

# CalendarBox: A P2P File Sharing Calendar



Team 23

Chen Wang, Di Li, Darsh Shah, Jason Xi

The authors of this document hereby grant the right to release this document to future versions of 18-842 Distributed Systems course including course website and ALE, etc.

# Abstract

Combining ideas of online file hosting service and calendar sharing service, this project creates a new, light-weighted, customer friendly application with corresponding web service, called CalendarBox. CalendarBox service integrates daily usage of calendars, events sharing, file sharing, and cloud storage into one application to ease users' schedule control and file management. Specifically, CalendarBox application relates online hosting files with user's scheduled event in calendars to ease file management. It also simplifies the file sharing service by relating the sharing of files with the sharing of calendar events. CalendarBox targets those customers who have busy schedules on cooperative works in which important files are created, edited, and shared at the same time by multiple users. The backend system supporting the service is designed in Peer-to-Peer architecture with JXTA as underlying platform. Goals of security, consistency, high availability and fault tolerance are targeted through the design of CalendarBox system.

## Table of Contents

1. [Introduction](#)
  - 1.1. [Purpose of this document](#)
  - 1.2. [Scope of this document](#)
  - 1.3. [Background Information](#)
  - 1.4. [Motivation](#)
  - 1.5. [Goals](#)
  - 1.6. [Applicability](#)
2. [Overview](#)
  - 2.1. [System Brief](#)
    - 2.1.1. [Calendar Sharing Network](#)
    - 2.1.2. [Cloud File Hosting system](#)
  - 2.2. [Basic Concepts and Modules](#)
    - 2.2.1. [Rendezvous Node](#)
    - 2.2.2. [Peer Node](#)
    - 2.2.3. [Backup Node](#)
  - 2.3. [User Interface Design](#)
    - 2.3.1. [Login Window](#)
    - 2.3.2. [Password Window](#)
    - 2.3.3. [Home Window](#)
    - 2.3.4. [Calendar Add Event Window](#)
    - 2.3.5. [Event View/Modify Window](#)
    - 2.3.6. [UI of Rendezvous and Backup Nodes](#)
3. [Requirement Specification](#)
  - 3.1. [Functional Requirements](#)
    - 3.1.1. [File Availability](#)
    - 3.1.2. [Durability of File Storage](#)
    - 3.1.3. [Calendar Update and File Operations](#)
  - 3.2. [Distributed System Requirements](#)
    - 3.2.1. [Security and Privacy on Users](#)
    - 3.2.2. [Consistency of Calendar Event Operations](#)
    - 3.2.3. [Consistency of updates](#)
    - 3.2.4. [High Availability of P2P calendar system](#)
    - 3.2.5. [Fault Tolerance](#)
  - 3.3. [Development and Operating Environment](#)
    - 3.3.1. [Hardware Platform](#)
    - 3.3.2. [Software Platform](#)
    - 3.3.3. [Network Environment](#)

4. [System Design](#)
  - 4.1. [Open Source Platform ---- JXTA](#)
    - 4.1.1. [Introduction](#)
    - 4.1.2. [DOLR Interfaces](#)
    - 4.1.3. [Overview of Protocols](#)
  - 4.2. [Architecture Design](#)
    - 4.2.1. [Bootstrapping](#)
    - 4.2.2. [Peer Discovery](#)
  - 4.3. [Distributed Systems Principles](#)
    - 4.3.1. [Fault Tolerance](#)
    - 4.3.2. [Consistency](#)
    - 4.3.3. [Availability](#)
    - 4.3.4. [Security](#)
    - 4.3.5. [Scalability \(Load Balancing\)](#)
5. [Implementation](#)
  - 5.1. [Application UI Packages](#)
    - 5.1.1. [Rendezvous & Backup Nodes](#)
    - 5.1.2. [Client Node](#)
  - 5.2. [Communication Package](#)
    - 5.2.1. [Discovery Service Listener](#)
    - 5.2.2. [Pipe Message Listener](#)
  - 5.3. [Node Configurations](#)
    - 5.3.1. [CalendarBox & Backup Nodes](#)
    - 5.3.2. [Rendezvous Node](#)
  - 5.4. [Fault Tolerance](#)
    - 5.4.1. [Edge Peer Failure](#)
    - 5.4.2. [Rendezvous peer failure](#)
  - 5.5. [Consistency](#)
  - 5.6. [Availability](#)
  - 5.7. [Security](#)
  - 5.8. [Scalability \(Load Balancing\)](#)
  - 5.9. [Other imported packages](#)
6. [Demonstration](#)
  - 6.1. [Demonstration Environment](#)
  - 6.2. [Normal/Basic Communication Test](#)
  - 6.3. [Rendezvous Node Failure Test](#)
  - 6.4. [Client Node Failure Test](#)
  - 6.5. [Backup Node Failure Test](#)

- 6.6. [Consistency of Event Operations Test](#)
- 6.7. [Detailed Test Plan](#)
- 7. [Project Management](#)
  - 7.1. [Team Organization](#)
  - 7.2. [Work Breakdown/Task list](#)
  - 7.3. [Skill Challenge](#)
  - 7.4. [Final Thoughts](#)
- 8. [Reference](#)
- 9. [Acknowledgements](#)

## 1. Introduction

### 1.1 Purpose of this document

This final report provides information about the CalendarBox project, which is a Calendar events assisted P2P file sharing application. CalendarBox supports sharing of calendar events and sharing of files at the same time while it also enables users to relate shared files with shared calendar events. This document includes specifications, design goals and implementation of CalendarBox application.

### 1.2 Scope of this document

This document provides backgrounds relevant to CalendarBox and motivations behind this project. Expected goals and applicability of CalendarBox are illustrated. The distributed system serving CalendarBox users are briefed. The functionalities of CalendarBox application are defined. Modules and concepts of Distributed systems which were used in CalendarBox are explained. Implementation and demonstration are detailed. In the end, skill challenge information and team organization is mentioned.

### 1.3 Background Information

People coming from the same organization usually solve scheduling issues for meetings via calendar sharing and scheduling softwares. For example, CMU students check course schedules via the calendar system in SIO system [1]. However, such calendar sharing system usually only offers several management people working on publishing or changing events. For groups or circles not belonging to big organizations, commercial calendar sharing service like Google Calendar [2], Oracle Calendar [3] and Microsoft Outlook [4] can be used to share events with a users who also subscribe the service. For users who do not subscribe those services, emails have to be sent each by each to notify these events. However, calendar sharing usually only provides simple information like meeting topic, location, time, etc. For other resources that will be shared with those events, like a shared document that needs cooperatively writing, a software that would be developed by multiple people, or a presentation slide that will be shared by presenters, users have to refer to other applications for sharing these resources.

There is a boom of cloud-based services in recent years [5], which offers individual users and enterprises computing and storage capacity on remote data-centers. The advantages of cloud servers is that it can get rid of the complexity of hardware management for customers. As an onlooker at a gold rush of cloud storage services, we witness the births of cloud storage applications from startups and big players. From Box.com, UbuntuOne, and Dropbox to Google Drive and One Drive, a huge amount of offline files are uploaded online. Since 2007, Dropbox counts over 50 million users with totally 500 million files uploading daily [6].

### 1.4 Motivation

We all have such experiences. Before a presentation, a lot of emails need to be sent to share the presentation slides with all audiences. After a wonderful party, we stay overnight to collect all interesting pictures and copied these pictures to different friends we met in the party in following days. After a quick discussion, we always forgot to share the meeting minutes with questions and comments. Sharing events in

calendars and sharing files in another app really just increase users' burden to be closer. In this project, a calendar events assisted P2P file sharing application, called CalendarBox, is proposed. Calendar Box provides functionality of tagging shared files with shared events in calendars. If you have a regular work report meeting every week, you can just drag your reports to your meeting event and your colleagues participating the event can get your slides from their own calendars. If you give lectures to students, please share your slides in your calendar. Your students will get your slides when they add your course events in their calendars. If you have a study group to learn stuff together, drag your notes to your group learning events. Other group members will see your notes in the same time slot in calendars. Similarly you can share photos related to that calendar event.

### 1.5 Goals

The goal of this software is to provide combined service of calendar sharing and file sharing. It is also supposed to serve users who would like to use the service without sharing. For users without sharing needs, the software provides file hosting and management services according to users' schedules on their calendars. The final goal is to provide users an easy daily tool to manage daily schedules, manage hosting files and sharing events and files more conveniently and efficiently.

### 1.6 Applicability

The CalendarBox application is designed as an application running on desktop OS platforms including Mac OS X, Windows and Linux. Web based CalendarBox service is also provided to users, so user can access CalendarBox service through browsers. Local storage of files and calendars are enabled through desktop application.

## 2. Overview

### 2.1 System Brief

#### 2.1.1 Calendar Sharing Network

Our system can share calendar events with specific group of friends so that each person can easily share and receive events and get real time updates of events. This will largely reduce the cost of communication and guarantee privacy and security. When a new user registers in our service, it can add new friends in this network, create events and choose random friends to share, get real-time updates from his friends.

As to privacy, a user who creates the event is called host. A host can access all the events that was created by him or shared with him. Host can modify the details of the event. Other users are called guests. Guests receive real-time change of events that have been shared with them. For sharing files, host will add guests into events. Guests can only modify events that have been shared with them.

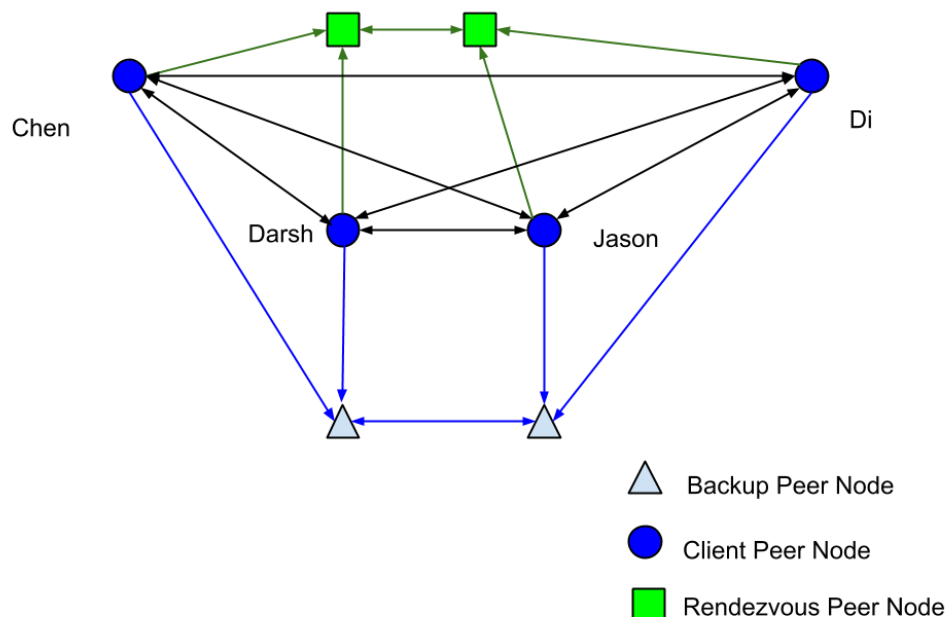


Figure 2.1.1.1 Sample P2P topology of the Calendar Sharing System

Figure 2.1.1.1 gives us an example P2P topology of our calendar sharing system. Calendar events are generated in client peers and shared among client peers directly. All event updates are pushed into backup peers in case client peers lose network connection or become offline. When a user comes online again, it will do a regular check for the missing data. It can retrieve all the messages sent from other friends from client peers directly. It should also refer to the backup server in case it requests the data this user sent



before but is not local to this machine. Every time, a user send over a new event to certain group, then the backup server is a default receiver to guarantee data availability. In this way, we do not have to rely on the single point(backup server) to implement Distributed File System.

