Software Design Specification

Document Version 1.0

13 March 2007

John Spann
Alex Chee
Bryan Unbangluang
Daniel LaBare
Mike Oster

Table of Contents

1. Introduction

1.1 Application Overview

Te@mSync© is an advanced decentralized file synchronization application used to synchronize folders between a group of computers using peer-to-peer networking. The main goal is to enable a group of users to easily share and access synchronized files on multiple machines.

Te@mSync© will provide lazy synchronization, local storage of shared directories and files, and controls read and write access. Users will not be required to manually update or commit files to a centralized server-based Team directory. The Client will automatically synchronize the shared files on the User's computer when other Users have made modifications. It will also control write access to ensure only one User is writing to a file while enabling other Users read access.

1.2 Document Overview

The Software Design Specification will serve as a comprehensive guide for the development process by providing in-depth and detailed designs of the intended goals and architecture of the product.

1.3 Design Considerations and Constraints

The Client will be operating on the Windows platform and must have an active Internet connection. File synchronization will not be instantaneous and requires a member of the specific Team with the latest version of the file to be online.

The Server will be running a Ruby on Rails framework utilizing an SQL database and an Apache web server. The machine will be required to have processing speeds of 1.4 GHz, 1 GB of RAM, and 200 MB of available hard disk space. The machine will also be running on a T1 connection with an uptime of 95%.

1.4 Definitions, Acronyms, and Abbreviations

- RoR – Ruby on Rails; A programming language that is meant for rapid and easy creation of web servers and services
- P2P - Peer-to-Peer
- DB - Database
- Team - For the purpose of this document, a team is a group of users who want to share files among themselves
- Client – The Te@msync© application running on a user's machine
- User – An individual who interacts with the server and client through web interfaces
- Server - A server refers to the Ruby on Rails framework, mySQL database, and Apache web server
- Location – A unique identifier that corresponds to a single host or computer

2. Design Specifications

2.1 Client

2.1.1 High-Level Overview

The Client runs on the user end machine. It monitors one folder (and all of its subfolders) per team. The user can be part of many teams, so the client may potentially be watching several folders. It sends messages to the Server and which sends back the actions it needs to take in order to become up to date with the latest revision of the team files. \

2.1.2 Low-Level Overview

The client starts by looking for a configuration file within the client folder which contains the users name, password, and a list of teams. If there is no configuration file present (i.e. the first time the program is run) it will prompt the user for their username and password. If the file is present, it will load the values from the file to identify the user and any teams that it is associated with. For every team it finds in this configuration file, it will create a DirectoryMonitor and a FileTransferManager for each associated team. The DirectoryMonitor object creates a ServerProtocol object which handles all of the communication between the Client and the Server.

Once all of these threads are running the client begins sending http requests to the Server via the ServerProtocol object to get the status xml file. This file will tell the Client if it is up to date on all of its files. If the user is not up to date, this status file will contain certain actions that the user must take to become so, such as a listing of the files the user needs, as well as what other clients currently connected have those files. If the user adds a file or deletes a file, or performs a check in/out, it will notify the server via http requests so it can tell the other clients about the performed actions when they receive their next update page.

2.2.3 Low-Level Component Design

Client	
Description	The client defines the entry point of the client application. This is where you will find _tmain()
Data	
Public Operations	

DirectoryMonitor	
Description	The DirectoryMonitor class is a thread that actively monitors a specified folder looking for new files, or deletion of files.
Data	path – the path of the folder to monitor revision – the global revision of the entire team folder teamNumber – the unique number that identifies a team
Public Operations	DirectoryMonitor(path, teamNum, rev) – constructs the object ~DeriectoryMonitor() – destroys the object run() – tells the object to begin monitoring the folder getTeamNumber() – returns the team number of the folder being monitored getRevision() – returns the global revision of the folder getPath() – returns the path of the folder being monitored by the object

FileReceiver	
Description	This class creates a socket bound to a specified port to receive a file to the specified path
Data	client – the host name of the client you are receiving the file from path – the relative path of the file port – the port on which to receive the file team – the unique team number revision – the revision of the file
Public Operations	FileReceiver(client, port, path, team, revision) – constructs the FileReceiver object run() – runs the Thread and receives the file

FileSender	
Description	Sends a file to the specified socket
Data	socket – the socket to which the file is to be sent
Public Operations	FileSender(socket) – creates the FileSender object run() – runs the Thread and sends the file

FileTransferManager	
Description	Listens on a specified port and creates FileSenders for file transfers
Data	port – the port on which to listen
Public Operations	FileTransferManager(port) – creates the object getFile(host, port, path, local) – gets a file from another client and puts it in the directory specified by local run() – runs the Thread

Message	
Description	This is the class that defines the structure of our messages. There are 9 types of actions a message can convey
Data	enum Action { ADD_FILE, DELETE_FILE, CREATE_TEAM, JOIN_TEAM, GET_FILE, SEND_FILE, CHECK_IN, CHECK_OUT, UPDATE }; type – the type of action args – the variable length list of arguments
Public Operations	Message(Action, …) – creates a message of type Action and accommodates for a variable number of arguments. ~Message() – destroys the object getType() – returns the type of message getMessage() – returns the message

ServerProtocol	
Description	Regularly polls the server to receive the status xml page. It stores the actions to be taken in a queue which is processed every 30 seconds.
Data	messageQueue – holds all of the messages to be performed before next poll revision – the global revision of the folder the object is tied to
Public Operations	ServerProtocol(revision) – creates the object with the team global revision number run() – runs the Thread addAction(action) – appends an action to the queue to be processed

Thread	
Description	An abstract class implementing a single running thread
Data	handle – a pointer to the thread process within the system callback – a pointer to the object implementing the Thread interface
Public Operations	createThread(obj) – creates a Thread from the specified object run() – runs the Thread setCallback(ptr) – sets the pointer to the ThreadManager getCallback() – returns the pointer to the ThreadManager getHandle() – returns the thread handle

ThreadManager	
Description	The ThreadManager creates handles, processes Thread requests, and cleans up threads as needed
Data	directoryMonitor – a vector of pairs. The pairs are between a pointer to a DirectoryMonitor object and an associated handle
Public Operations	createDirectoryMonitor(path, revision, id) – creates a threaded DirectoryMonitor object runThread(thread) – runs the given Thread killThread(thread) – kills the given Thread killAll() – kills all currently running threads in Client findTeam(teamNum) – returns a pointer to the DirectoryMonitor associated with the given team number

XMLStreamer	
Description	Connects to the TeamSync server to a specified page and grabs XML as needed
Data	hSession, hConnect, hRequest – all used to form http requests that are sent to the TeamSync server dwSize – the size of the buffer returned from the http request
Public Operations	XMLStreamer(location) – creates an object that connects to the specified location to retrieve an XML page ~XMLStreamer() – destroys the object read(buffer, sizeToRead) – reads a buffer of size sizeToRead to be parsed getSize() – returns the size of the stream

2.2 Server

2.2.1 High-Level Overview

The Server is an active machine interacting with Users through HTML web interfaces and interacting with Clients through XML pages. The web interface handles User registration and authentication, selects the Client, controls file check-in and check-out, and provides Team management abilities. The XML pages are responsible for Team status, adding and deleting files, file and peer listing, the ability to create and join Teams, and Client registration and authentication.

