**

**Manager**

**A.1 Interoperability Engine**

**TP Handler**

TP handler handles the incoming request from various devices for suitable service invocation. It sends the response back to the device

<form action="rest/cloud/download" method="GET" >

@GET

@Path("/downloadexample")

@Produces("application/octet-stream")

MCM m= new MCM();

m.downloadFiles(filename,filepath,cloud,code);

result="success";

**API Setup:**

The request gets the required parameters such as (FileName, CloudVendor, Function). This setup is done by calling the adapter for the particular cloud vendor in the Adapter file system

mcm.java

if(!drop)

throw new IOException();

System.out.println(authentication(cloud));

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

code\_dropbox=br.readLine();

client\_drop.downloadFile(filename,filepath,code\_dropbox);

dropboxclient.java

DbxAppInfo appInfo = new DbxAppInfo(APP\_KEY, APP\_SECRET);

config = new DbxRequestConfig("JavaTutorial/1.0", Locale.getDefault().toString());

webAuth = new DbxWebAuthNoRedirect(config, appInfo);

authFinish = webAuth.finish(url);

client = new DbxClient(config, authFinish.accessToken);

DbxEntry.File downloadedFile = client.getFile("/" + fileName, null,outputStream);

**Cache Management:**

Cache management is handled by Redis. Redis is an in-memory key-value store for small chunks of arbitrary data.Currently all the files are stored in cache (in case of upload and download)

Code sample to cache a file:

import redis.clients.jedis.Jedis;

 Jedis j=new Jedis("localhost");  
        j.connect();

//Read a file and store it in cache

  j.hset(username, filename, readFile());

**A.2 Adapter File System**

API setup uses the cloud adapter files based on the user request to particular cloud service.The client ID and Secret Key are set to communicate with the cloud provider. A temporary token is generated to authenticate the application/user to use the particular cloud service. Once authentication is proved valid, the requested service is granted.The code sample denotes the adapter instance creation and call to authentication

Dropbox Adapter:

DropboxClient client = new DropboxClient();

String s= client.doDrop\_authentication();

Google drive Adapter:

GDriveClient client = new GDriveClient();

String s=client.doGDrive\_authentication();

**A.3 Mobile API set**

The service has been deployed at the following URL

<http://www.middleware.narendrakumar619.cloudbees.net>

The client devices can access the URL to obtain the following services

* Upload a file to google drive and dropbox
* Download a file from google drive and dropbox
* List files of google drive and dropbox
* Search files from google drive and dropbox
* Transfer file from google drive to dropbox and vice versa

The android mobile set can access the services with the following code sample,

**Authentication**

public void provide\_authentication\_code()

{

Spinner cloud = (Spinner) findViewById(R.id.cloud);

String cloudname = cloud.getSelectedItem().toString();

Intent browserIntent=null;

Toast.makeText(getApplicationContext(), cloudname, Toast.LENGTH\_LONG).show();

if(cloudname.equalsIgnoreCase("dropbox"))

{

browserIntent = new Intent(Intent.ACTION\_VIEW, Uri.parse("https://www.dropbox.com/1/oauth2/authorize?locale=en\_US&client\_id=6ue32gjzt79gmtm&response\_type=code"));

startActivity(browserIntent);

}

else if(cloudname.equalsIgnoreCase("google drive"))

{

browserIntent = new Intent(Intent.ACTION\_VIEW, Uri.parse("https://accounts.google.com/o/oauth2/auth?access\_type=online&approval\_prompt=auto&client\_id=553419148291.apps.googleusercontent.com&redirect\_uri=urn:ietf:wg:oauth:2.0:oob&response\_type=code&scope=https://www.googleapis.com/auth/drive"));

startActivity(browserIntent);

}

}

**ListFiles:**

public void listfiles() throws ClientProtocolException, IOException

{

EditText txt\_filename = (EditText) findViewById(R.id.filename);

String filename = txt\_filename.getText().toString();

EditText txt\_code = (EditText) findViewById(R.id.code);

String code = txt\_code.getText().toString();