### 2.1.2. Cloud File Hosting System

Files that are shared by client users are uploaded to backup peers. Backup peers are installed in servers hosting files as cloud storage. Multiple backup peers guarantee the high availability of files. Users who need to access files download files from cloud storage directly.

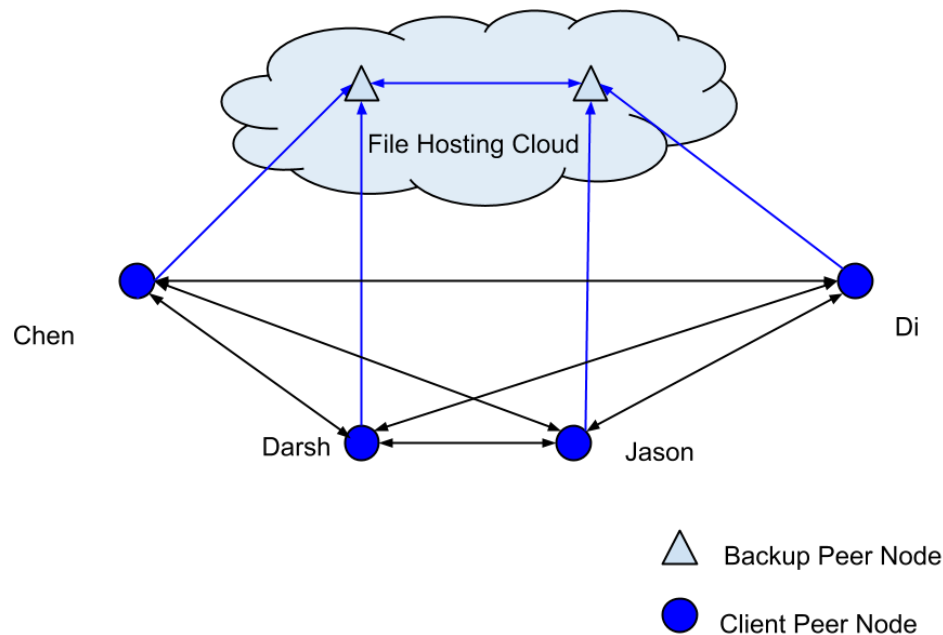


Figure 2.1.1.1 Sample topology of the Cloud File Hosting System

## 2.2 Basic concepts and modules

### 2.2.1 Rendezvous Node

The rendezvous node are used as trackers that tracks all the status of registered peers. As seen in the graph, the green nodes are rendezvous nodes and they keep record of four client peers and a backup server node. The data stored in the rendezvous node are status of the list of registered nodes including name, ip, online or offline etc. The function of rendezvous node is to track all the nodes and let each node know all others under this tracker.

### 2.2.2 Peer Node

Peer node is also referred as Client Peer. They are displayed in blue in the graph above. Peer node can be divided into two kinds. One kind is peer node behind NAT such as “Chen”, “Darsh” and “Jason” nodes. The other kind is peer node that is directly talk to other nodes such as “Di” node. When a new peer node wants to join the service, it will find a rendezvous node to register(the details will be illustrated later). The data stored in one peer node is all the events it want to share to others and all the events that shared by its friends.

### 2.2.3 Backup Node

Backup node exists due to the requirement of reliability. Its functionality is to store all the data sent inside this rendezvous group. When some node is down or restarted, losing data can be found here.

## 2.3 User Interface Design

CalendarBox application is a desktop application installed on user’s desktop. Management nodes like rendezvous nodes and backup nodes are hosted on cloud. It can be accessed from any desktop computer.

### 2.3.1. Login Window

The user comes lands up on the login screen. The user has an option to register or login to the application. User just need to type in his/her user name if he/she uses the application for the first time. he/she ever used CalendarBox before, user needs to remember his/her password set before for later input.



Figure 2.3.1.1 Login Window of the CalendarBox Application

### 2.3.2. Password Window

This password UI let users type in their password. If the user first comes to CalendarBox service, he/she just needs to create a new password to log in. If the user name new user choose is duplicated with existing user, the password input will not be the same as what has been recorded by the login server. Then the new coming user needs to choose a new user name. With the same username and password, returning users can retrieve all events and related files by re-login.

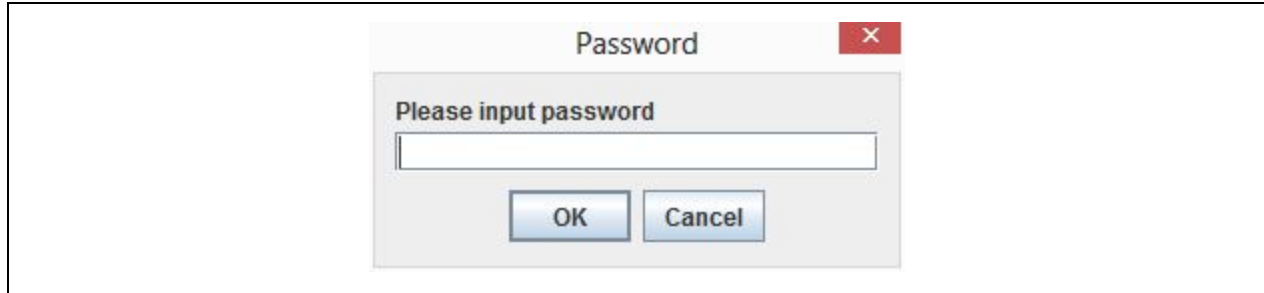


Figure 2.3.2.1 Password Window for CalendarBox service

### 2.3.3 Home Window

When a user input his/her correct username and password, CalendarBox application will bounce out a dialog showing that the user is connecting to a rendezvous server successfully. After connecting to rendezvous successfully, the CalendarBox application running on client machine runs as a client peer node in the P2P calendar sharing system. In this UI, there are three buttons, Add Event, Modify/View Event and Cancel. Add Event will allow user to add a new event. Modify/View button will show all events that have been shared with current user and allow user to modify these events. Cancel will allow user to exit CalendarBox application.

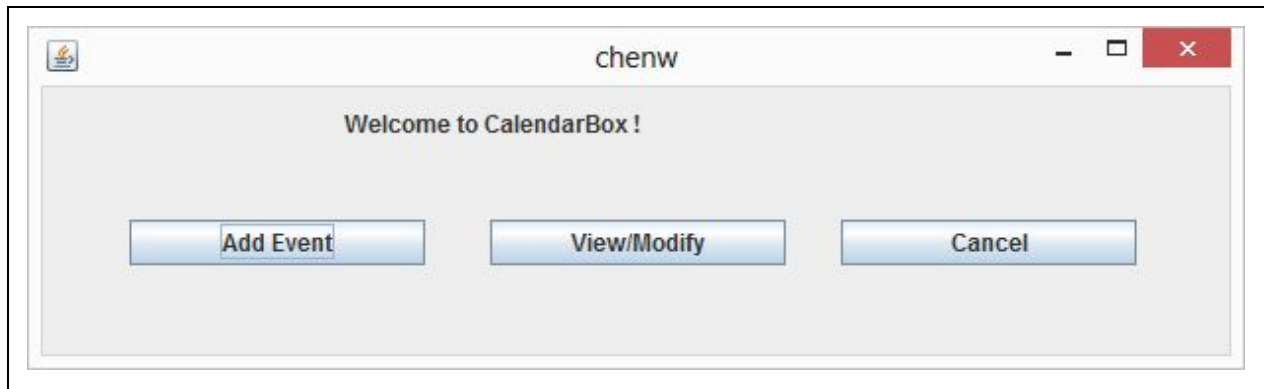


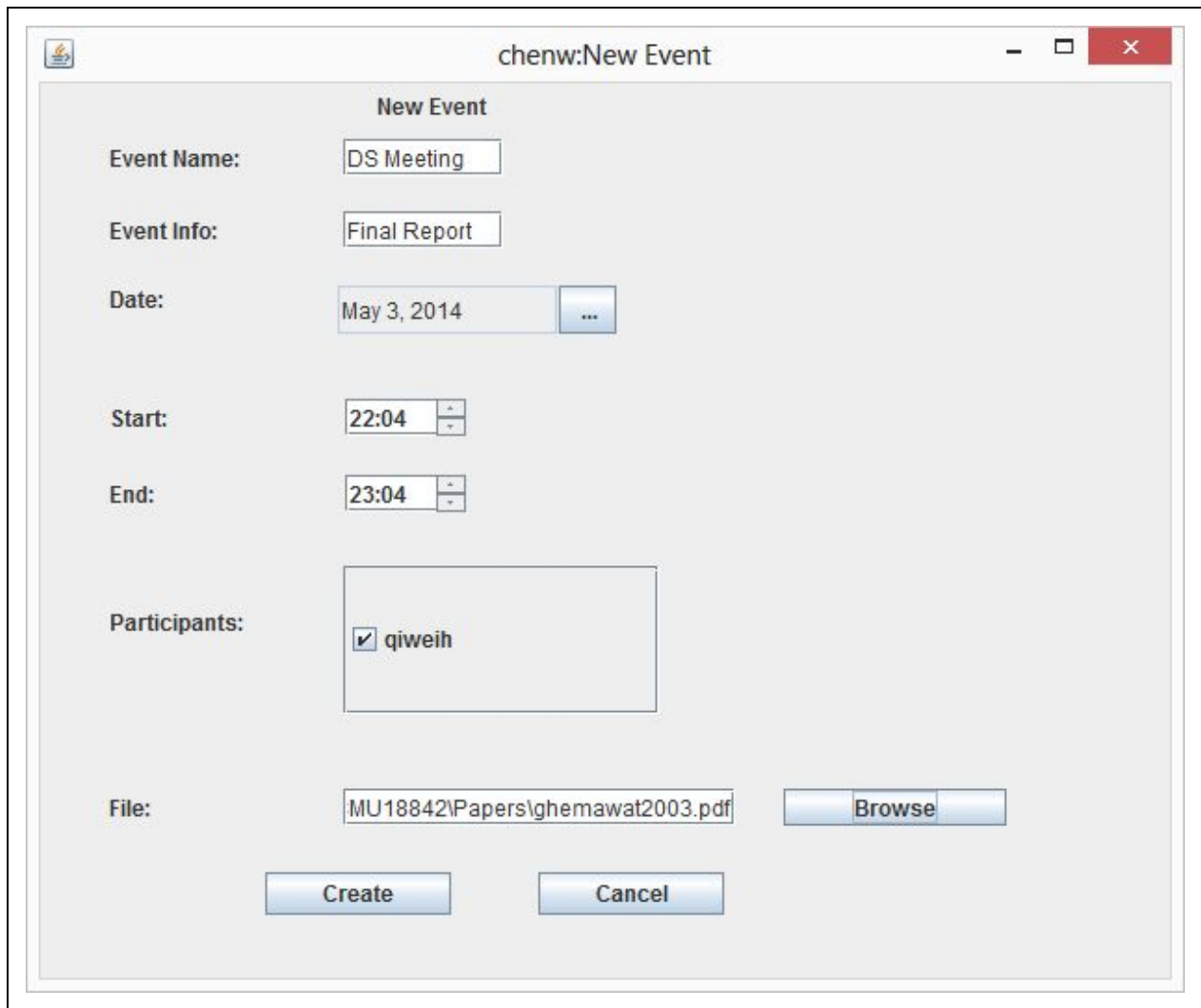
Figure 2.3.3.1 User's home Window in CalendarBox application

This UI will help the user to keep track of this appointments. He will also get to know if some person invited him to the meeting. Also, the user can download files of the upcoming event so that they can be used later on.