2.2.2 Low-Level Overview

The Server will be using multiple controllers to provide interfaces and pages to interact with the Users and Clients. The User Controller generates various views depending on the User input. The Client Controller generates various views depending on the Client input.

2.2.3 Low-Level Component Design

Client Controller	
Description	Client Controller is responsible for providing XML pages to the Client application by responding and interacting with Client input
Data	
Public Operations	Login(Email, Password) – Verifies a Client is who the host claims to be CreateTeam(Email, Password, Team Name) – Creates a new Team in the database CreateLocation(Email, Password, Team ID, Location Name) – Creates a new Location in the database Check-in/out(SharedFile) – Provides a XML page informing the Client to modify file permissions; Check-out will make files read-only; Check-in will make files read-write ShowPeers(User ID, Team ID) – Displays the list of peers in the Team that are online; Each peer listing will include last time peer was online ShowFiles(User ID, Team ID) – Displays the list of files belonging to a single Team that the User is a member of; Each file will also display the latest version number TeamStatus(User ID, Team ID) – Displays the status of the local files compared to the most up-to-date Team status; the Client is either up-to-date or requires file synchronization AddFile(Filename, Team ID, User ID) DeleteFile(File ID) JoinTeam(Team ID)

User_Interact Controller	
Description	User_Interact Controller handles the HTML interaction between the User and the Server relating to registration, authentication, location, file control, and Team management.
Data	
Public Operations	Register(Email, Password) – Creates a new User in the database Authenticate(Email, Password) – Verifies a User is who he or she claims to be ClientSelect(Location, User) – Specifies the Location the User wishes to use Check-in/out(SharedFile, User) – Enables a User to check-out a shared file to have write access only if it is checked-in; otherwise a User will have read access only ShowTeams(User) – Displays the list of Teams the User is a member of ShowFiles(User, Team) – Displays the list of files belonging to a single Team that the User is a member of

User Model/Class	
Description	User Model handles the process of authenticating Users.
Data	ID – Unique number to identify a User Email – A unique address identifying a User Password – A unique password to verify a User is who he or she claims to be
Public Operations	Add(Email, Password) – Creates a new User in the database and assigns a new ID number to User Remove(ID) – Removes the User from the database given unique ID number

Location Model/Class	
Description	Location Model identifies a unique Location to a single User; Includes the Port Number that aids the Server to communicate with the Client
Data	ID – Unique number to identify a Location Name – A name for the Location for the User. Port Number – The port the Server should use to connect to Client IsUpdated – Indicates whether the User is updated or not LastON – Indicates last time the User was connected to Server
Public Operations	Add(Name, Port Number, User) – Creates a new Location, linking it to a User while specifying the listening port Remove(ID) – Removes the Location from the database given unique ID number

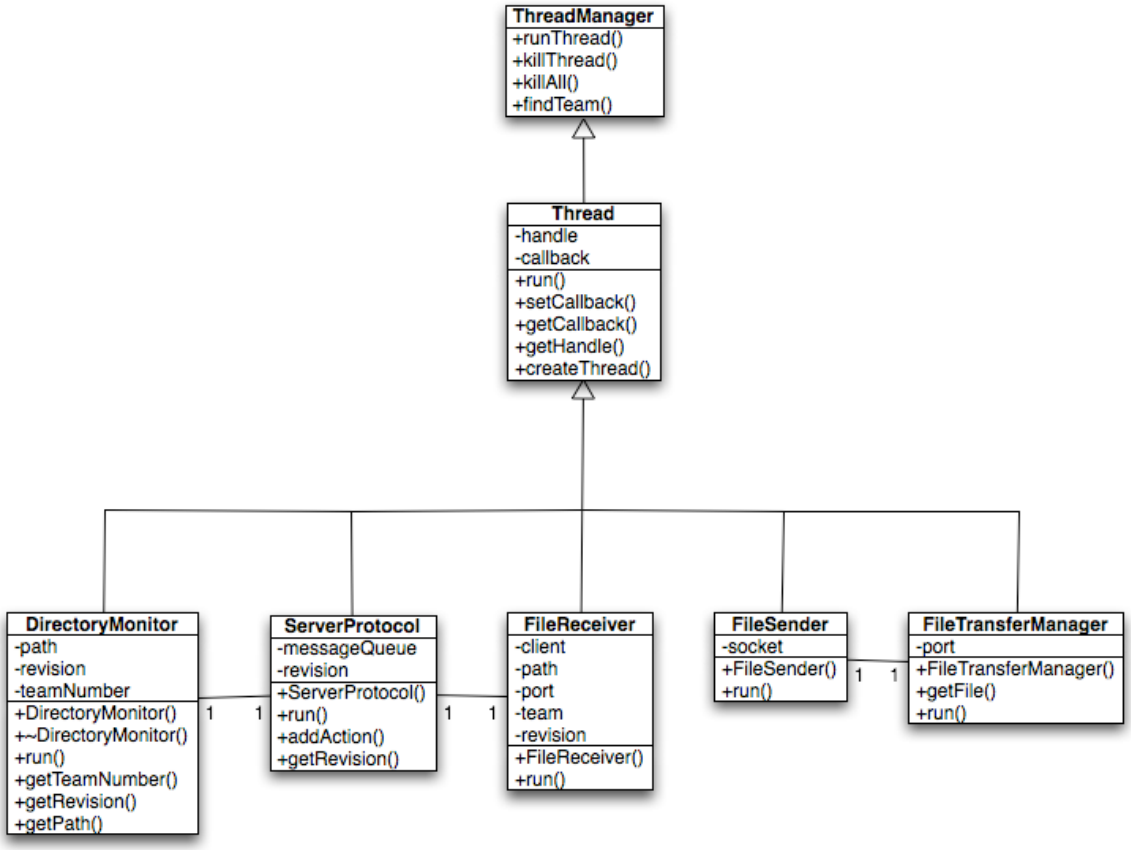
Team Model/Class	
Description	Team Model identifies the folders to synchronize
Data	ID – Unique number to identify a Team Name – A name for the group folder to share Date – Time of Team creation Revision – Number to indicate most up-to-date Team version status; It will be incremented after every check-in of a file in the Team
Public Operations	Add(User, Name) – Creates a new Team in the database with the User as the creator. Then it timestamps Date at the time of the method call and sets Revision to 0. Remove(ID) – Removes the Team from the database given unique ID number

Team Member Model/Class	
Description	Team Members identities a User to a Team.
Data	ID – Unique number to identify a Member
Public Operations	Add(Location, Team) – Adds a Location to the specified Team. Remove(ID) – Removes the Location from the database given unique ID number

Shared File Model/Class	
Description	Shared File Model identifies a Filename to a Team
Data	ID – Unique number to identify a file Filename – Path with the name of the file; Path will start at the shared folder Status – Indicates if the file is checked-in or checked-out Version – Specifies the version of the file; Increments value every time the file is checked-in CheckedOutBy – Identifies User who last checked-out the file
Public Operations	Add(Filename, User, Team) – Creates a new Shared File entry in the database for Team. Sets Status to be checked-out and Version to 0. Remove(ID) – Removes the Shared File from the database given unique ID number