Spinner spinner\_cloud = (Spinner) findViewById(R.id.cloud);

String cloud = spinner\_cloud.getSelectedItem().toString();

/\*Document doc;

doc = Jsoup.connect("http://mcm.narendrakumar619.cloudbees.net/listfile\_index.jsp").data("cloud", "Dropbox").post();\*/

HttpClient httpclient = new DefaultHttpClient();

HttpPost httppost = new HttpPost("http://middleware.narendrakumar619.cloudbees.net/rest/cloud/list");

List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();

nameValuePairs.add(new BasicNameValuePair("cloud", cloud));

nameValuePairs.add(new BasicNameValuePair("code", code));

httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

HttpResponse response = httpclient.execute(httppost);

System.out.println("Response Code : "

+ response.getStatusLine().getStatusCode());

BufferedReader rd = new BufferedReader(

new InputStreamReader(response.getEntity().getContent()));

StringBuffer result = new StringBuffer();

String line = "";

while ((line = rd.readLine()) != null) {

result.append(line);

}

Toast.makeText(getApplicationContext(), response.getStatusLine().getStatusCode() + " " + filename + " " + cloud + " " + result , Toast.LENGTH\_LONG).show();

//GetPageContent("http://mcm.narendrakumar619.cloudbees.net/ListServlet");

}

**SearchFile:**

public void searchFile() throws ClientProtocolException, IOException

{

EditText txt\_filename = (EditText) findViewById(R.id.filename);

String filename = txt\_filename.getText().toString();

EditText txt\_code = (EditText) findViewById(R.id.code);

String code = txt\_code.getText().toString();

Spinner spinner\_cloud = (Spinner) findViewById(R.id.cloud);

String cloud = spinner\_cloud.getSelectedItem().toString();

HttpClient httpclient = new DefaultHttpClient();

HttpPost httppost = new HttpPost("http://middleware.narendrakumar619.cloudbees.net/rest/cloud/search");

List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>();

nameValuePairs.add(new BasicNameValuePair("filename", filename));

nameValuePairs.add(new BasicNameValuePair("cloud", cloud));

nameValuePairs.add(new BasicNameValuePair("code", code));

httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));

HttpResponse response = httpclient.execute(httppost);

System.out.println("Response Code : "

+ response.getStatusLine().getStatusCode());

BufferedReader rd = new BufferedReader(

new InputStreamReader(response.getEntity().getContent()));

StringBuffer result = new StringBuffer();

String line = "";

while ((line = rd.readLine()) != null) {

result.append(line);