### 2.3.4 Calendar Add Event Window

When a user click "Add Event" from the home UI, the following New Event UI will bounce out. "chenw" is the sample user name. User can edit "Event Name", add "Event Info", choose "Date" and "Time" for a meeting. The participants list will show all users so the user can choose anyone who also uses

the application to share a event. File uploading frame allows user to upload any files related with this event on to the file hosting cloud. Other users who have been shared with this event will have access to download the file. User will click “Create” to create this event and share with chosen friend.



The screenshot shows a window titled "chenw:New Event" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, the title "New Event" is centered at the top. Below the title, there are several input fields and controls:

- Event Name:** A text box containing "DS Meeting".
- Event Info:** A text box containing "Final Report".
- Date:** A date picker showing "May 3, 2014" with a small calendar icon to its right.
- Start:** A time picker showing "22:04" with up and down arrows.
- End:** A time picker showing "23:04" with up and down arrows.
- Participants:** A list box containing a single entry "qiweih" with a checked checkbox to its left.
- File:** A text box containing "MU18842\Papers\ghemawat2003.pdf" and a "Browse" button to its right.

At the bottom of the dialog, there are two buttons: "Create" and "Cancel".

Figure 2.3.4.1 Add Event UI in CalendarBox Application

If the event has been created and shared with other users successfully, following dialog will bounce out.

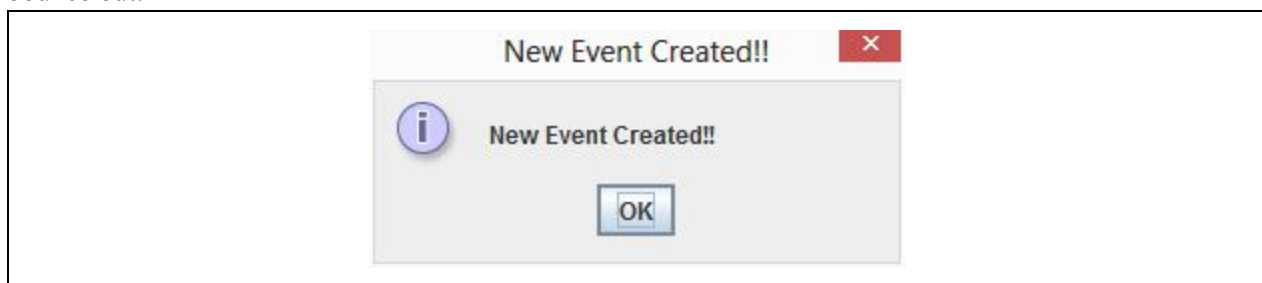


Figure 2.3.4.2 Event Added Successfully Dialogue

### 2.3.5 Event View/Modify Window

After adding some events into a user's calendar, he/she may want to view, modify or delete existing events. For example, user "chenw" just created an event for a DS meeting and shared with user "qiweih". However, at the same time, user "qiweih" created another event for exam preparation and shared with user "chenw". Then user "qiweih" wants to check if there is schedule conflicts. He can click "view/modify" button on the home window and jump to "Modify Event" window as shown in Figure 2.3.5.1. As it can be seen, both of these events will show in user "qiweih" 's window, so he may select one of the conflicted event and delete it using delete button. In alternative, he can also select one of the events and click "Modify" button to modify the start and end time of the event.

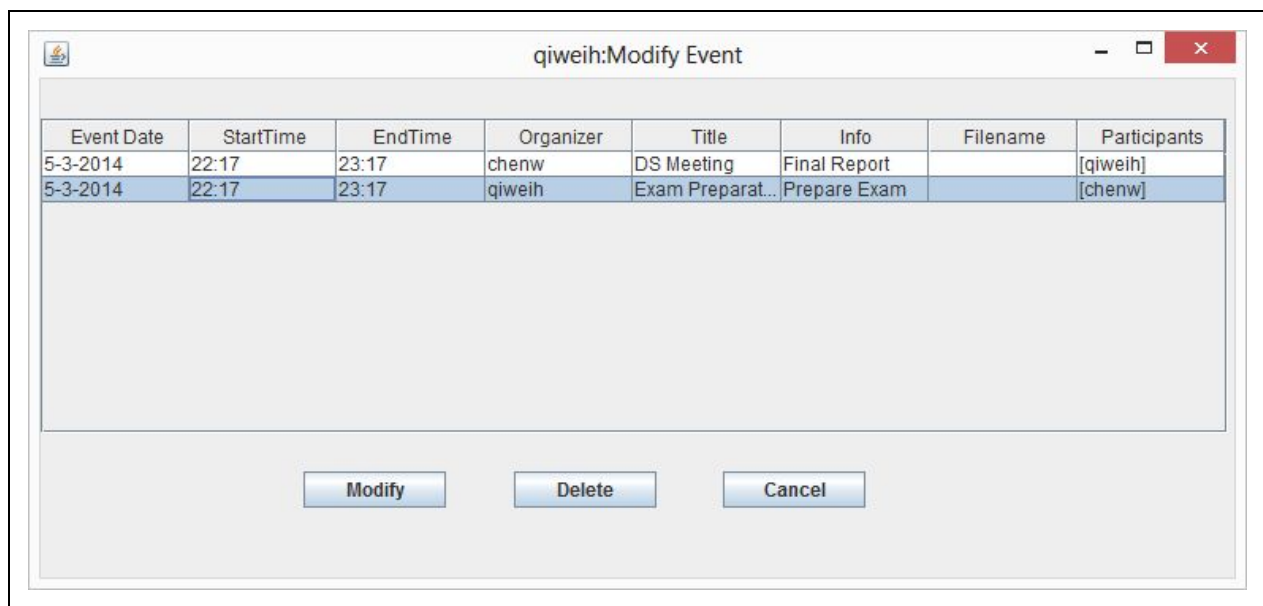


Figure 2.3.5.1 Event View/Modify Window in CalendarBox Application

The Modify Event window then comes out for user to modify some attributes of an event. The Modify Event Window looks similar as the Add Event Window. When the user finishes modifying attributes of the event, a dialog will show that the event has been successfully modified.

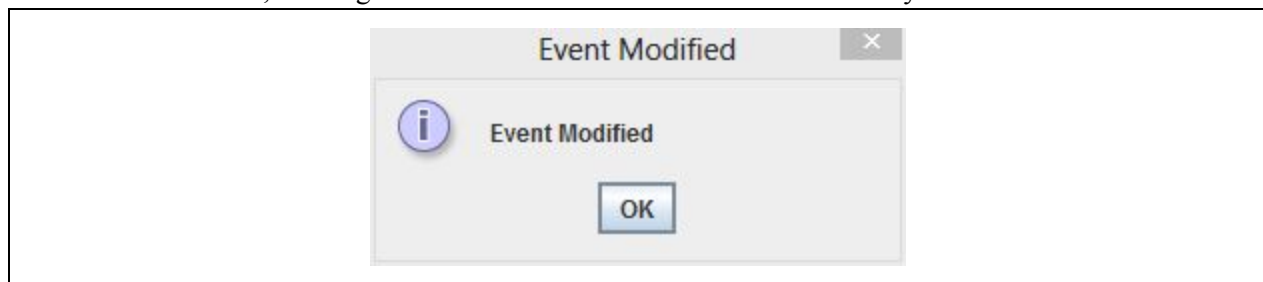


Figure 2.3.5.2 Event View/Modify Window in CalendarBox Application

### 2.3.6 UI of Rendezvous and Backup Nodes

Rendezvous and backup nodes are supposed to start before CalendarBox service starts. Though there is no need to design UI for these nodes, we add some simple dialog windows to show whether they are working normally. Figure 2.3.6.1 comes out whenever the rendezvous starts working.

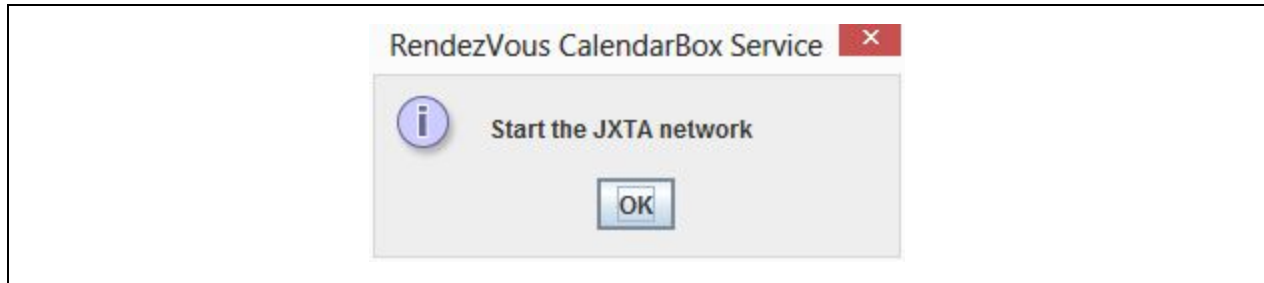


Figure 2.3.6.1 Rendezvous Start Window

When the rendezvous node starts successfully, the dialog in Figure 2.3.6.2 will bounce out showing that rendezvous is waiting for other peers to connect to it.

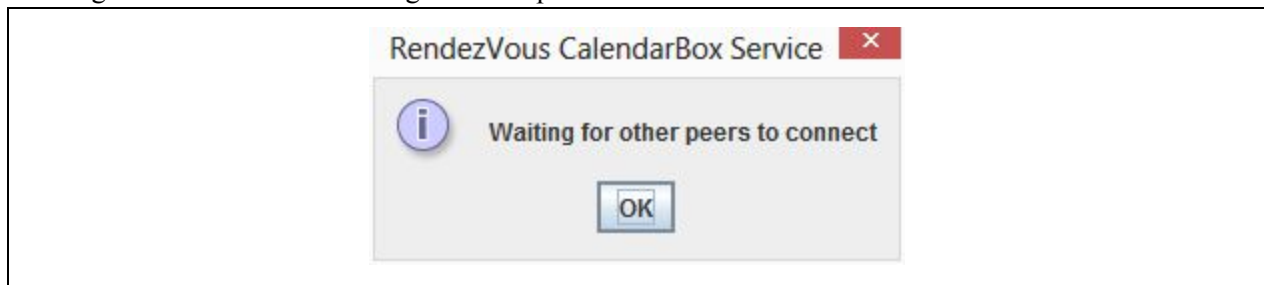


Figure 2.3.6.2 Rendezvous ReadyWindow

Similarly as rendezvous node, the backup node has some simple UI to show whether it is functioning normally. When the backup node starts, it first needs to connect to a rendezvous node to connect to P2P network. The dialog in Figure 2.3.6.3 will come out to show it is ready to connect to rendezvous node.