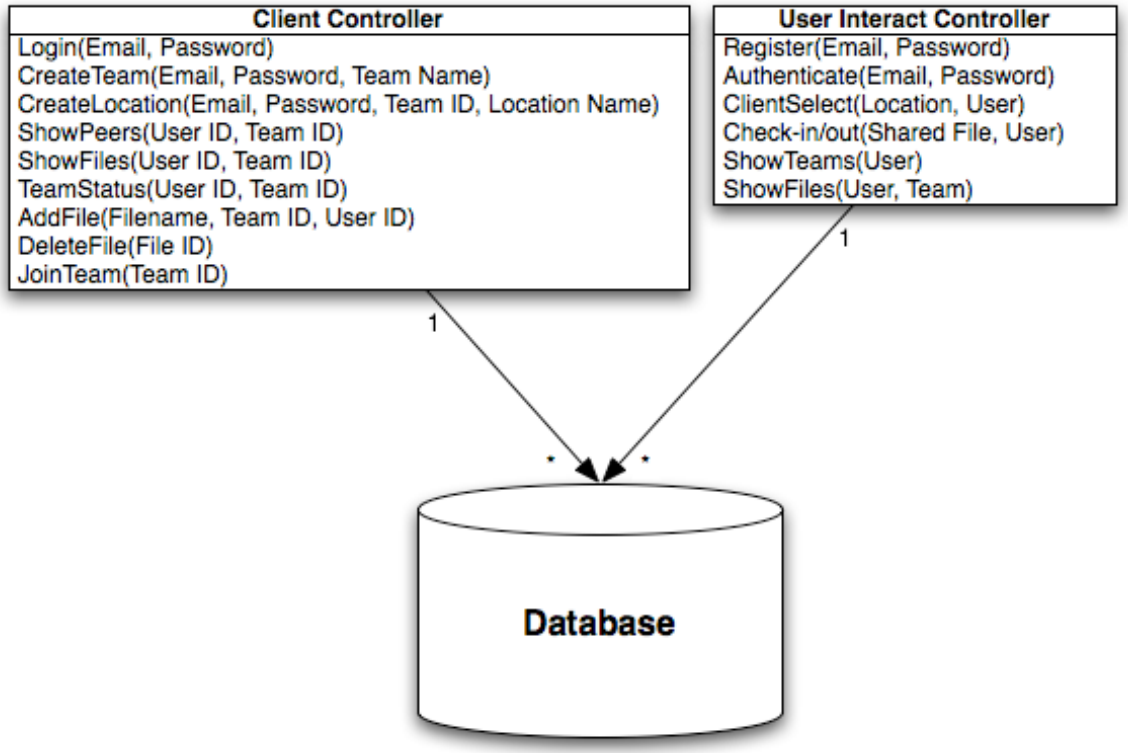
3. Class Diagrams

3.1 Client Class Diagrams

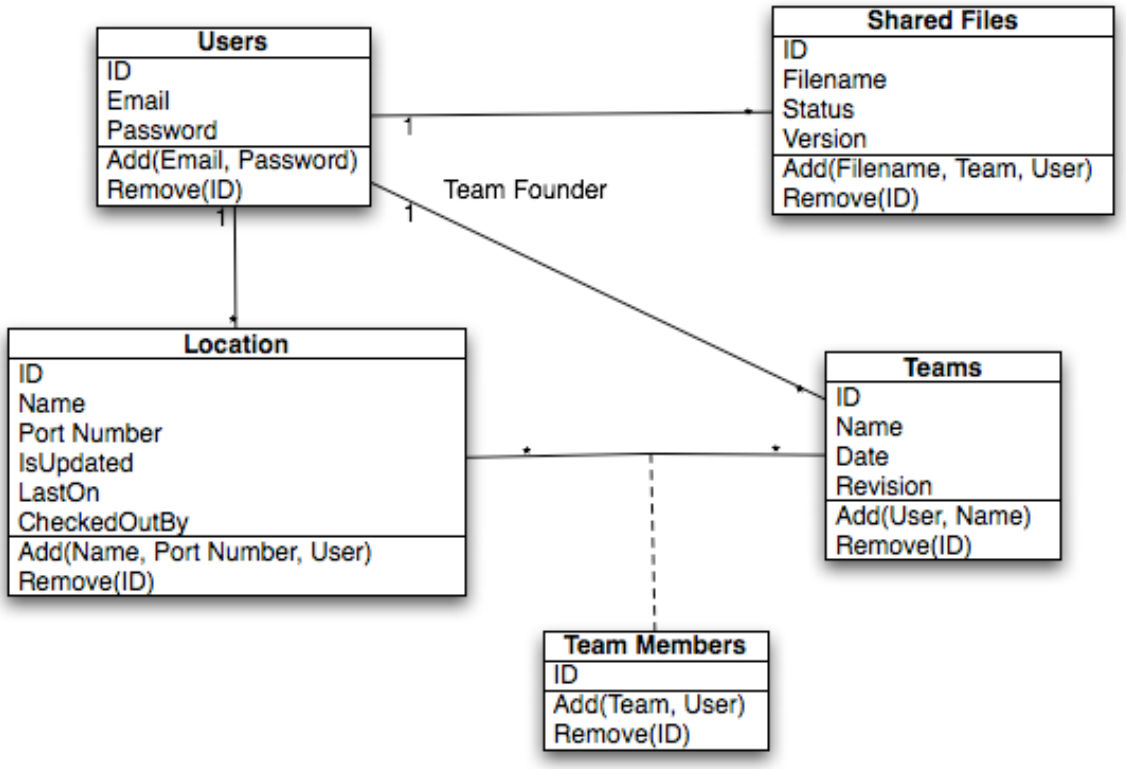


3.2 Server Class Diagrams

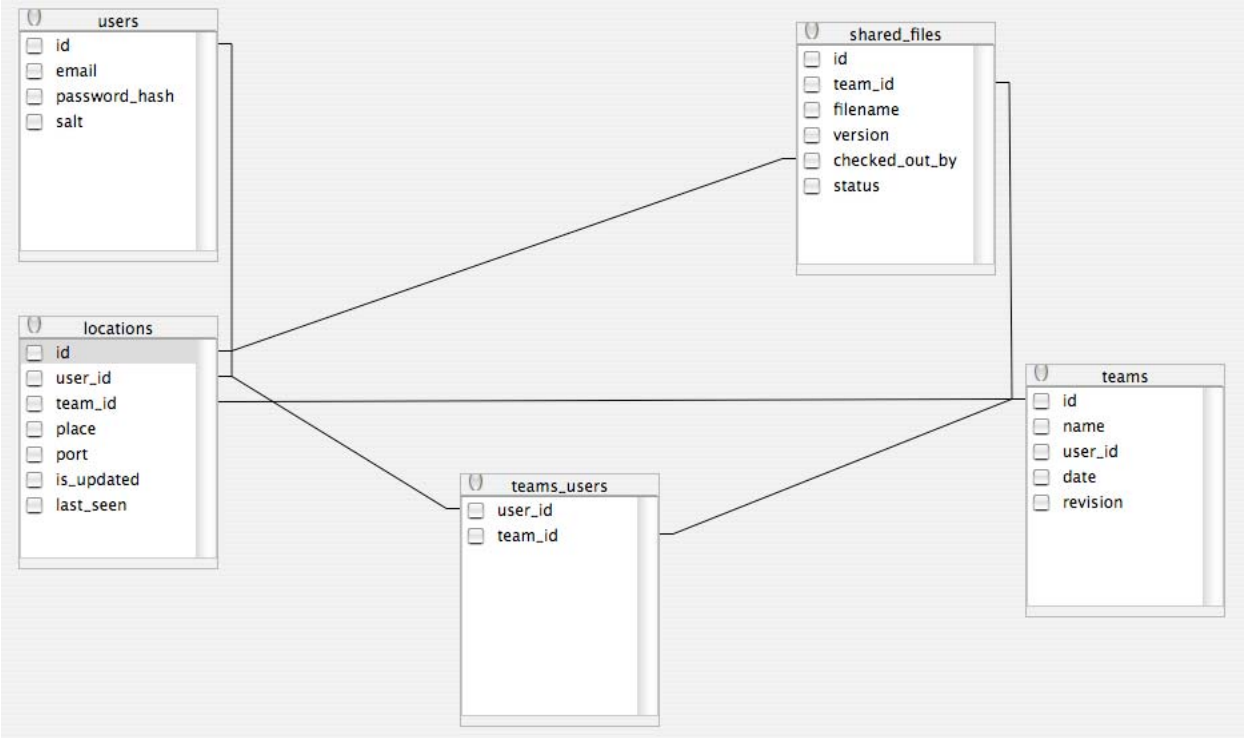
