Bilkent University

Department of Computer Engineering

Senior Design Project

S.U.N (Social University Network)

Low-Level Design Report

Ahmet Ertan, Basri Karademir, Damla Özge Özgen, Dilan Yatan, Yeliz Kurt

Supervisor: Can Alkan
Jury Members: Hakan Ferhatosmanoğlu and Çiğdem Gündüz Demir
Innovation Expert: Mehmet Çakır

Progress Report
Feb 16, 2015

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS492.

Table of Contents

1. Introduction

SUN app, our senior project, is a smart phone application which is based on check-in control system. The main purpose of the application is socializing university students via some event-based creations of the application. Beside the initial goal of the application, gamification method will be applied to make application more interesting.

The initial goal of our application is to make university student more social in their university life. The loaded syllabuses and exam concerns leads students to communicate lesser and makes socializing harder. Via our app, it will be easier to arrange events in order to encourage students to communicate each other even with the students who they never met.

For this purpose, SUN application aims to make students work together, participating events together or creating events. Application will have study session events part in order to organize people to study together, events to participate like library or sports hall events part and all these events will have their own points to earn.

In this report, the low level design of our system will be explained in detail. Firstly, our object design trade-offs of the system will be determined and explained. Then interface documentation guideline for our system will be provided. Next, engineering standards that our structure of the system supports and the detail of applying these standards will be explained. In the following section, the packages of our system and their functions will be described clearly and in detail.

1.2 Object Design Trade-Offs

In the low-level object design phase some trade-off decisions may needed to be made:

1.2.1 Cost versus Functionality

SUN application has lots of functions for the user such as registration, event creation, course change and etc. Every function of the system requires extra design which causes extra cost for the development.

1.2.2 Functionality versus Usability

SUN application aims to meet at a point, which satisfies both efficient user interface and functionality. SUN is a high functional program, which also aims to provide an easy use user interface for the users. If the functionality is too complex, user may not use easily. So, a middle point for the user will be found while meeting all the functional requirements.

1.2.3 Cost versus Understandability

Codes must be clear in order to make further improvements. So developers should write comments and write the codes clear. If the codes are not understandable then this situation will cause extra cost in the improvement stage.

1.2.4 Cost versus Security

User should be logged in to the system in order to use the application. Before they register they should be authorized with a username and password. This causes extra cost.

1.3. Interface documentation guidelines

The guidelines of the class interface is as follows:

Class/Interface Name	Name of the class/interface
Class/Interface Description	Short description of the class/interface
Package	Package name that includes the class
Attributes	Attributes that the class have
Operations	Operation definitions provided by the class

Figure 1: Class interfaces table

1.4 Engineering Standards

In the SUN application reports, there are several UML standards that we have provided such as class diagrams, activity and sequence diagrams and the use case scenarios. Using these standards and visual models of the system show that, the software design of the application is in object-oriented structure. With diagrams and visual models we aim to show how to use and manage the system by providing detailed explanations.