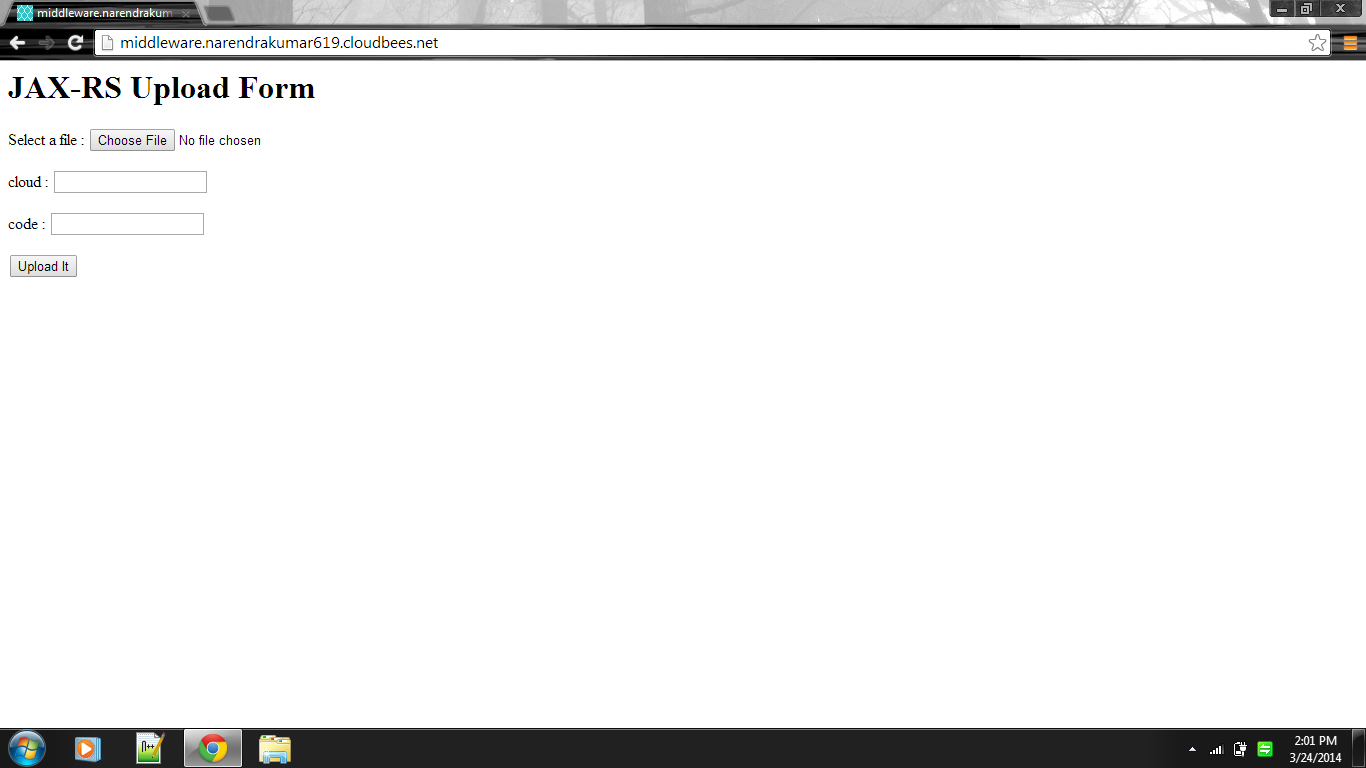
}

Toast.makeText(getApplicationContext(), response.getStatusLine().getStatusCode() + " " + filename + " " + cloud + " " + result , Toast.LENGTH\_LONG).show();