3.2.1 Controller-Model Class Diagram



3.2.2 Detailed Model Class Diagram

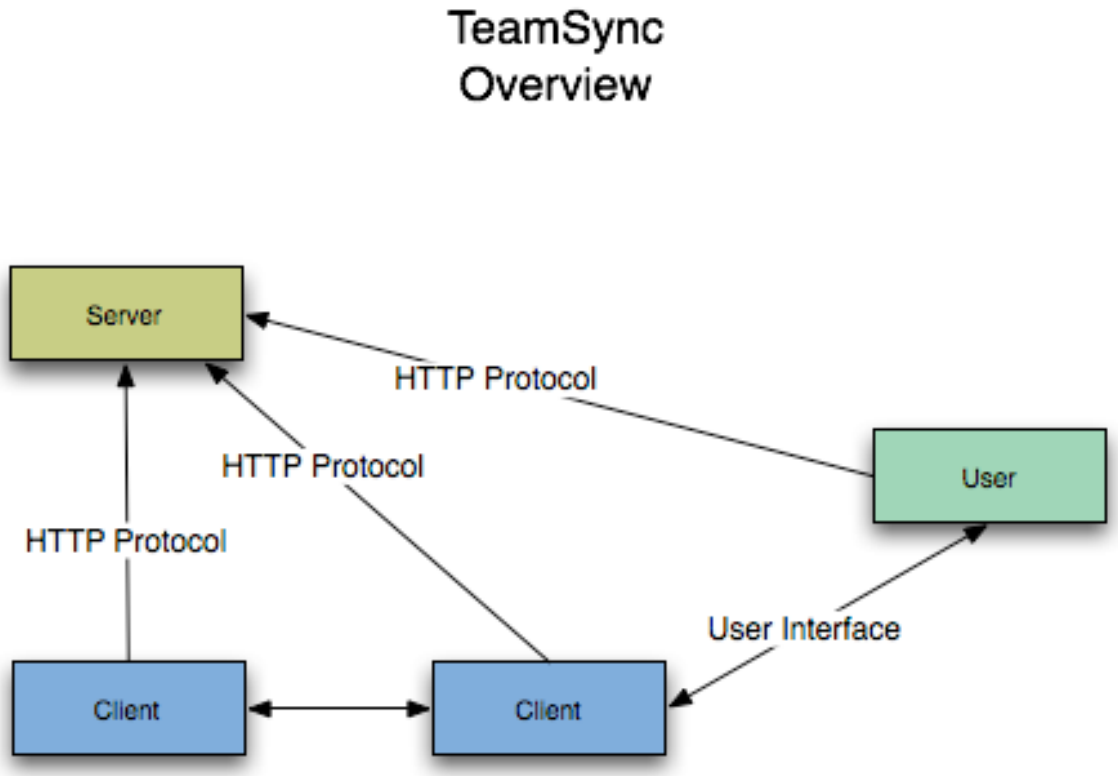


3.2.3 Detailed Database Diagram



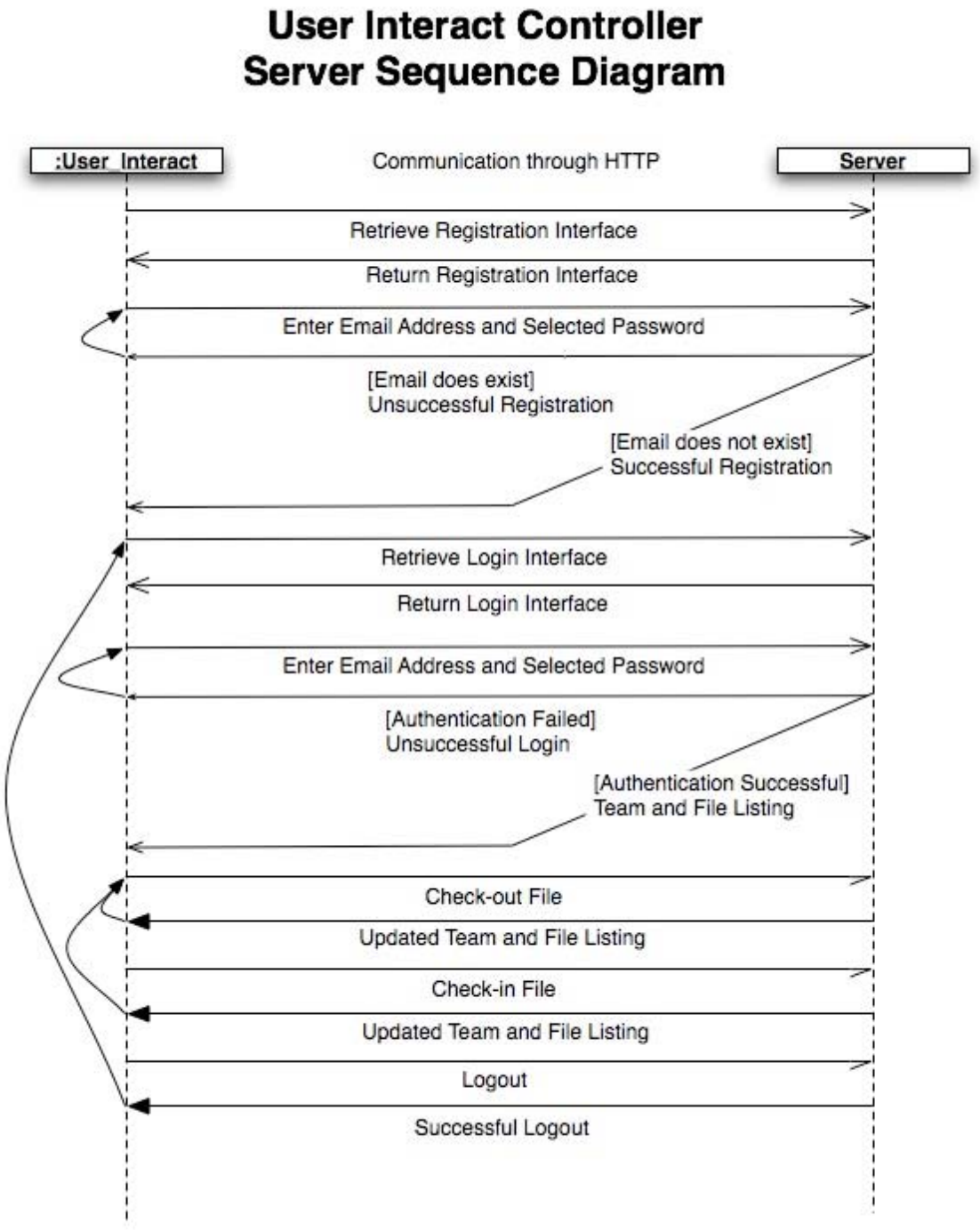
4. Interaction Diagrams