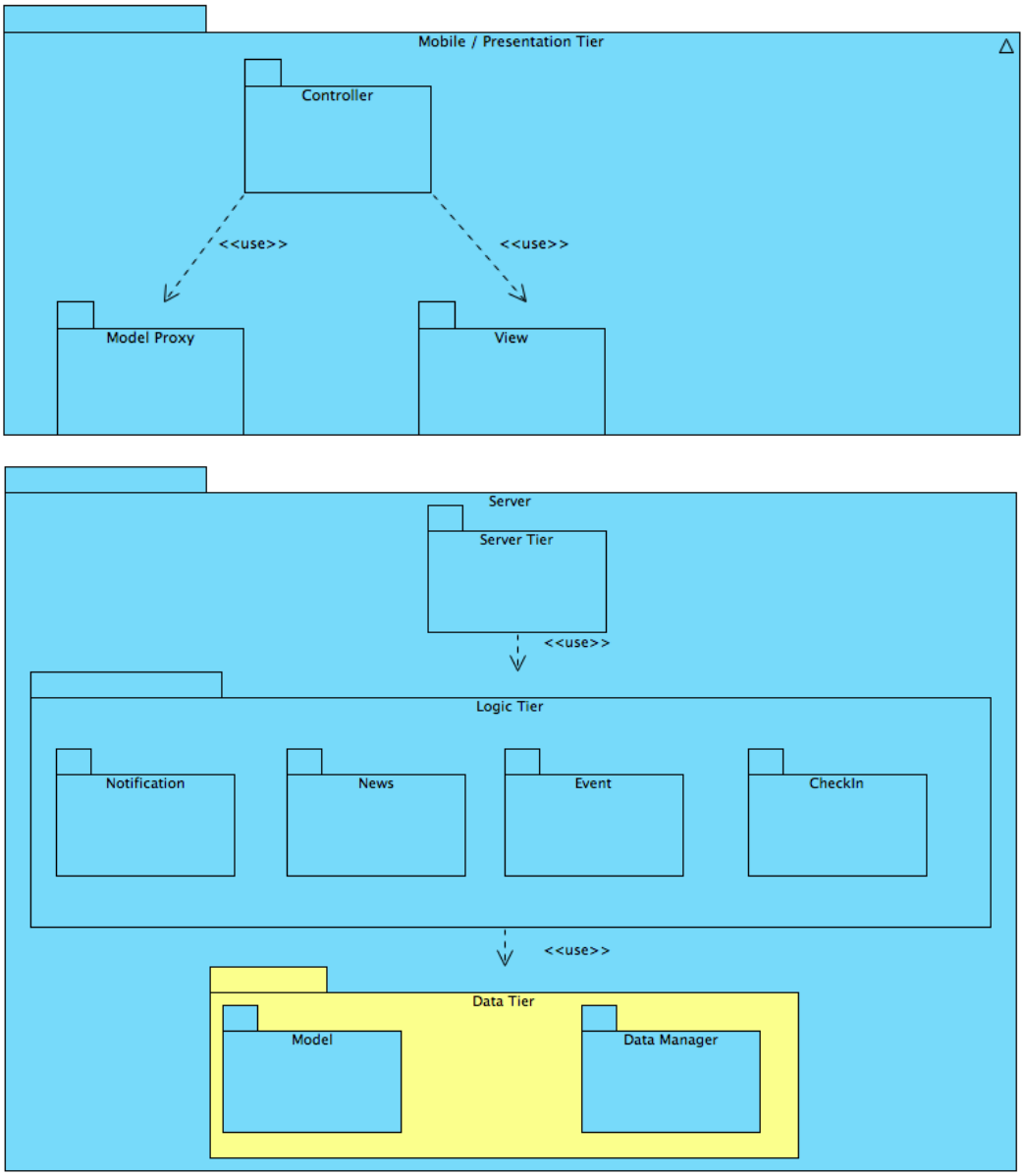
IEEE 802 is a standard for wireless standards (LAN connections). The standard composes all the networking standards that will be a physical layer on wireless to Ethernet connection. Since we provide a system that will use network to changing the data among the smart phones, this standard will be helpful.

IEEE 1471 is a standard for software architecture description languages. The architectural definitions and models are used to create and manage the software design to develop our application for better communication among other smart phones. The diagrams and visuals are helpful to integrate the pieces of the system together and for the co-operation.

These IEEE standards the most significant ones that are used commonly among the other software designs.

2.Packages

There are 2 main packages of the system. These are mobile and server packages, and they handle the android application and the backend of the system respectively.



3.Class Interfaces

3.1 Global

Interface	**SunActionPerformer**
Description	This interface declares every possible action that a client can request. So API clients and CommunicationHandler's must realize these methods to work properly.
Package	sun
Attributes	-

Operations	createEvent(name: String, date: Date, joinable: Joinable): Event deleteEvent(event: Event): Boolean getNewsFeed():FeedItem[] checkIn(event: Event): CheckIn joinEvent(event: Event): Boolean leaveEvent(event: Event): Boolean addInterest(interest: Joinable): Boolean removeInterest(interest: Joinable): Boolean setProfilePicture(avatar: String): Boolean searchUser(keyword: String): User[] addFriend(friend: Student): Boolean getNotifications(): Notification[] getJoinable(id: int): Joinable getJoinableEvents(joinable: Joinable): Event[]