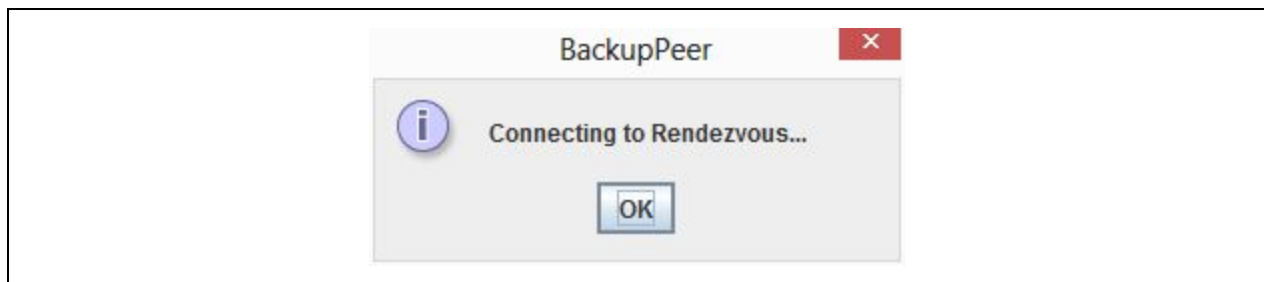


Figure 2.3.6.3 Backup Node Starting Window

After connecting to rendezvous node, there would be a Message window coming out showing that the backup node is running. When the backup running message comes out, it says the backup node is functioning normally and is backing up all the events and files now. Clicking Ok twice will stop the backup node.

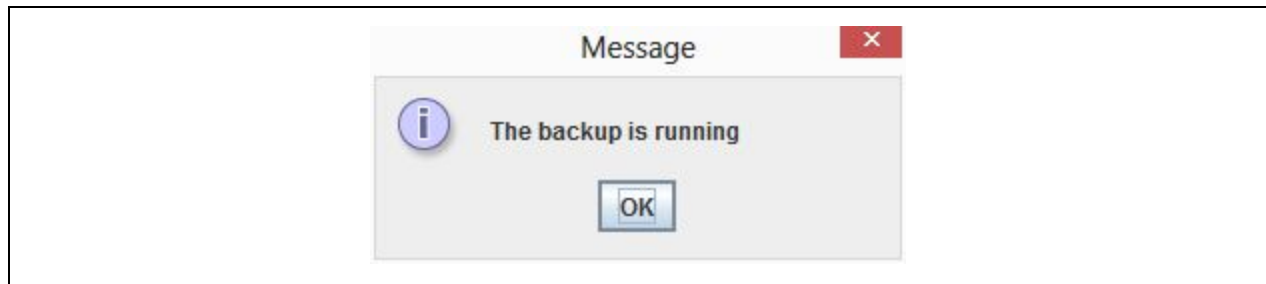


Figure 2.3.6.4 Backup Node Running Window

### 3. Requirement Specification

#### 3.1 Functional Requirements

##### 3.1.1 File Availability

In our system, an existent file or update can always be retrieved when a peer gets online. When files or updates are created from an online peer, they can be retrieved by sending query messages to the online peer in his group to get their most recent version; When files or updates are from an offline peer, they can always be retrieved from their corresponding backup server, and if backup peer is crashed, it waited until his friend being online again and all the files and updates would go to a new backup server. For a backup server, it will store all the files for the group of node that under control of current tracker (rendezvous peer).

##### 3.1.2 Durability of File Storage

In our design now, no matter whether the peer is online or not, the file is always available. And a file can only be deleted and uploaded from user's interface. For a file, it is immutable when it is stored, which is similar to the dfs, and it can only be changed by directing to a new storage block. Since the file is now hosted in Cloud Service, it is guaranteed that the file hosting service is durable.

##### 3.1.3 Calendar Update and File Operations

- When a calendar creator create, add, edit or delete a shared event, we push the updates to everyone in our current group.
- And if a peer is not online, it will always pull the updates of events before the current peer do any operation on anything. Therefore, we forbid offline editing in current design, a peer cannot operate on the calendar when he logs off.
- Backup keeps the events in the latest version. When a peer gets back online, it gets the newer version of events from the backup server.

#### 3.2 Distributed System Requirements

##### 3.2.1 Security and Privacy on Users

Users can log in to the application from different devices. Multiple users can share the same device as well. Therefore, there needs some access control for each user. In addition, one user should be able to access all his/her calendar events when he/she login Calendar Events in a new device. Therefore, there should be some login service for each user. Every new user should be able to set their username and password. There should be a mechanism to guarantee there is no duplicated usernames. Furthermore, user should be able to set his/her own password to protect his/her privacy on CalendarBox. Other users should not be able to access these calendar events unless they know both the username and password.



### 3.2.2 Consistency of Calendar Event Operations

For the isolation property, it will ensure that the concurrent execution of results of users' operations in a system state that would be obtained if any user's operations were executed serially, that is, one after the other. In this way, we can achieve the goal of concurrency control, which is demanded in the system design. And the effects of an incomplete user's operation might not even be visible to another one.

### 3.2.3 Consistency of updates

The consistency property ensures that any updates will bring the system state from one valid state to another. For the update of calendar, an update would go to all group members, otherwise, no one get an update if the other do not get an update either.

### 3.2.4 High Availability of P2P calendar system

High availability is a system design approach and associated service implementation that ensures a prearranged level of operational performance will be met during a contractual measurement period. As what illustrated before, an existent file or update can always been retrieved when a peer gets online. This is the characteristic of distributed file system.

### 3.2.5 Fault Tolerance

A fault in a system is some deviation from the expected behavior of the system: a malfunction. Faults may be due to a variety of factors, including hardware failure, software bugs, operator error, and network problems. And fault tolerance refers to a system's ability to deal with malfunctions. The general approach to building fault tolerant systems is redundancy. Redundancy may be applied at several levels. Information redundancy seeks to provide fault tolerance through replicating the data. For example, we can make several copies of the files in different place. Time redundancy achieves fault tolerance by performing an operation several times. Timeouts and retransmissions in reliable point-to-point and group communication are examples of time redundancy. An example is TCP/IP's retransmission of packets. Physical redundancy deals with devices, not data. We add extra equipment to enable the system to tolerate the loss of some failed components. For example, we have redundant servers to avoid single point of failure. In system design part, we will talk about fault tolerance strategies for different node failure.

## 3.3 Development and Operating Environment

### 3.3.1 Hardware Platform

In this project, two types of physical nodes are considered, client peers on users' devices and management peers on servers. Management peers include rendezvous nodes and backup nodes. These nodes are supposed to be available all the time and should seldom lose connection to Internet. We used ECE cluster servers are considered to ensure the availability of Calendar box services.

ECE hosting server [7] has a public domain name with global identifiable ip address.

The ECE cluster is a group of linux based remote access machines that CMU ECE students can use for general computing. These do not have console access. Again, any student that has an andrew or ece account also has access to all the machines in the ECE Cluster. Machines follow the naming convention of ece000.ece.cmu.edu through ece008.ece.cmu.edu that allow off campus ssh access to the cluster [15].

### 3.3.2 Software Platform

For desktop application, JAVA is used to develop CalendarBox application. Java is chosen because its portability, which means that computer programs written in the Java language must run similarly on any hardware/operating-system platform. Java Runtime Environment (JRE) are required to be installed on client peers.

For the frontend, we will use Twitter Bootstrap framework which has HTML/CSS inbuilt. For the initial prototype of the web service, java servlets will be used. Later on we will be migrating to either Spring or Play framework. Both of them are in Java. For database, mysql will be used initially. Later on, we will migrate to HBase (nosql) if it provides better performance.

### 3.3.3 Network Environment

Various peer nodes are supposed to be deployed in two continents, possibly in more than three geographical locations. Client peers would be deployed in Pittsburgh, Silicon Valley and Porto, Portugal. Communications are supposed to cross intercontinental cables and US backbone cables. There would be large propagation delay between peers. Network congestion in peak hours might occur. System is deployed in real, global-wide, Internet environment.

## 4. System Design

### 4.1 Open Source Platform ---- JXTA

#### 4.1.1 Introduction

JXTA, also named as Juxtapose, is an open source peer-to-peer protocol specification [9]. Sun Microsystems started the project in 2001. The JXTA protocols are defined as a set of XML messages which allow any device connected to a network to exchange messages and collaborate independently of the underlying network topology.

JXTA protocols can be implemented in any modern computer language. Implementations are currently available for Java, C/C++ and C#. JXSE is the standard edition of Open Source Java implementation for the JXTA protocols. Programmer's guides and source code are provided in [10].

JXTA strongly assembles Tapestry that is an implementation of P2P system based on distributed hash table. Tapestry [11] routes messages to nodes based on GUIDs associated with resources using prefix routing in a manner similar to Pastry. However, Tapestry conceals the distributed hash table from applications behind Distributed Object Location and Routing (DOLR).

#### 4.1.2 DOLR Interfaces

Basic programming interfaces have following forms and there are similar interfaces in JXSE platform. The abstract representations of DOLR are proposed by [12] in 2003 and they can be shown as follows.

- *publish(GUID)* : GUID of object, in this software objects can be a file, a calendar event etc, is generated by 160 bit SHA-1 [13]. This publish function makes the node performing a publish operation on the host for the object corresponding to its GUID.
- *unpublish(GUID)* : This unpublish function do an unpublish operation to make the object corresponding to GUID inaccessible.
- *sendToObj(msg, GUID, [n])*: This interface defines an operation that sends message to object with GUID for various purpose, like editing, query of accessing, or deleting etc. n parameter, if exists, denotes that the message would be reliably sent to n replicas of the object.

#### 4.1.3 Overview of Protocols

The JXTA specifications defines a small number of protocols for the basic setup and communications in the peer-to-peer system. Currently, there are six protocols defined and their main task are briefly described as follows.

- Peer Resolver Protocol (PRP)

PRP is a protocol that defines the sending of a query messages to any number of other peers. PRP also defines the receipt of a response.

- Peer Discovery Protocol (PDP)

Peers use the JXTA PDP to discover JXTA resources dynamically. The PDP defines the lowest-level technique that is available for peers to discover resources. A peer using PDP will discover all JXTA resources (all other peers including relay peers, backup peers and client peers) that are on the local network or are known by the peer's rendezvous peers.

- Peer Information Protocol (PIP)

The peer information protocol is a implementation of a peer resolver query. When a peer is first initialized, it publishes its *PeerAdvertisement* that is picked up by at least one rendezvous. When a request for a specific peer's information is requested, a query is made to locate the *PeerAdvertisement*. Recalling that the *PeerAdvertisement* contains endpoint information for contacting the peer, so by using the endpoint, the peer is contacted directly to obtain its peer information, such that the peer status information can be obtained.

- Pipe Binding Protocol (PBP)

The pipe binding protocol defines the connection setup between peers and how information are sent between peers. PBP is used to create communication path.