4.1 High-Level Overview Diagram

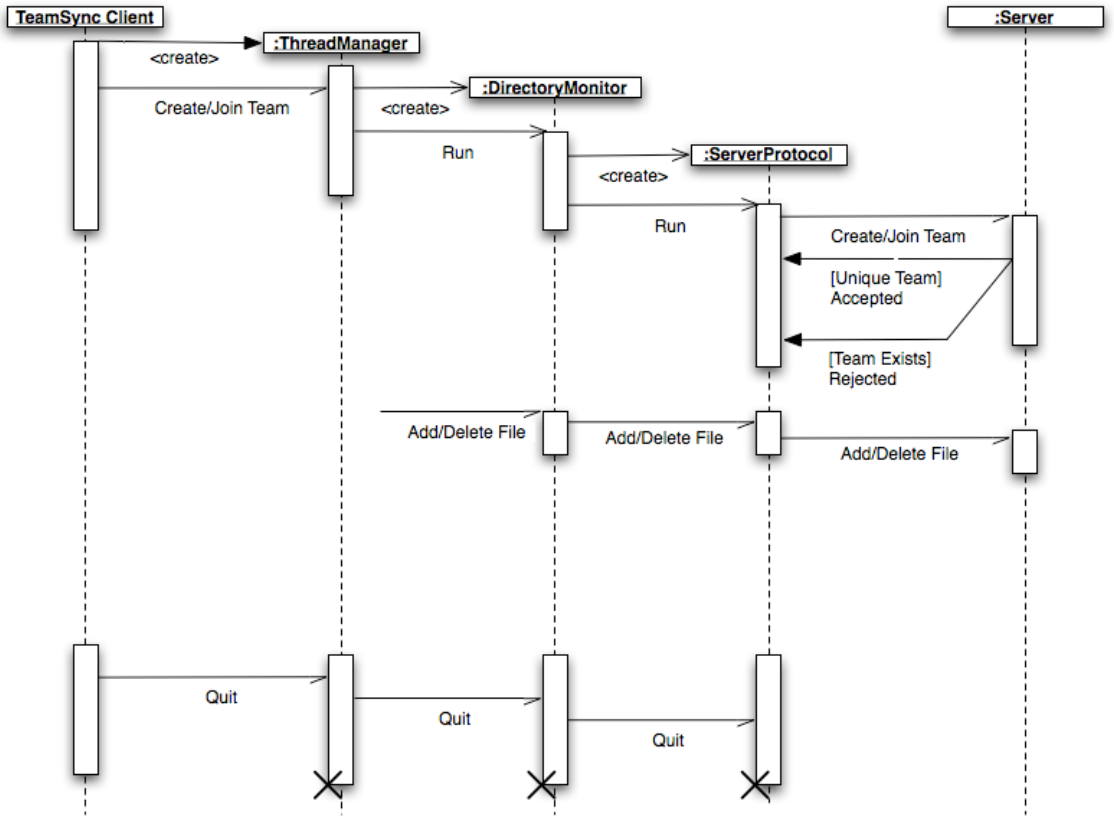


4.2 Sequence Diagrams

4.2.1 User Interact-Server Sequence Diagram

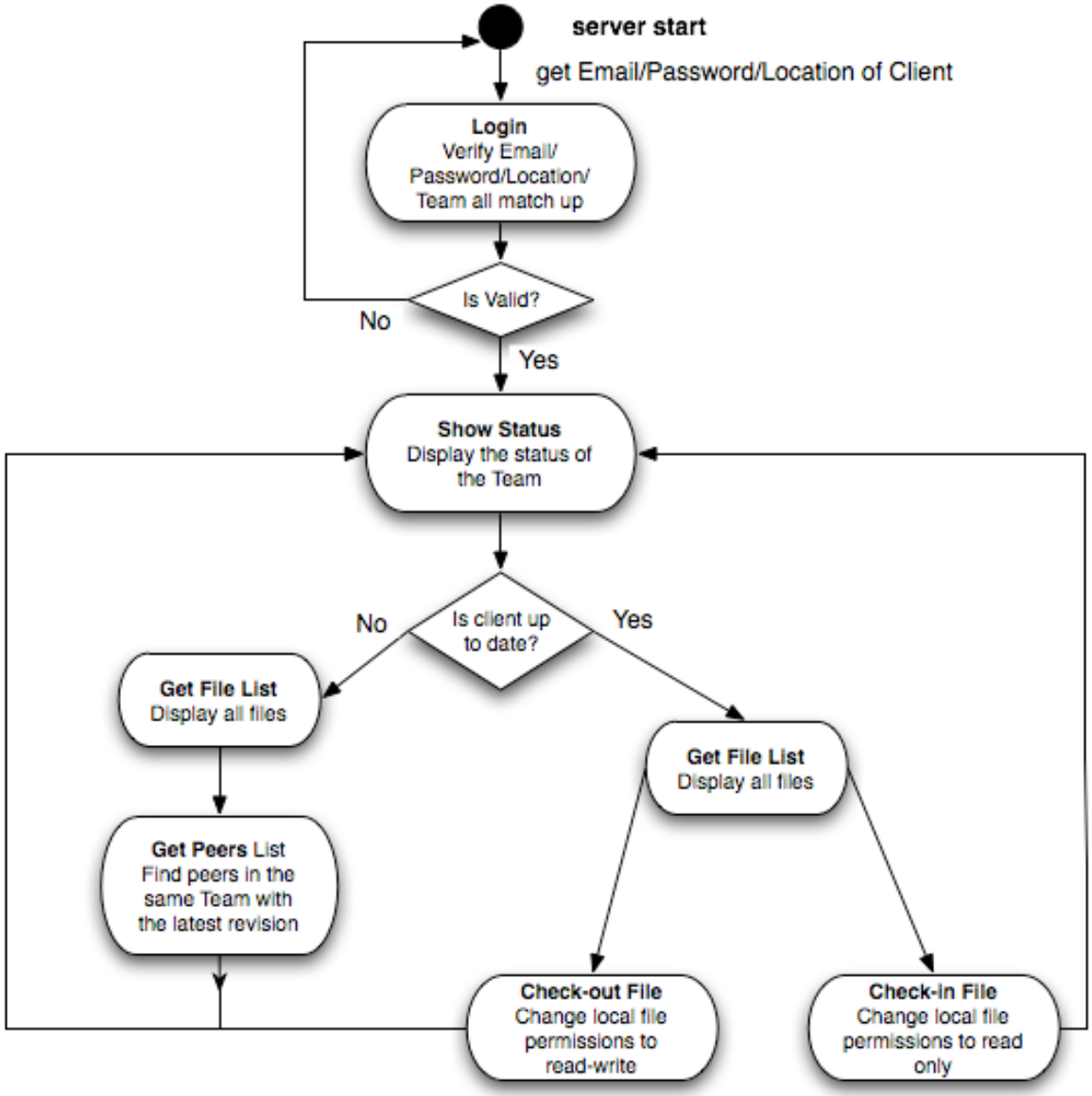


4.2.2 Client Sequence Diagram