3.2 Server

Class	**SUN**
Description	This class is the main manager of the backend system. It has the main method and instantiates the other managers and manages the communication between them.
Package	server.server
Attributes	communicationHandlers: CommunicationHandler[] // Keeps the currently active communication handlers for clients serializers: SerializationManager[] userManager: UserManager
Operations	main(args: String[]) getHandlerForUser(user:User): ConnectionHandler performAction(serializedAction:String):Object // Requests from the client arrives as a serialized string. This method is the entry point of the request. It deserializes the request and delegates it. createUserHandlerForLogin(user:User):ConnectionHandler

Class	**CommunicationHandler**
Description	Manages the relation between users and their assigned server sockets
Package	server.server
Attributes	loggedInUser : User // User that this handler belongs to clientSocket : Socket
Operations	-

Class	**SerializationManager**
Description	An abstract class that defines the needs of a SerializationManager instance. Each SerializationManager subclass need to be able to deserialize the given String and understand the request and

	serialize the result
Package	server.server
Attributes	-
Operations	SerializationManager(sun: SUN): SerialiazationManager deserializeAndPerform(action: String): Object // Method that understands the request and delegates it to other helpers. serialize(object: Serializable) : String

Class	**JSONSerializationManager**
Description	A subclass of SerializationManager, which handles requests that serialized using JSON notation.
Package	server.server
Attributes	-
Operations	-

3.3Logic

Class	**UserManager**
Description	Handles client requests, which are related to user operations like login and signup.
Package	server.logic
Attributes	-
Operations	login(email: String, password: String) : User signup(email: String, password: String, name: String) : Student UserManager(sun: SUN) : UserManager // Only constructor of this manager gets the reference to the SUN object to work.

Class	**NotificationManager**
Description	Handles the logic behind sending notifications to the students.
Package	server.logic.notification
Attributes	-
Operations	sendNotification(notification: Notification): Boolean getNotificationsOfUser(user: Student): Notification[]

Class	**NewsManager**
Description	This manager handles the items that students see on their news feeds.
Package	server.logic.news
Attributes	-
Operations	addFeedItem(feedItem: FeedItem): Boolean getNewsFeed(user: Student): FeedItem[]

Class	**EventManager**
Description	Handles the logic of creating an event and all actions that are performed on events
Package	server.logic.event
Attributes	-

Operations	createEvent(name: String, date: Date, joinable: Joinable, user: User): Boolean
	deleteEvent(event: Event, user: User): Boolean
	// User parameter indicates the user that sends the delete request. Only the creator of the event may delete it.
	joinEvent(event: Event, user: User): Boolean
	leaveEvent(event: Event, user: User): Boolean
	getEventsOfUser(user: User, requester: User): Event[]
	// Some events can only be seen by the participator.

Class	**CheckInManager**
Description	Handles the check-in logic of the application
Package	server.logic.checkin
Attributes	-
Operations	checkIn(user: User, event: Event): CheckIn

3.4 Data

Class	**DatabaseManager**
Description	This abstract class defines the actions that a DatabaseManager instance must handle; data saving and retrieval functions of every action.
Package	server.data
Attributes	-
Operations	login(email: String, password: String): User signup(email: String, password: String, name: String): User saveEvent(event: Event): Boolean deleteEvent(event: Event): Boolean getNewsFeedForUser(user: User): FeedItem[] saveCheckIn(checkin: CheckIn): Boolean addUserToEvent(user: User, event: Event): Boolean removeUserFromEvent(user: User, event: Event): Boolean addUserInterest(user: User, interest: Joinable): Boolean removeUserInterest(user: User, interest: Joinable): Boolean saveUser(user: User): Boolean // Saves the changes that have been made on the user instance findUser(keyword: String): User[] addFriend(adder: User, target: User): Boolean removeFriend(remover: User, target: User): Boolean getNotificationsOfUser(user: User): Notification[] addNotificationToUser(notification: Notification, user: User): Boolean getJoinable(id: int): Joinable

Class	**HibernateDatabaseManager**
Description	A subclass of DatabaseManager which uses Hibernate to handle database operations.
Package	server.data
Attributes	-
Operations	-