- Peer Endpoint Protocol (PEP)

Endpoint routing is used to enable pipes or simplistic messaging, such as that found in the peer info, peer resolver, and pipe binding protocols, so PEP is used to find a route from one peer to another.

- Rendezvous Protocol (RVP)

RVP is used to propagate messages and manage all other peers in the network. Its functionalities in the system will further be illustrated in Section 4.2.

## 4.2 Architecture Design

### 4.2.1 bootstrapping

The JXTA network uses an universal resource binding mechanism called the resolver to perform all the resolution found in a traditional distributed system, such as resolving a peer name into IP address, binding a socket into a port or searching content in NFS. In JXTA, all resolution operations are unified under the simple discovery of one or or advertisements. JXTA protocols do not specify how the search of advertisements is performed, but provide a generic resolver protocol framework with a default policy that can be overwritten. We can tailor their resolver implementations to use a decentralized, centralized or hybrid approach to propagate queries and receive responses. The resolver performs authentication and verification of credentials, and drops invalid messages.

#### 4.2.2 Peer Discovery

Each peers first check in their local cache for any rendezvous advertisements. Rendezvous advertisements persist across edge peers connections. The edge peer orders candidate rendezvous in batch of five and try them five at a time until a rendezvous connection is established. Each edge peer only maintain one connection. If not a rendezvous connection is established after a tunable period(30s), the edge peer will attempt to search for rendezvous via a propagate request on its available transport or on peer group specific pipe if the edge peer has joined a peer group. Existing rendezvous peers are listening, and will reply to the request. If after an extended period, no rendezvous has been found, the edge peer will query one of the seeding rendezvous. Finally, if none of the seeding rendezvous is reachable after a period, the edge peer will become a rendezvous. The edge peer can return into an edge peer mode as soon as a rendezvous is found. The peer will continuously search for rendezvous in the background.

### 4.3 Distributed System Principles

#### 4.3.1 Fault Tolerance

The application will be made robust and fault tolerant. Below paragraphs discuss various points of failure and methods of recovery in those scenarios.

##### 1. Peer Failure

Peer Failure occurs when the endpoint client system crashes. Since the application doesn't have any control on the user's computing device, a peer failure is unrecoverable. The data is still saved on the backup peer. So, when the peer comes back online using a different machine, he can see the same state as before the crash.

##### 2. Backup Node Failure

At any point of time in the deployed system, there is only one backup node. The backup node is an invisible participant in all the events and so it has all the data in the system. When the backup node fails, we can start a new backup node and upon startup, its syncs all the existing data from all the nodes.

##### 3. Rendezvous Node Failure

At any point of time in the deployed system, there will be two replicated rendezvous nodes. The ip addresses of both the servers are included in the application. So if one of the rendezvous fails, then the other rendezvous will take over. Also, the system could detect the failure of rendezvous node and will launch a new rendezvous node. The system also works without any rendezvous as rendezvous node is just used for bootstrapping.

#### 4.3.2 Consistency

All peer nodes in the system will have a consistent view of the system. We are aiming for one

copy semantics where each node sees same state of the events. Hence, if one node tries to modify or delete the event, it first checks whether any other participant is currently modifying it. If not, then that peer node will be allowed to modify or delete that event. Hence a consistent view is maintained.

#### 4.3.3 Availability

Since the system has a backup node, all the data will be saved in the backup node. Hence, even if a node goes down, the other nodes can get the data/event from the backup. If the backup goes down, then the node can get data from other nodes. Also, the system works without rendezvous node if bootstrapping is done. Hence, even the all rendezvous nodes fail, the system is still available.

#### 4.3.4 Security

The application requires username and password to join the system. The username-password are stored in a sqlite database on a django server. Hence, no user can see any other users data. Also, user can login from any machine since each user is identified via a username and password.

#### 4.3.5 Scalability (Load Balancing)

For a system to be scalable, it should support many users. In our application, we can create any number of rendezvous nodes. The peer node randomly selects one rendezvous node to connect to. The rendezvous nodes communicate among themselves and share a list of connected peers whenever a new peer gets connected. So, in this way, load is balanced among rendezvous nodes and the system can scale.

## 5. Implementation

The whole system is implemented in JAVA using java library version 7 and JXSE version 2.6. In addition, a lot of other libraries in JAVA are also imported to support JXSE library and UI design. Detailed packages are listed in 5.3. The framework of code design can be described in Figure 5.0. The whole system includes two main parts of code that are included in all nodes' code, UI package and communication package. For each different types of nodes, according to their functionalities, various packages are implemented as described below.

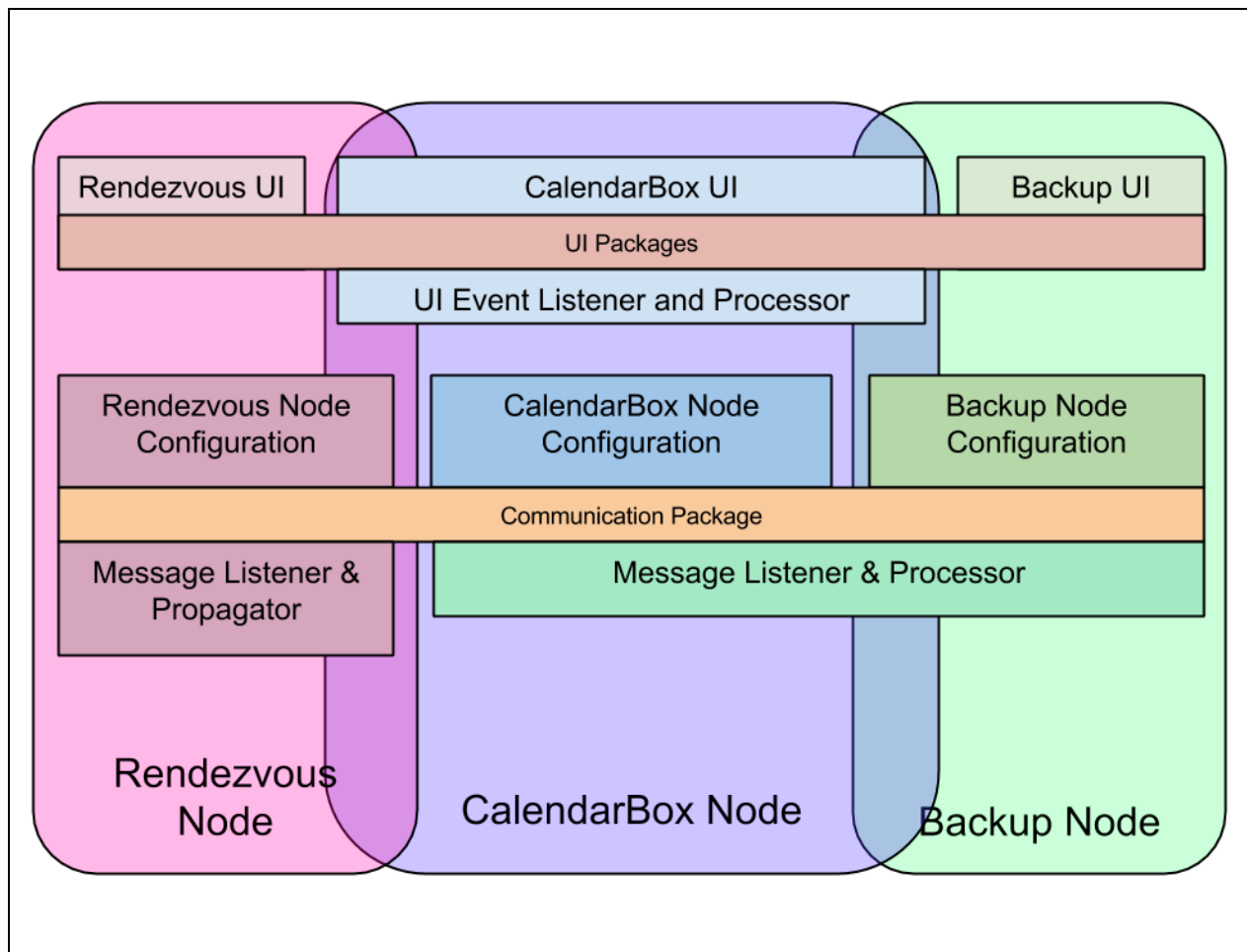


Figure 5.0 Implementation Framework

### 5.1 Application UI packages

Windows and dialogs are created to show messages and allow users to operate CalendarBox application. As we introduced above, CalendarBox UI allows users to add, view, modify and delete events. For Rendezvous & Backup Nodes, there are only some simple

information window coming out showing running progress. For Client Node, UI design must consider the functionalities allowed for user operation. We in the following, briefly introduce UI package implemented in CalendarBox.

#### 5.1.1 Rendezvous & Backup Nodes

All the UI in Rendezvous and Backup Nodes are written in package CalendarBox.Tools. It includes popping out all connected peers for a rendezvous node, popping up all connected rendezvous node for a backup or client node, popping up error, warning information messages as necessary. If the node requires some input, PopInputMessage would help. To verify again with user's operation, PopYesNoQuestion can be used.

##### → CalendarBox.Tools

- ◆ popConnectedPeers
- ◆ popConnectedRendezvous
- ◆ PopErrorMessage
- ◆ PopInformationMessage
- ◆ PopInputMessage
- ◆ PopWarningMessage
- ◆ PopYesNoQuestion

#### 5.1.2 Client Node

For client node UI, UI packages are programmed in GUI.CalendarBoxUI, GUI.AddEvent and GUI.ModifyEvent. GUI.CalendarBoxUI creates the home window for CalendarBox application running on client desktop. "initialize()" function initializes the home window and buttons. "callnewjframe()" method is called whenever user wants to add a new event. It will create a new window called "Add Event" window to ease user's operation on creating a new event. "modifyeventjframe()" is called when user click "Modify/View" button to modify or view existing events. Corresponding window will be generated for user to modify and view existing events. "cancelEvent()" and "cancelModifyEvent()" will be called to close these windows. For detailed implementation, "Add Event" window is programmed in GUI.AddEvent package and is instantiated in callnewjfram() in GUI.CalendarBoxUI package. Similarly, GUI.ModifyEvent package is programed for "Modify/View Event" window and is instantiated in "modifyeventjfram()" in GUI.CalendarBoxUI. Basic operations like, selecting files, send and update events, modify and delete events are included in GUI packages as well.

##### → GUI.CalenarBoxUI

- ◆ initialize();
- ◆ callnewjframe();
  - getEventData();



- cancelEvent();
- ◆ modifyeventjframe();
  - getModifiedEventData();
  - cancelModifyEvent();
- GUI.AddEvent
  - ◆ constructor;
  - ◆ initialize();
  - ◆ selectFile(ActionEvent e);
  - ◆ getAndSendEvent();
  - ◆ ModifyAndSendEvent();
- GUI.ModifyEvent
  - ◆ constructor;
  - ◆ initialize();
  - ◆ deleteRow();
  - ◆ deleteList(int sel);

## 5.2 Communication Package

Communication package is implemented to discover peers and exchange messages among different peers. It is used by all three types of peers. The communication package implements listeners from JXSE like DiscoveryListener and PipeMsgListener. It includes several facilities to discover, get, and process events. It builds up the foundation of CalendarBox communication.