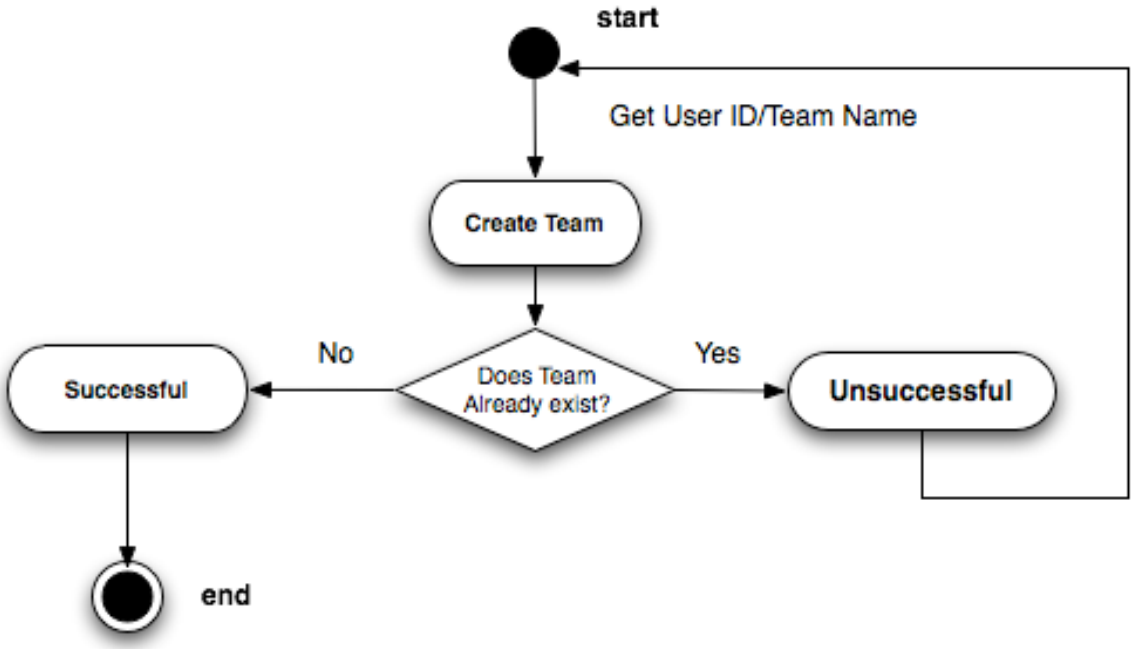


4.3 Flow Charts

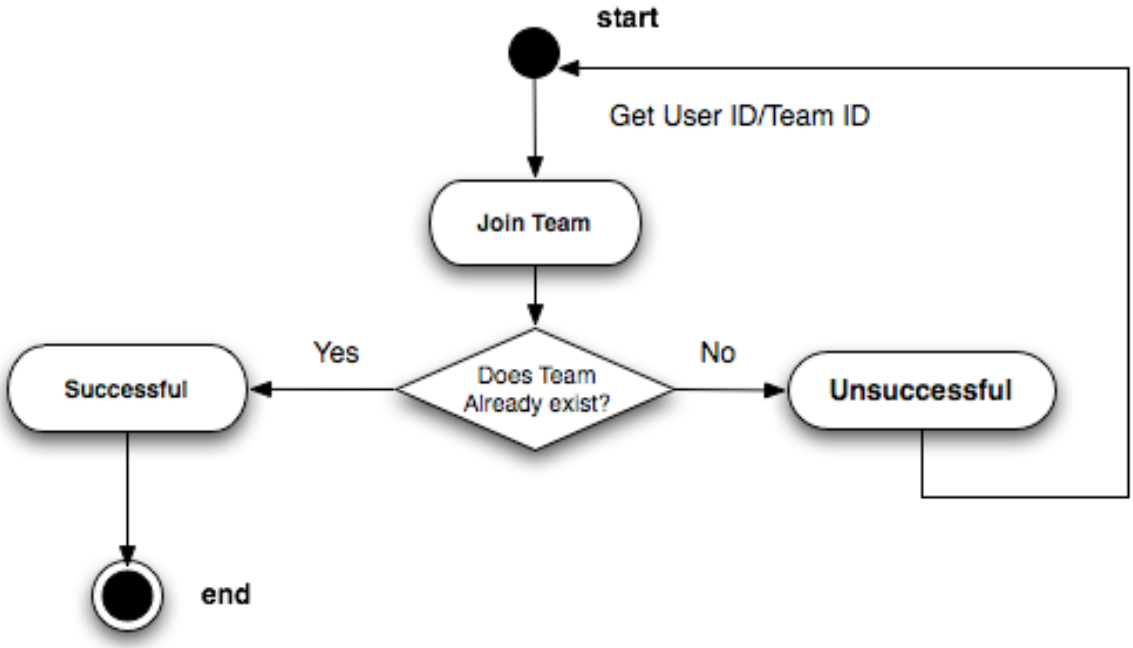
4.3.1 Client-Server Login Flow Chart



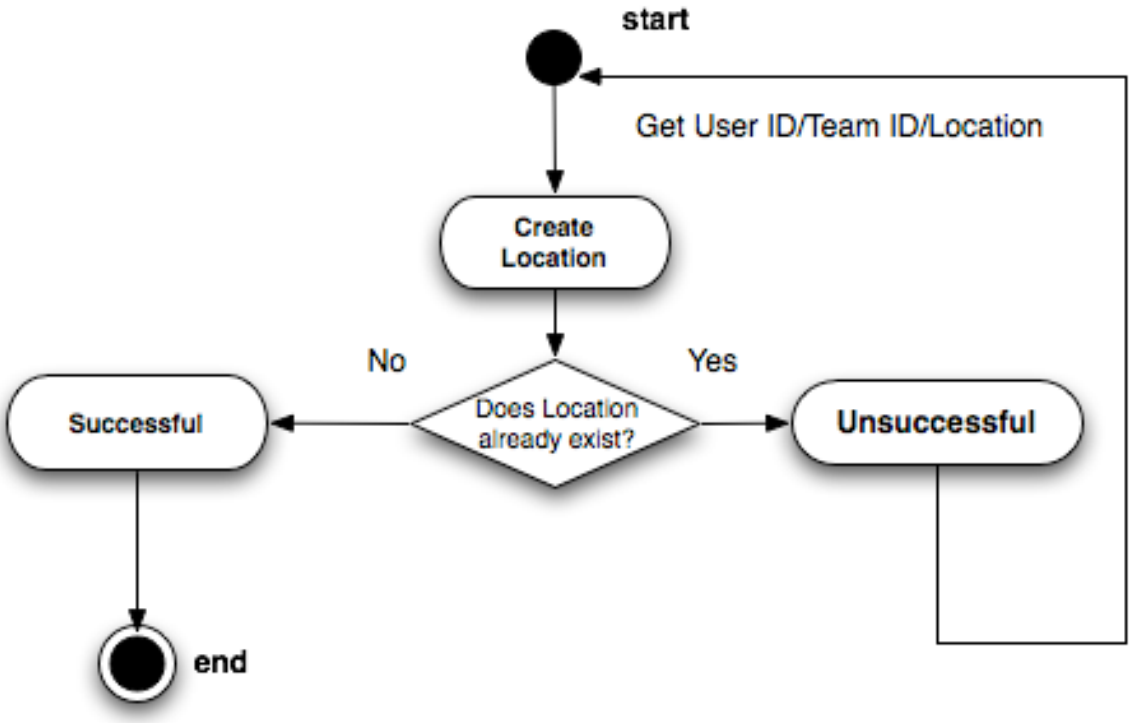