Class	**User**
Description	An abstract class that represents minimum requirements of a User object
Package	server.data.model
Attributes	id: int email: String password: String name: String avatar: String contactInfo: String lastLogin: Date
Operations	-

Class	**Student**
Description	A subclass of User that keeps student informations
Package	server.data.model
Attributes	friends: Student[] events: Event[] interests: Joinable[] points: int notifications: Notification[] department: String // Departments of the Students will help the system to offer events and friends to them
Operations	-

Class	**ClubRepresentative**
Description	A subclass of User which keeps information about club representatives.
Package	server.data.model
Attributes	represents: StudentClub numberOfPosts: int validUntil: Date // Representative accounts may expire at the end of the semester.
Operations	-

Class	**Joinable**
Description	An abstract class, which keeps basic information about concepts that, a Students can join or track.
Package	server.data.model
Attributes	id: int name: String info: String followerCount: int latestEventAdded: Date // In order to sort the Joinable
Operations	-

Class	**Course**
Description	A subclass of Joinable, which represents a course (e.g. CS101).
Package	server.data.model
Attributes	code: String
Operations	-

Class	**StudentClub**
Description	A subclass of Joinable, which represents a student club (e.g. IEEE).
Package	server.data.model

Attributes	-
Operations	-

Class	**Hobby**
Description	A subclass of Joinable, which represents a hobby that may be done on campus (e.g. Biking).
Package	server.data.model
Attributes	-
Operations	-

Class	**FeedItem**
Description	An abstract class that represents everything that can be displayed on users' news feeds.
Package	server.data.model
Attributes	id: int date: Date creator: User showCount: int // How many students has seen this item
Operations	-

Class	**News**
Description	A subclass of FeedItem that keeps news about events and activities.
Package	server.data.model
Attributes	text: String
Operations	-

Class	**Event**
Description	This class holds information about events that students can join.
Package	server.data.model
Attributes	id: int name: String joinable: Joinable // subject of the event, such as CS101 or Biking joinedStudents: Student[] eventInfo: String owner: Student displayCount: int
Operations	-

Class	**Notification**
Description	Keeps the information of notifications that the students will see.
Package	server.data.model
Attributes	id: int

	source: User target: Student type: NotificationType date: Date data: String reached: Boolean opened: Boolean
Operations	-

Class	**CheckIn**
Description	Model object that keeps the information of a checking including rewarded points.
Package	server.data.model
Attributes	id: int student: Student event: Event date: Date rewardedPoints: int
Operations	-

3.5 Mobile

Class	**SUNApp**
Description	Main class of the Android application.
Package	mobile.controller
Attributes	-
Operations	handleNotification(notification: Notification) // When a push notification arrive at the mobile application this method will be invoked to display it to the user

Class	**LoginPage**
Description	Displays the username and password inputs to get credentials of the user and tries to login with the given information using the SUNClient
Package	mobile.controller
Attributes	username: EditText password: EditText loginButton: Button signupButton: Button
Operations	loginPressed() toSignup()

Class	**SignupPage**
Description	Handles signup process by asking the name and the credentials of the user.

Package	mobile.controller
Attributes	name: EditText username: EditText password: EditText signupButton: Button backButton: Button
Operations	back() signup()

Class	**ProfilePage**
Description	Displays personal information of any user.
Package	mobile.controller
Attributes	events: Event[] user: User list: ListView // Changes the displayed events depending on the users' choices.

Operations	showMyEvents()
	// Shows the events that the user marked as he will join
	showNextEvents()
	// Shows the upcoming events of the Joinables that the user follow
	showPastEvents()
	// Displays the events that the user participated in.
	showContactInfo()
	changeAvatar()
	// Show picture selection dialog
	setAvatar(file: Image)

Class	**MainPage**
Description	This is the dashboard page of the application, which will be opened when the application launches.
Package	mobile.controller
Attributes	showProfileButton: Button showEventsButton: Button findFriendsButton: Button bumpButton: Button
Operations	showProfile() showEvents() findFriends() bump() // Opens the dialog that allows users' to bump their phones to check in with a friend.