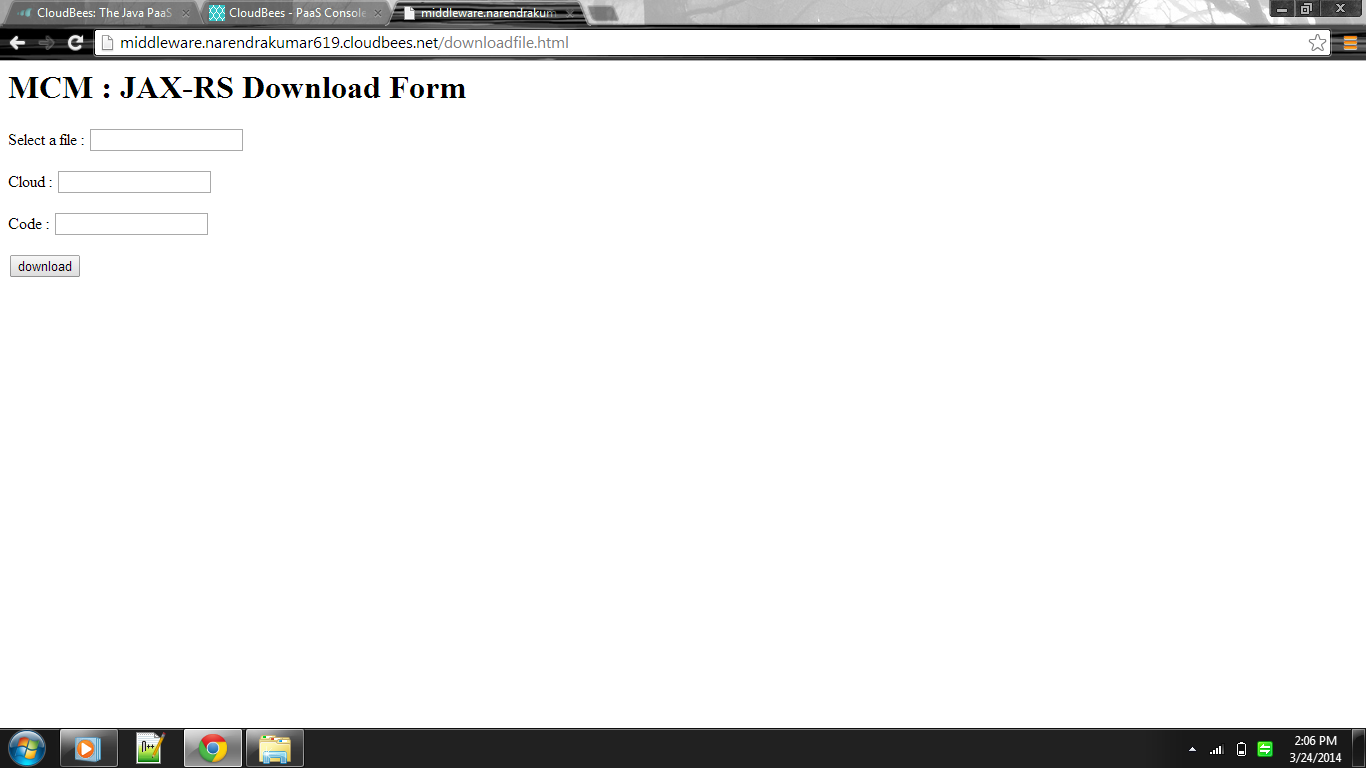
}

**Appendix B**

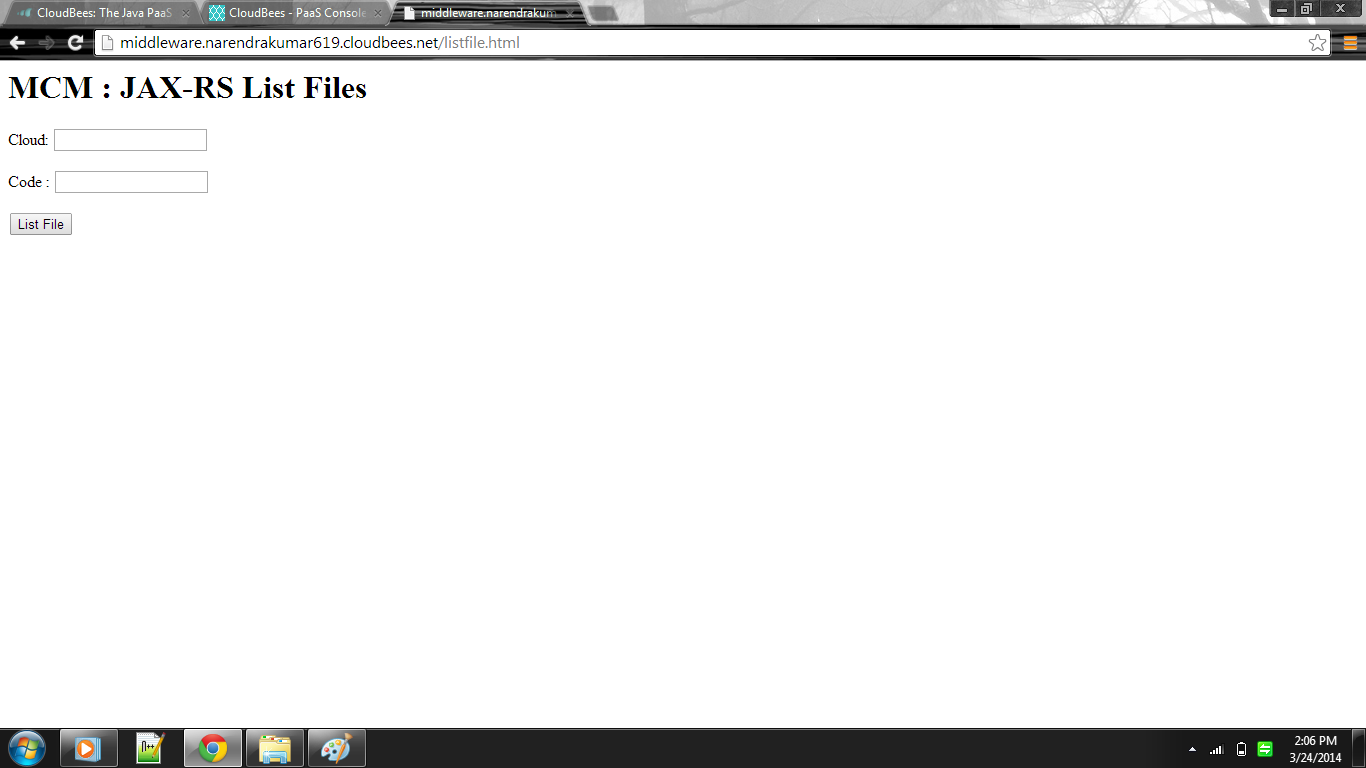
**SCREENSHOTS**

****