4.3.2 Client-Server CreateTeam Flow Chart



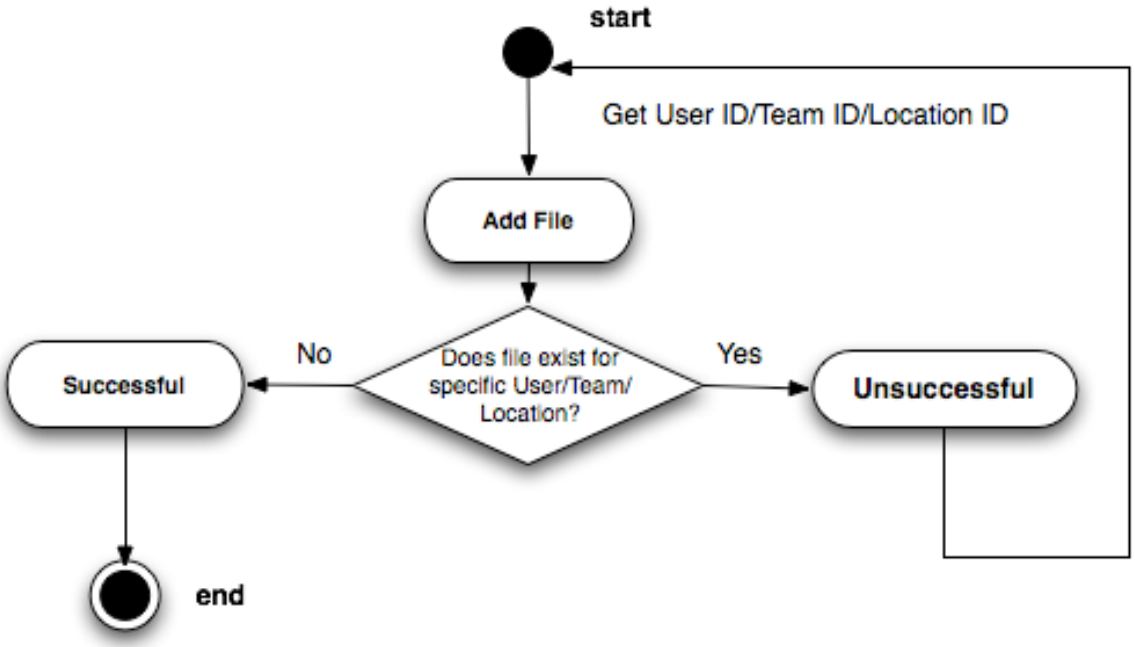
4.3.3 Client-Server JoinTeam Flow Chart



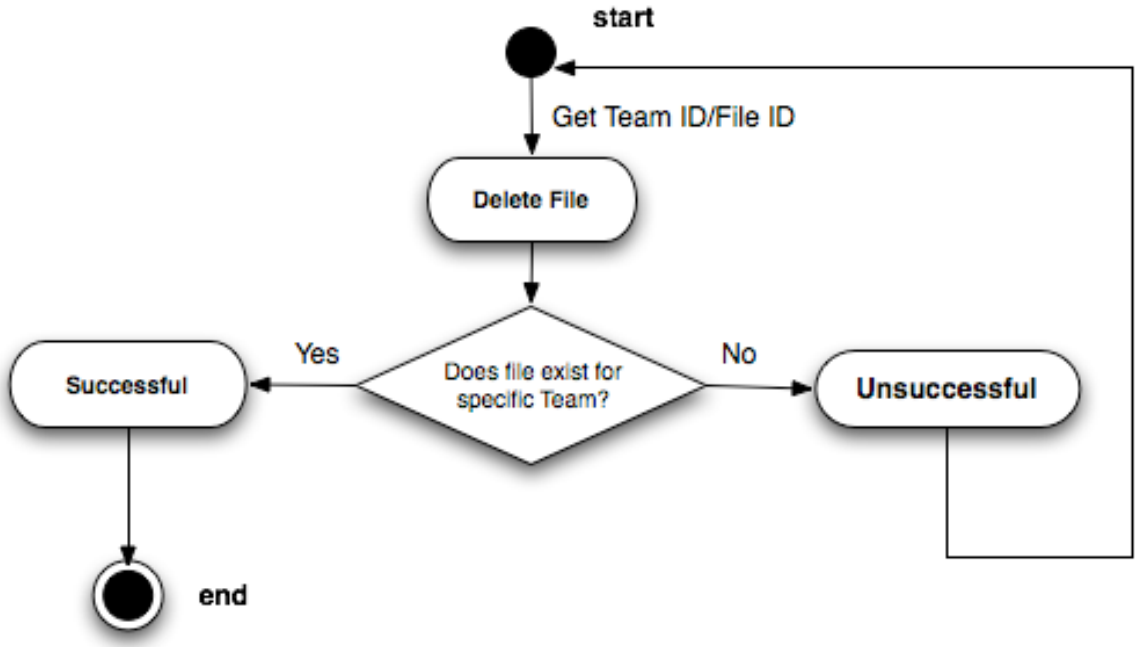
4.3.4 Client-Server CreateLocation Flow Chart