### 5.2.1 Discovery Service Listener

The discovery service listener is the default listener in jxse for discovering all types of Advertisement (Peers, Groups, Pipes, Modules, etc.). `discoveryEvent(DiscoveryEvent event)` method is the default method that will get the “event” from the discovery service listener. In our implementation, we read the message obtained from the event, analyze the source of the message and use “`send_to_peer`” method to send back a “NOTIFY” message.

- `calendarBox.communication`
  - ◆ constructor;
  - ◆ `start()`;
  - ◆ `discoveryEvent(DiscoveryEvent event)`;
  - ◆ `send_to_peer(CalendarMessage message, PeerID found_peer_id)`;

### 5.2.2 Pipe Message Listener

The pipe message listener is the default listener for messages sent to the peer. `pipeMsgEvent(PipMsgEvent event)` will grab all message events out from pipe message listener and obtain the message from “event”. After obtaining message from `pipeMsgEvent()`, the

message is passed to processRcvMsg() that acts as a message processor. Basically, it will pass the add event and modify event messages to UI for further operations. All notify messages are processed and acked. Details of message processor will be illustrated in 5.4.

→ calendarBox.communication

- ◆ constructor;
- ◆ start();
- ◆ pipeMsgEvent(PipeMsgEvent event)
- ◆ processRcvMsg(String peerNameStr, CalendarMessage msg, String from);

### 5.3 Node Configurations

#### 5.3.1 CalendarBox & Backup Node

Configurations of nodes are done in package CalendarService, CalendarEdge for client node and CalendarBackup for backup node. They both have constructors and config methods for basic configurations like, configuring rendezvous nodes, setting up with username, and enabling tcp connections coming in and out, etc. To provide an interface to communication layer for convenience of sending and receiving messages, sendOutNewEvent() and checkForNewEvents() methods are added in node configurations. Since backup node is assumed to be always online, only a sendOutNewEvent() method is available in backup node.

→ CalendarService.CalendarEdge

- ◆ Constructor
- ◆ configEdge();
- ◆ sendOutNewEvent();
- ◆ checkForNewEvents();

→ CalendarService.CalendarBackup

- ◆ Constructor
- ◆ configBackUp();
- ◆ sendOutNewEvent(Event newEvent, String kind);

#### 5.3.2 Rendezvous Node

main() function in package CalendarService.RendezVousTest does all configuration needed in rendezvous node. It also starts and stops rendezvous according to UI actions.

→ CalendarService.RendezVousTest

- ◆ main()

### 5.4 Fault Tolerance

### 5.4.1 Edge peer failure

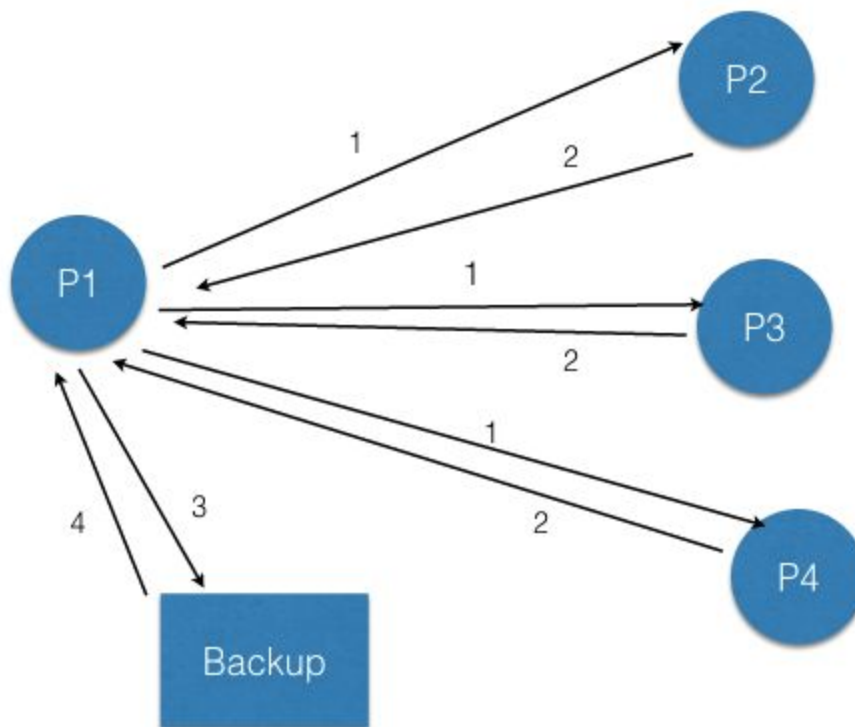


Figure 5.4.1 Edge Peer Failure

1: When P1 online again, multicast “**GETFILE**” message to all of his “friends”

1.5: When a peer receive “GETFILE” message, it will check all local events to see whether it has P1 as organizer or participant. If it is, it will append to a list.

2: Other edge peers will send back this list in a message with kind “**GETRESPOND**”.

2.5: P1 will check the received events and eliminate duplicates before adding them into local events list.

3:P1 will send list of pairs <Organizer, SeqNum> which can be regarded as the list of unique ids of received events to backup peer. This list is the message body while the kind is “**CHECKUPDATE**”.

3.5: Backup will compose all the events that P1 is involved in but are missed by 1st round recovery into a list.

4: Backup send this list in the message with the kind of “**REUPDATE**”.

The advantage of this implementation is:

First, we will retrieve events from other online peers instead of backup peer directly. So we can take full advantage of p2p property to avoid single point failure of backup peer. Only when all the peers holding the event are offline, we will have to bother backup for files which is surely an extreme situation.

Second, When we have to refer to backup peer for possible missing events in the 2nd round, we do not have to send out the whole event list. We only send out the list of unique ids(<Organizer, Seqnum> pair) which will efficiently reduce the load of network.

The disadvantage of this implementation is:

In the 1st round recovery, P1 will receive lots of repetitive events from others. This will consume too much network resource since same event will be sent back several times from different peers to P1.

#### 5.4.2 Rendezvous peer failure

Multiple rendezvous nodes are started in our P2P sharing network for CalendarBox service. Underline JXTA communications on rendezvous nodes can let each rendezvous node discover all other rendezvous nodes and propagate all types of messages to edge peers connecting to other rendezvous nodes. Multiple rendezvous nodes can tolerate rendezvous peer failure in following cases.

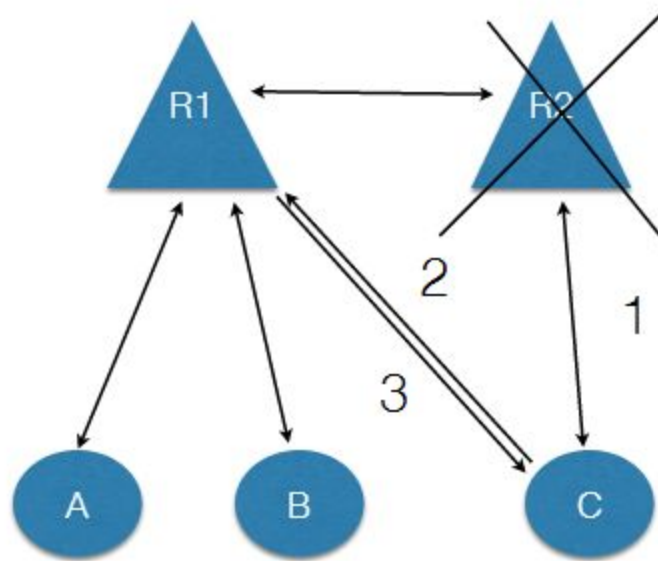


Figure 5.4.1 Rendezvous Failure

Case 1: A new coming client peer lose connection to a rendezvous node on its list.

Assuming there are two rendezvous nodes R1 and R2, and three client nodes A, B and C, client A and B randomly choose R1 as their rendezvous peer and client C choose R2.

- 1) Client C tries to connect to R2 and wait for 1 minute for the new connection. However, C's connection to R2 fails. It can be various causes, C lose network connection to R2, R2 failure, or R2 lose internet connection.
- 2) Anyway, client C gets a timeout when it connects to R2. It deletes R2 from its list and checks the availability of other rendezvous nodes. Then, client C finds R1 and tries to set up connection with R1.
- 3) R1 sends advertisement message to client C and they set up a rendezvous connection.

Case 2: A rendezvous node with connected peer failed suddenly.

Assuming there are two rendezvous nodes R1 and R2, and three client nodes A, B and C, client A and B have connected to R1 and client C has connected R2.

- 1) There is an advertisement message communicating between rendezvous node and edge peer nodes every 10 second. It functions like a heartbeat message between rendezvous node and client nodes.
- 2) When rendezvous node R2 fails, it stops sending heartbeat message to client C.
- 3) Client C regularly checks the number of advertisements received every 10 seconds.
- 4) If there is no message coming to client C for a 10 second period, client C tries to stop service and reconnect to the default rendezvous node.
- 5) If the reconnection fails, client C will follow the steps in case 1 to find a new rendezvous node to connect to.

## 5.5 Consistency

All peer nodes in the system will have a consistent view of the system. We are aiming for one copy semantics where each node sees same state of the events. So we specify a boolean field "isOK" in the class Event. If it is true, it represents no one wants to modify/delete. If it is false, it represents someone else is modifying/deleting this event.

Workflow:

1. When A wants to modify/delete an event, it will check whether the field "isOK" is true locally. If not, this operation is not allowed.

2. If it is true, A will multicast “**ASKFOROK**” message to all participants of this event. A wants to check whether others are doing modify/delete.
3. Other peers will send back “**RESPONDOK**” message with true or false to A.
4. A will check whether others are doing modify/delete based on returned messages. If it is, we will give popup to users showing you cannot modify/delete right now. If not, it will modify isOK to false which is like holding the lock and enter to modify/delete. After that, A will recovery the isOK flag to true which is like releasing lock.

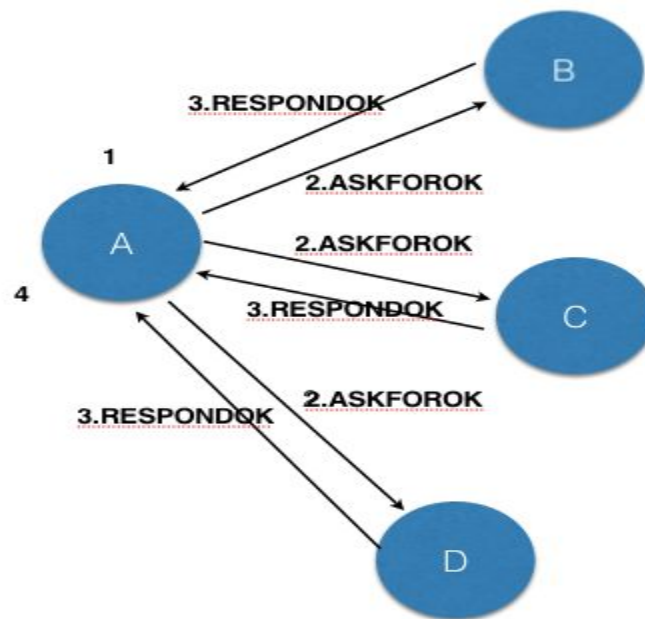


Figure 5.5.1 Consistency Control

Figure 5.5.1 shows the “Cannot modify” window that locks the modify and delete operations.