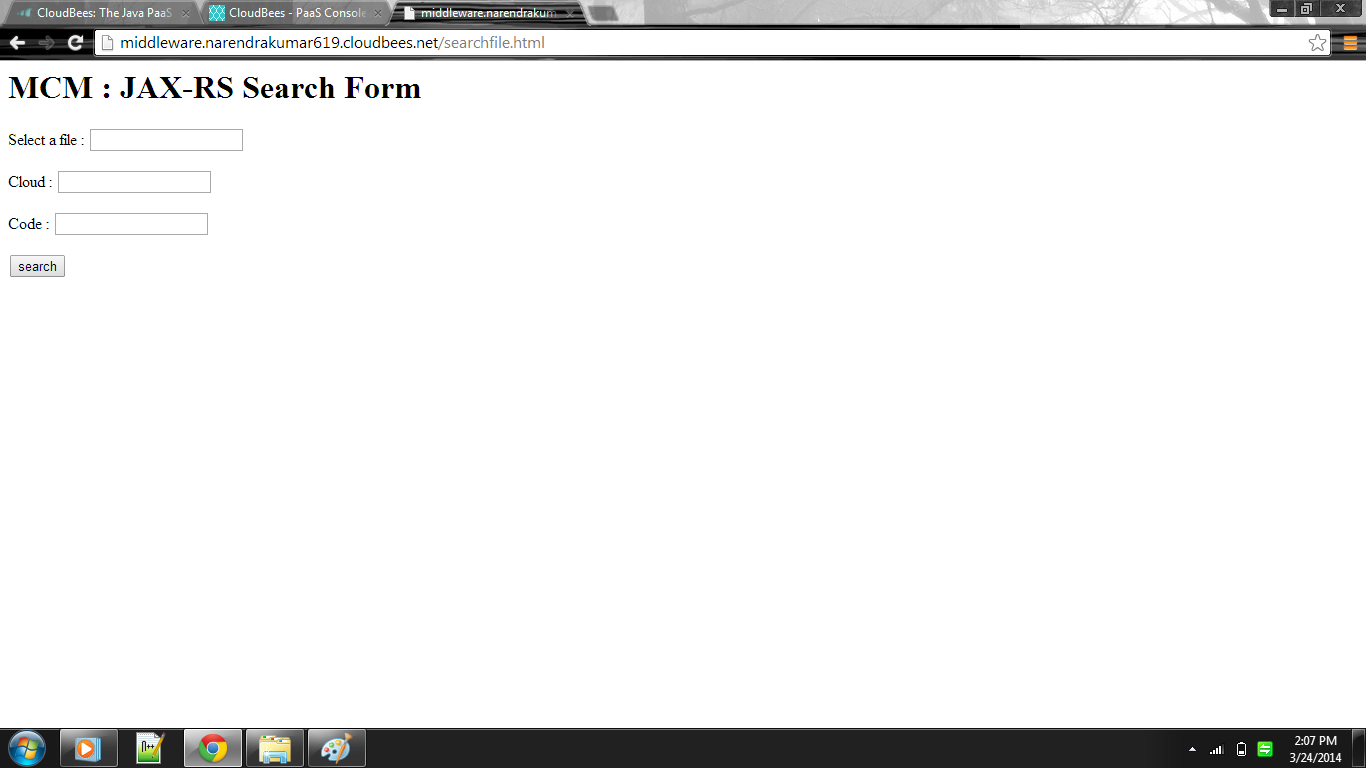
**B.1 File upload to cloud service**

****

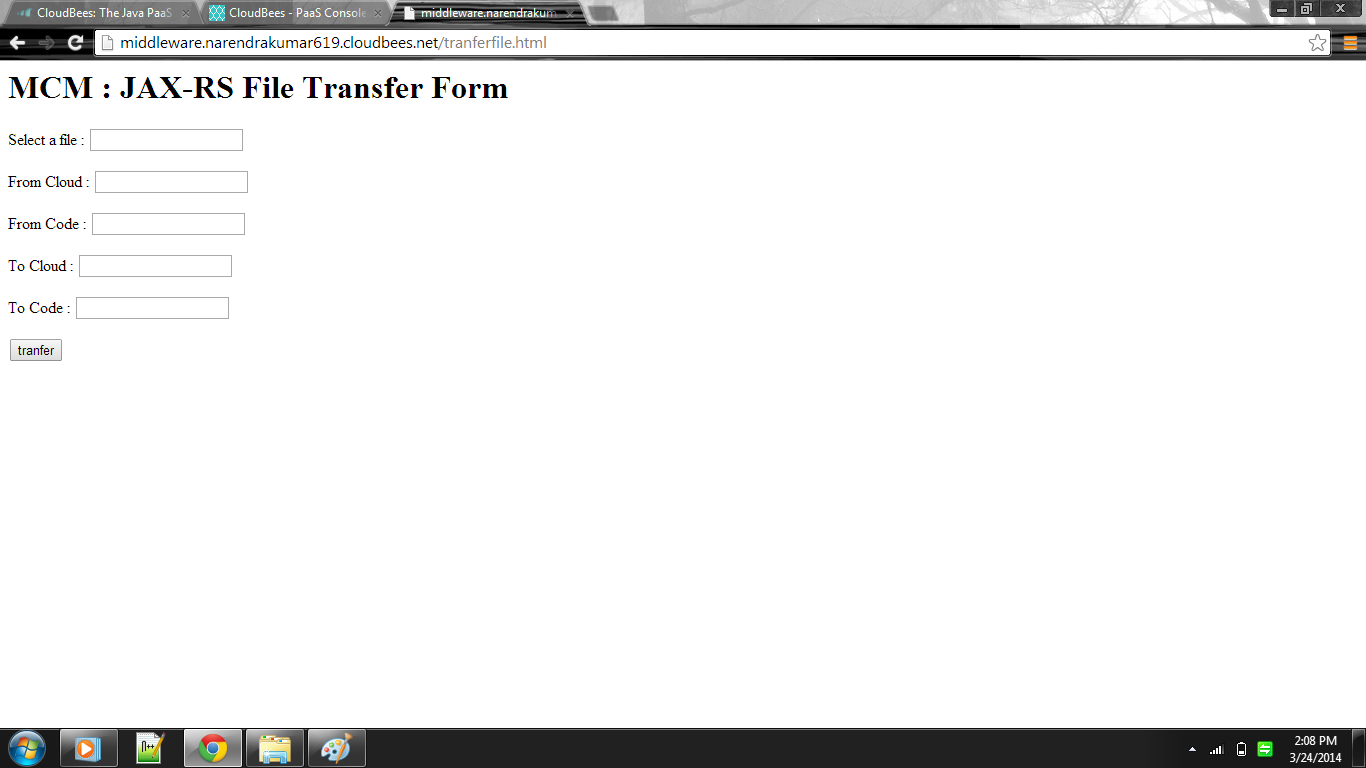
**B.2 File Download from cloud service**

****

**B.3 List files on cloud**

****

**B.4 File search on cloud service**

****

**B.5 File transfer between cloud storage**