Class	**JoinableDetailPage**
Description	Displays details of a Joinable instance.
Package	mobile.controller
Attributes	name: TextView

	info: TextView eventList: ListView events: Event[] createEventButton: Button backButton: Button
Operations	showCreateEventPage() showInfo() back()

Class	**CreateEventPage**
Description	This page collects necessary information to create a new event about a Joinable
Package	mobile.controller
Attributes	joinable: Joinable name: EditText info: EditText saveButton: Button backButton: Button

Operations	showConfirmation()
	createEvent()
	// Sends the new event information to server and tries to create the event
	back()

Class	**SearchInterestsPage**
Description	This is the page where users follow or un-follow Joinables.
Package	mobile.controller

Attributes	filterBox: EditText
	// The textbox that users enter keywords to filter the results
	typeSelection: SegmentedButton
	// Allows users to select the kind of Joinable that they want to follow, selections are; Course, Club and Hobby
	list: ListView
	backButton: Button

Operations	showCourseList()
	showClubList()
	showHobbyList()
	addInterest(interest: Joinable)
	// This method will be called when the user select a Joinable to follow
	removeInterest(interest: Joinable)
	back()

Class	**FindFriendsPage**
Description	This page allows users to find and add new friends to their friend lists.
Package	mobile.controller

Attributes	filterBox: EditText // An input box that will be used to enter a keyword to find a friend. The keyword may be the part of the name or the username of the friend. list: ListView searchButton: Button back: Button

Operations	searchFriend(keyword: String)
	// Sends a request to find relevant Students for the keyword and updates the list according to the results
	sendFriendRequest(target: Student)
	back()

Class	**SUNClient**
Description	A singleton class that handles the communication between the mobile application and the server. This class implements the SUNActionPerformer interface and can handle every possible request that the application components need to make.
Package	mobile.controller
Attributes	instance: SUNClient
Operations	getInstance(): SUNClient

Class	**User**
Description	An abstract class that represents minimum requirements of a User object
Package	mobile.modelproxy
Attributes	id: int email: String name: String avatar: String contactInfo: String
Operations	-

Class	**Student**
Description	A subclass of User that keeps student informations
Package	mobile.modelproxy
Attributes	friends: Student[] events: Event[] interests: Joinable[] points: int notifications: Notification[]
Operations	-

Class	**ClubRepresentative**
Description	A subclass of User which keeps information about club representatives.
Package	mobile.modelproxy

Attributes	represents: StudentClub
Operations	-

Class	**Joinable**
Description	An abstract class, which keeps basic information about concepts that, a Students can join or track.
Package	mobile.modelproxy
Attributes	id: int name: String info: String
Operations	-

Class	**Course**
Description	A subclass of Joinable, which represents a course (e.g. CS101).
Package	mobile.modelproxy
Attributes	code: String
Operations	-

Class	**StudentClub**
Description	A subclass of Joinable, which represents a student club (e.g. IEEE).
Package	mobile.modelproxy
Attributes	-
Operations	-

Class	**Hobby**
Description	A subclass of Joinable, which represents a hobby that may be done on campus (e.g. Biking).
Package	mobile.modelproxy
Attributes	-
Operations	-

Class	**FeedItem**
Description	An abstract class that represents everything that can be displayed on users' news feeds.
Package	mobile.modelproxy
Attributes	id: int date: Date creator: User
Operations	-

Class	**News**
Description	A subclass of FeedItem that keeps news about events and activities.
Package	mobile.modelproxy

Attributes	text: String
Operations	-

Class	**Event**
Description	This class holds information about events that students can join.
Package	mobile.modelproxy
Attributes	id: int name: String joinable: Joinable // subject of the event, such as CS101 or Biking joinedStudents: Student[] eventInfo: String
Operations	-

Class	**Notification**
Description	Keeps the information of notifications that the students will see.
Package	mobile.modelproxy
Attributes	id: int source: User target: Student type: NotificationType date: Date data: String
Operations	-

Class	**CheckIn**
Description	Model object that keeps the information of a checking including rewarded points.
Package	mobile.modelproxy
Attributes	id: int student: Student event: Event date: Date rewardedPoints: int
Operations	-

4. References

1- http://hibernate.org/orm/[2]
2- http://searchmobilecomputing.techtarget.com/definition/IEEE-802-Wireless-Standards-Fast-Reference