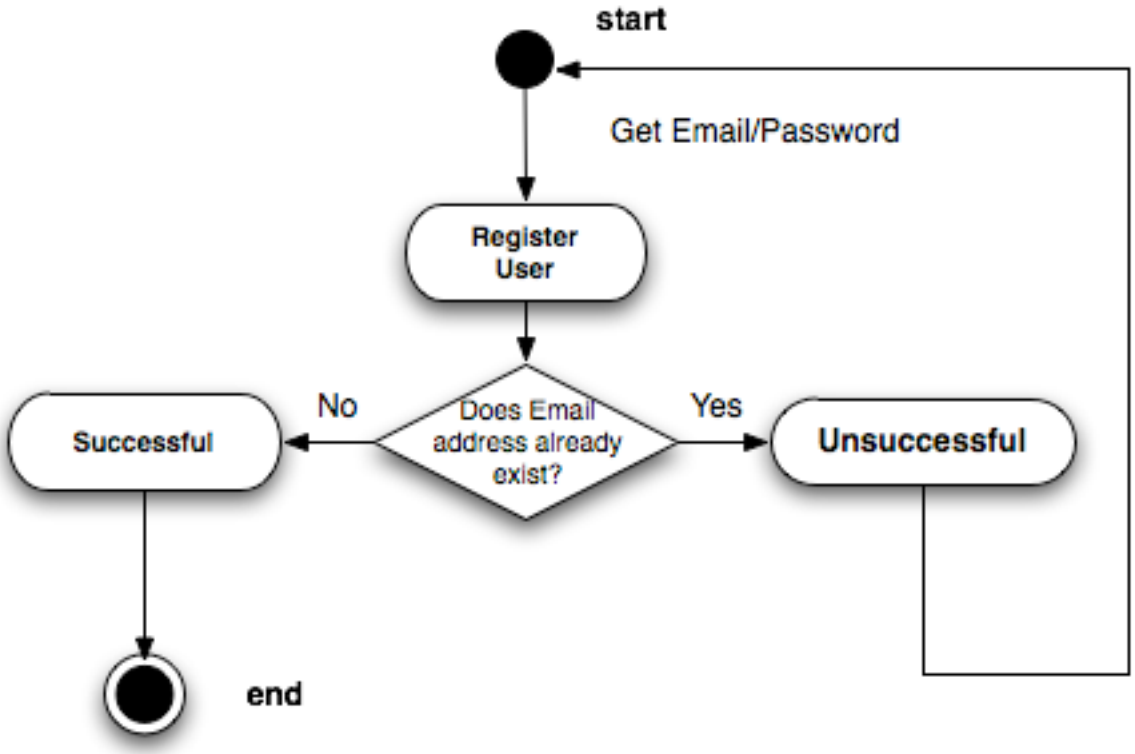
4.3.5 Client-Server Adding File Flow Chat



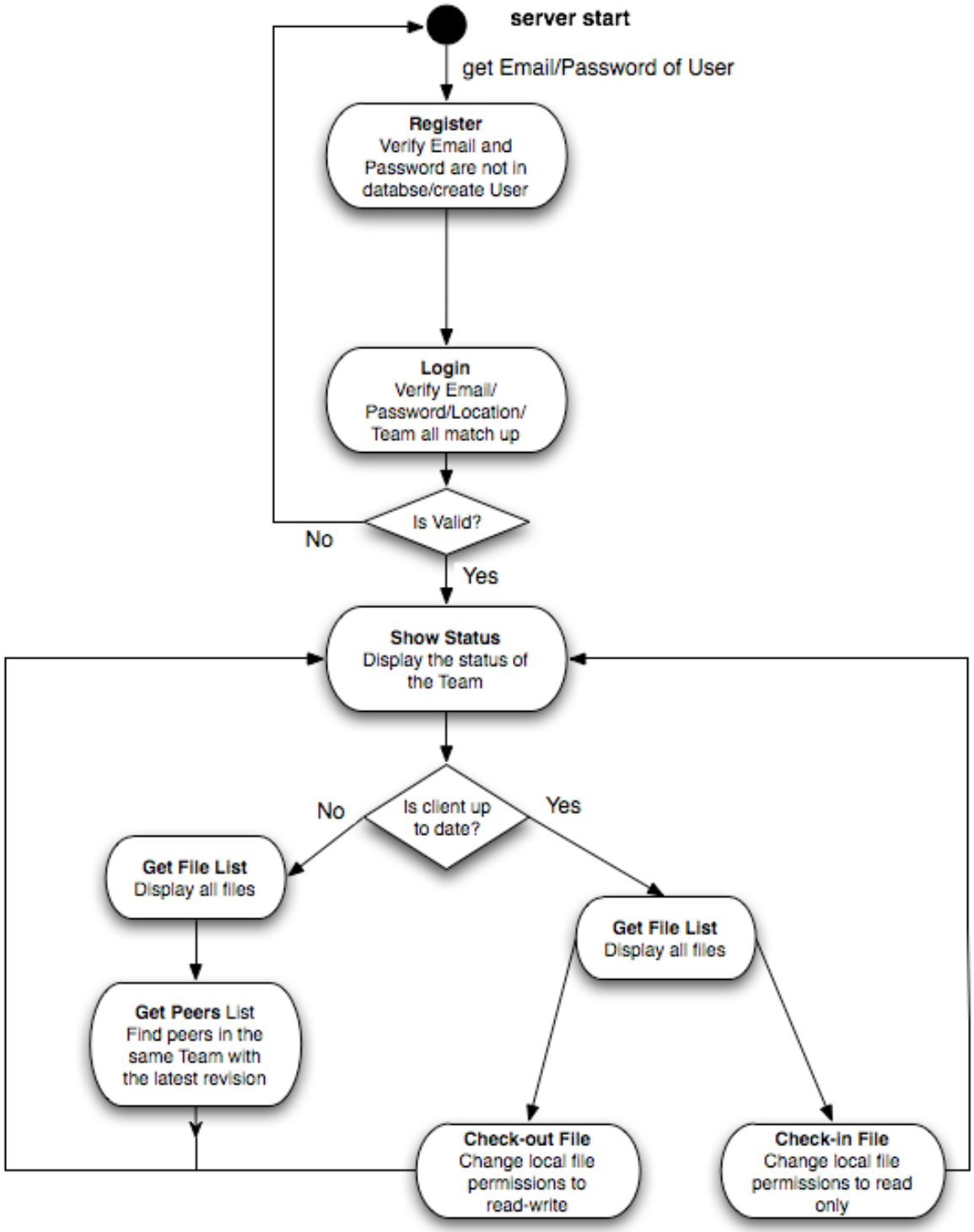
4.3.6 Client-Server Deleting File Flow Chart



4.3.7 User_Interact-Server Register Flow Chart



4.3.8 User_Interact-Server Login Flow Chart



5. Conclusion

Te@mSync© serves as a decentralized Client-based application to provide Users with the ability to easily and seamlessly share and synchronize folders across multiple Locations with their Teams and peers. The key benefits include less reliability and dependency on the Server, local storage of shared directories and files, ability to control read and write access, and distribution and synchronization of Team files through peer-to-peer networking.

Te@mSync© differs from existing version control applications such as CVS and SVN such that the User will interact with their Teams and Clients through an HTML web-based interface. The key difference is that Users will not be required to manually update or commit files to a centralized server-based Team directory. Instead, the Client will automatically synchronize when modifications have been made in the Team directory. Another key feature includes controlling the User who has write access to any given file at any given time. This will prevent conflicts from multiple Users writing to the same file simultaneously.

Te@mSync© will fulfill and exceed all expectations from corporations and businesses regarding file sharing and synchronization. Te@mSync© strives to provide the best services and products.

6 Appendices

6.1 Client API

Visit http://teamsync.castleparadox.com/docs/client/

6.2 Server API

Visit http://teamsync.castleparadox.com/docs/server/