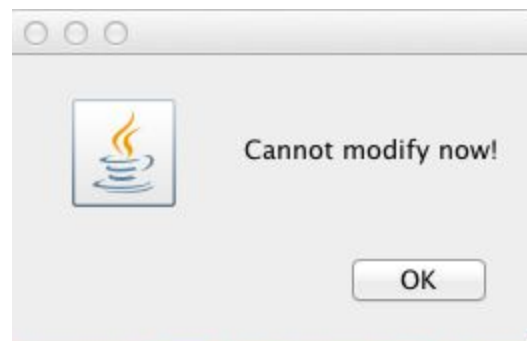


Figure 5.5.1 Cannot Modify Window

## 5.6 Availability

### a. Rendezvous availability

In the project, we hardcoded a list of rendezvous peers. When the new edge peer becomes online and wants to connect to rendezvous. It will try the rendezvous one by one until it find a available rendezvous peer. Only when all rendezvous are offline, the edge peer will give up joining p2p network and exit directly.

### b. Backup availability

Now we only have one backup in the network. It is better to have more than one in reality. But we will not demo it here. The mechanism is the same. We can have a list of backups with each hold same copies. When we find one is down, we can refer to another one in the backup peer list.

Another thing is that we can still handle faults while the backup peer is down. The current implementation is we will refer to other edge peers first. So in most situations, the unavailability of backup peer will not affect anything.

## 5.7 Security

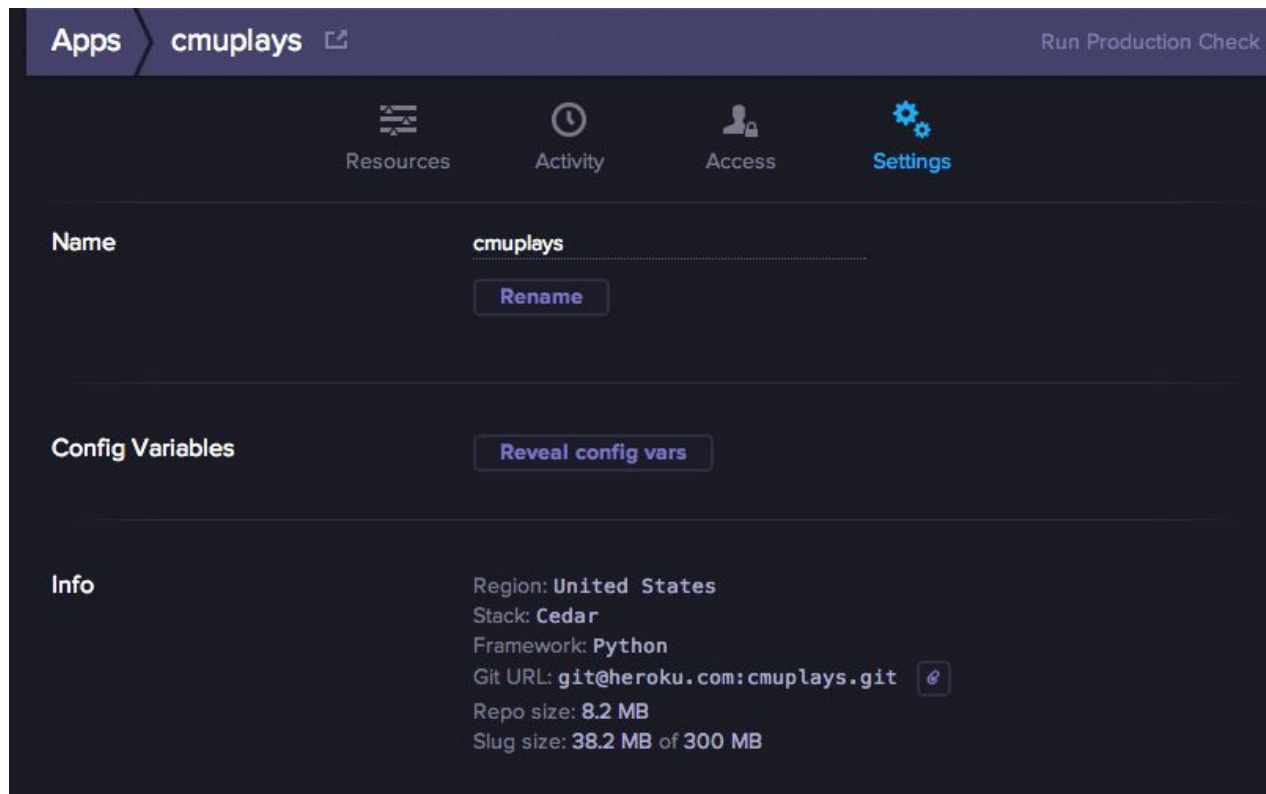


Figure 5.7.1 Django Registry Server

In order to address security and authentication problems, we deploy a django app on heroku. This web server will only be responsible for user registry and user authentication. We run sqlite as the database. It will store all the usernames and passwords in the database. When you open our app, you will type in your username and password.

If you are a new user, our server will register and save user data in the database when your username is not existed in the database. If you are a old user, our server will authenticate your provided username and password in the database to see whether it is a match.

Certainly, we will transmit and save your password in an encoded way which is handled by django internal models. So you do not have to worry about this. It is not plain text.

### 5.8 Scalability (Load Balancing)

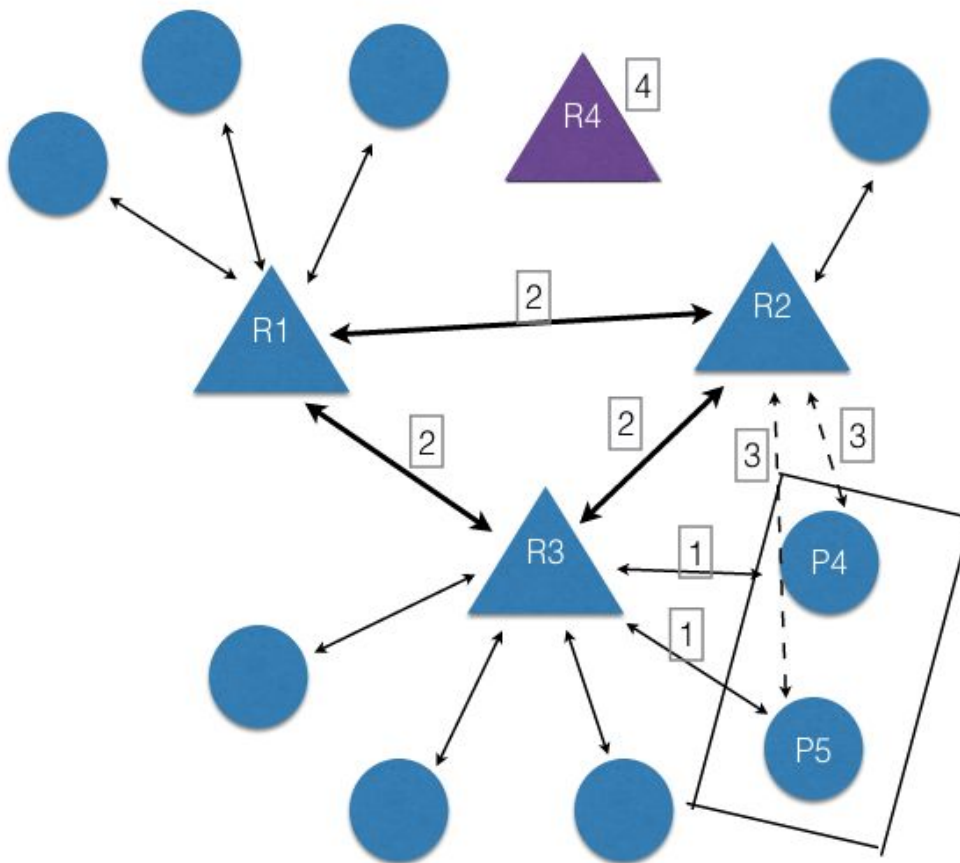


Figure 5.8 Rendezvous peer load balance



In our application, we scale our edge peers through rendezvous nodes. When you have a tremendous amount of users, you can increase the number of rendezvous nodes to let each rendezvous handles a reasonable number of edge peers.

Workflow:

1. When a new edge peer wants to join the p2p network, it will randomly select an available rendezvous peer to connect. As plotted in the graph, P4 and P5 randomly select R3 as rendezvous.
2. All the rendezvous will do a message multicast to exchange the number of edge peers they organize each constant time period(Ex. 1min). So each rendezvous will know who has peers more than average while who has empty slots to handle more edge peers. As plotted in the graph, R1,R2 and R3 do the message exchange.
3. We set up the maximum number of edge peers that a rendezvous peer can handle into a constant. So a rendezvous who has extra load can migrate them to ones who has less load to balance the whole system. As plotted in the graph, P4 and P5 are extra load and are being migrated to R2.
4. When all rendezvous reach the maximum bound, we will open up a new rendezvous to handle incoming edge peers. In this way, we will have more freedom and space to migrate the working load. As plotted in the graph, we set up a new R4 into the p2p network.

## 5.9 Other Imported Packages.

We mainly use jxse 2.6 in our development. All other packages that we have imported can be shown in Figure 5.9.1.

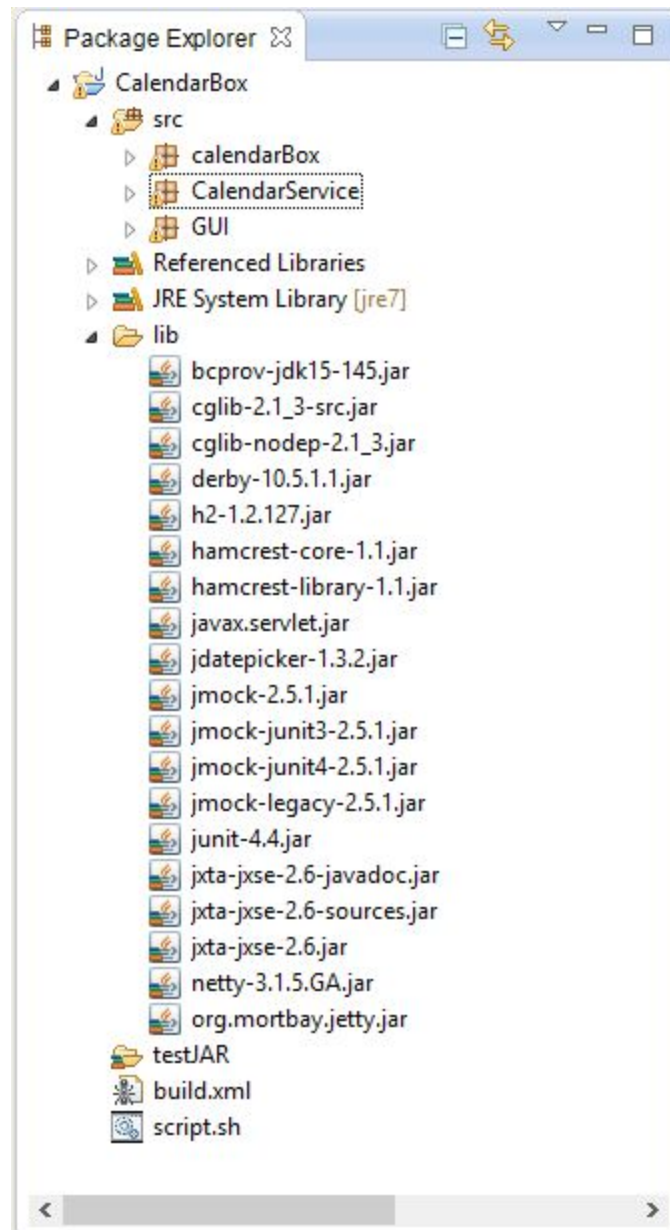


Figure 5.9.1 Java packages imported in CalendarBox Application

## 6. Demonstration

### 6.1 Demonstration Environment

To demonstrate our app, we would be using multiple servers in ECE clusters. The below diagram shows our test setup:

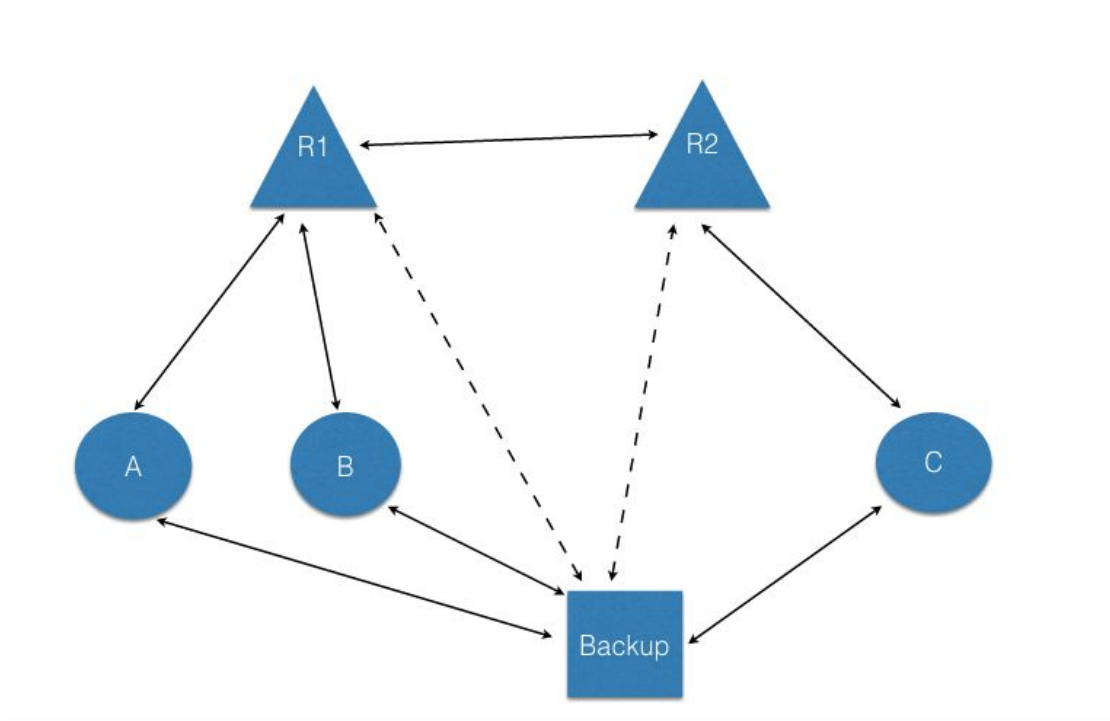


Figure 5.1.1 Demonstration Topology

As seen above, in the current setting, we have 2 rendezvous nodes(R1 & R2), 3 peer nodes(A, B & C) and 1 backup node. Two rendezvous nodes are the main servers which will be used during bootstrapping and for coordination later on. The backup server will be used to store files as we are making a fault tolerant system. These coordinator and backup servers are ECE hosts. Apart from these, client nodes could be laptop/tablets/smartphone devices that use our CalendarBox service.

### 6.2 Normal/Basic Communication Test

Our app can be accessed from any client user interface. Hence, any personal computer like laptop or desktop, tablets and smartphones can access this application. There is no limitation of compatibility. In the basic communication, after bootstrapping, clients could create, modify and delete calendar events and synchronize these operations with other clients.

### 6.3 Rendezvous Node Failure Test

In the current setting, for convenience, we used two ECE hosts as our rendezvous nodes. Client nodes and backup nodes can randomly pick up the rendezvous node to bootstrap, as a result, we balance the workload on different rendezvous nodes. If any one of the rendezvous node dies(In the test, We kill one of the rendezvous nodes), his bootstrapping workload will automatically transfer to the other rendezvous node. In this way, we ensure the high availability of our CalendarBox service. If all rendezvous nodes die(We kill the other rendezvous node in the test), new clients cannot get the CalendarBox service but current clients can still synchronize calendar events with each other.

### 6.4 Client Node Failure Test

For the demo, we will use the above setup. When there is a client failure(In the test, we kill one client node and restart it), we need to ensure that the calendar events are synchronized with him when he gets online again next time. In our design, when client becomes online again, it will first send request to all online users. All other users will send back list of events that this user involved in. So this user get list of events locally. But it may not be the complete copies. So we will refer to Backup to check whether we get the complete ones. Backup node will send back the missing ones. The advantage is that we utilize the p2p to get the files in distributed way. We did not get them all from backup. Finally, we refer to backup and it is a light-weighted request and it will not send back anything under most situations.

### 6.5 Backup Node Failure Test

In our design, backup node will be a default participants for event, in this way, it has all the events copy. And making the events consistency, all the modify, create and delete operations will be reflected on the backup node. If the backup node is down, we cannot make sure the integrity and consistency of the events. But it can still get updates from other peers only. Only when all the hosts holding this event is offline, we cause a problem. So in most situations, backup failure will not affect the correctness of the system.

### 6.6. Consistency of Event Operations Test

In our design, we have to make sure that when one client executes delete or modify operation, all other clients involved in this event should not do the same operation. In the test, we do modify operation at one client node, then we check that we cannot modify or delete it at another client node that participate in this event.

### 6.7 Detailed Test Plan

Setting: 2 rendezvous nodes(R1 & R2), 3 peer nodes(A, B & C), 1 backup node(Z)  
[Backup node should be started before peer nodes]

1. Normal/Basic communication[**Normal/Basic Communication Test, Consistency Test**]
  - share files
  - create/modify/delete files
  - demonstrate lockings[one copy]
2. Kill 1 rendezvous node(R1) [**Rendezvous Node Failure Test**]
  - Kill & Restart C
  - Normal operations still works(Transfer files, create/modify/delete events)
3. Just show backup node's usage[**Client Node Failure Test**]
  - A creates events with B, kill A, B will create event with A
  - Kill B, then start A, A will get data from backup
4. Don't use backup node[**Backup Node Failure Test**]
  - Kill Backup node, A creates with C
  - Kill A, C creates/modifies a new event with A
  - Start A, A will get data from C
5. Kill 1 rendezvous node(R2) [**Rendezvous Node Failure Test**]
  - A & C share/modify/delete events [Availability]
6. Start B[**Rendezvous Node Failure Test**]
  - Since there is no rendezvous node, B would exit

#### Key Points:

- Security[Authentication/Privacy]
- Fault Tolerance[Peer node dies, still have data]
- Availability[Rendezvous node dies]
- Consistency[One copy]
- Scalability[Load Balance]

#### Nodes Allocation

ece002:R1	ece004:Z
ece003:R2	ece005:A
ece006:B	eve007:C



## 7. Project Management

### 7.1 Team organization

The most challenging aspects of a team projects is that we have to reconcile different ideas from team members, and even challenging for our group, members are scattered in different places and face-to-face meeting is impossible. To address this, we relied heavily on different communication tools including emails and google hangout. We used Git for version control. The task breakdown is shown in the table below.

### 7.2 Work Breakdown / task list

Task	Work Done
JXTA Tutorials	Chen, Darsh, Di, Jason ( $\approx 4$ days)
JXTA Peer-to-Peer Basic Setup	Chen, Darsh, Di, Jason ( $\approx 7$ days)
Calendar & Event on single node	Di, Jason ( $\approx 3$ days )
Event Sharing	Di, Jason ( $\approx 3$ days)
File System	Chen, Darsh ( $\approx 4$ days)
File Sharing	Chen, Darsh ( $\approx 4$ days)
Frontend GUI (Web)	Darsh, Di ( $\approx 2$ days)
Frontend GUI (Application)	Chen, Darsh ( $\approx 3$ days)
Debugging and Testing on Baseline	Jason started, Darsh, Di, Chen join later (total $\approx 4 \sim 5$ days )
Final System Debug, Test and Demo	Chen, Darsh, Di, Jason ( $\approx 6$ days)
Presentation and Final Document	Chen, Darsh, Di, Jason ( $\approx 4$ days)

Table 6.1 Work Breakdown Table (Blue Tasks can be dropped later according to schedule)

### 7.3 Skill Challenge

In this project, we choose JXTA as our skill challenge. It builds up our basic communication framework in our P2P sharing network. However, the learning curve is deep. Though we are able to implement basic communication framework upon JXTA, a lot more

functionalities can be explored. From the perspective of accomplishing a good project, JXTA is a too hard skill challenge for all of us. However, from the view of learning P2P network, JXTA is not only an open source for P2P, it is also a complete, detailed, well defined protocol set for P2P network.

#### 7.4 Final Thoughts

It was initially difficult to coordinate since the team members were in two different timezones. But as the project progressed, the understanding between the teammates grew and thus the project was executed successfully.



## 8 References

- [1] <https://s3.as.cmu.edu/sio/index.html#home>
- [2] <https://www.google.com/calendar/render>
- [3] <http://www.oracle.com/us/products/applications/communications/unified-communications/calendar-server/overview/index.html>
- [4] <http://www.officecalendar.com/>
- [5] Zhang, Qi, Lu Cheng, and Raouf Boutaba. "Cloud computing: state-of-the-art and research challenges." *Journal of internet services and applications* 1.1 (2010): 7-18.
- [6] <http://www.dropbox.com/news>
- [7] 842 hosting server, <http://www.ece842.com/S13/>
- [8] Amazon EC2 instances, <http://aws.amazon.com/ec2/instance-types/>
- [9] JXTA HomePage, <https://jxta.kenai.com/>
- [10] JXSE Tutorials, [https://jxse.kenai.com/Tutorials/Tutorials.html#Prog\\_Guide](https://jxse.kenai.com/Tutorials/Tutorials.html#Prog_Guide)
- [11] Coulouris, George F., Jean Dollimore, and Tim Kindberg. *Distributed systems: concepts and design*. pearson education, 2005.
- [12] Dabek, F., Zhao, B., Druschel, P., Kubiawicz, J. and Stoica, I.(2003). Towards a common API for structured peer-to-peer overlays. In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03) , Berkeley, CA, February, pp. 33–44.
- [13] SHA-1, <http://en.wikipedia.org/wiki/SHA-1>
- [14] JXTA 2.0 Super-Peer Virtual Network, <http://www.di.unipi.it/~ricci/JXTA2.0protocols1.pdf>
- [15] CMU ECE Clusters, <https://wiki.ece.cmu.edu/index.php/Clusters>

## 9 Acknowledgements

We would like to thank Prof. William Nace for his guidance and feedback provided during the entire project. We would also like to thank our mentor Darrin Willis for his patient feedback on our project idea.