Time Traveler

Application Specs:

Name:	Time Traveler
OS:	iOS (6.1, 6, 5.1, 5, 4.x, 3.x)
Frameworks:	CoCoa Touch/Core Services
	NSUserDefaults
	XCTest
IDE:	xCode
Language:	Objective C
Coding Style:	Google Objective C Style Guide

Motivation:

With the advances in current technologies travelling across time zones has become a vital part of many professions and vacation plans. NASA Astronaut and OSU Alum Don Pettit is no stranger to that since an integral part of his work consists of travelling across many time zones at a time. As a result, he and many other travelers need an application that assists in managing jet leg.

Hugh McDonald, Brandon Straley, Maria Powell, Sean McDonald, Zac Carlson, Robert Combs, Nathan Christopher

Table of Contents

I Introduction

Purpose & Goals

The purpose of this document is to present thorough documentation on the application's intended function, implementation, requirements, and architecture. It will not only provide a clear and concise walk-through on how Time Traveler application was created but it will furthermore offer a guide as how to install and use the final product.

General Description

Time Traveler is an iOS application that assists travelers in managing their jet lag when passing through multiple time zones. The assistance models the modern approach recommended by leading medical doctors, and spans the three stages of travel: before the trip, during the flight, and after arrival.

Specific/User Description

The traveler will be able to input his/her trip details including departure date, destination time zone, and the current sleep schedule. Based on the provided information, the application will then output a schedule and frequent notifications that the traveler can follow in order to avoid jet lag symptoms.

II Process

Development Process

Our team followed the extreme programming (XP) methodology for the Time Traveler project. We knew from the beginning that we would need to use a development technique that can accommodate ever-changing software requirements, and has frequent development cycles. The following sections are all written in the context of XP.

Iterative Development

Iterative development is defined by diving the development into short, cyclic iterations with an ultimate goal of adding agility to a project with ever changing requirements. The team scheduled meetings with the customer at a bi weekly rate and from there reevaluated coming iteration goals.

Refactoring

The big overarching idea of XP is to keep design decision dynamic. Hence, we did not follow a static flow of requirements set at the beginning of the project. Instead, we constantly adapted to changes throughout the entire project. We frequently refactored code to accommodate design revisions.

Testing

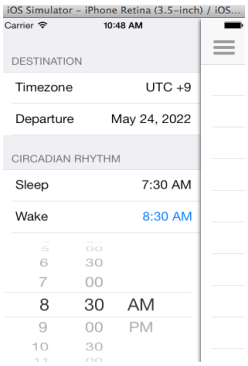
The project followed a test driven development approach, which means that we have many test cases to account for all code functionality. Due to the size of the project we decided to host both automated code based tests as well as manual tests.

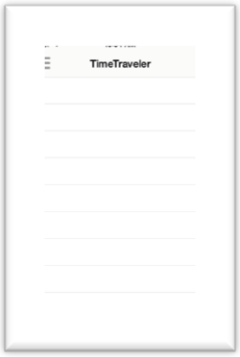
We do not have an extensive UI in this project, which is why it did not make sense to set up automated tests for the interface. Below is an example of a manual tests we created during one of the iterations.

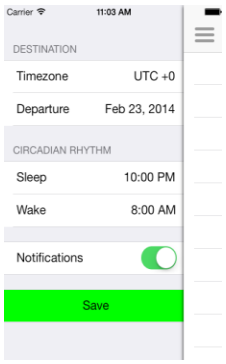
Manual UI Unit Testing

Test: The user is able to input travel time information. The information only gets saved once the button is pushed but is not preserved otherwise.

1. Insert information into the UI
2. Enter the home screen without pushing the save button
3. Go back to the insert screen and check that the information is back to the default values.







Collaborative Development

The XP methodology demands design decision to be a collaborative effort where no one person makes all of the design decisions. While we had group leaders who coordinated meetings and time schedules, everything was decided as a team. Additionally, we made use

of pair programming in order to divide up coding tasks. Pair programming is set up in a way that two programmers sit side by side at a computer. One person takes on the thinker role. That is usually the person not typing, whereas, the thinker role ensures that possible bugs are eliminated and keeps the typist in check. Furthermore, the thinker could and should tell the typist what to program. Hence, the typist is the one who is doing the coding itself. The roles should be switched regularly.

We followed the pair programming approach by taking a verbal poll on member's special interest. We had three teams within our group. We have a model team, a testing team and a user interface team. At first, the teams stayed the same and worked on their assigned part during our group meetings. Of course we would still all meet as a group and discuss issues that arose in particular teams.

In the latter part of the project development we changed up the teams and we had everyone work on different tasks.

III Requirements & Specifications

Use Cases

Actor	Task-Level Goal	Priority
User	Create a Jet Lag coping schedule	1
Calendar	Synchronize the created schedule with the user's schedule.	2

Actor	Goal	Brief
User	Create Trip Schedule	Input trip information, which is analyzed and processed, resulting in a Jet Lag coping schedule, which is then presented to the user.
Calendar	Synchronize Jet Lag Coping Events	Merge Jet Lag coping Events with the user's calendar.

Casual Use Cases:

Casual Use Case 1: User schedule creation

The user inputs their trip location, date, and time, for departure and arrival, into the areas provided. The trip information is checked for valid dates, times, and locations internally. The trip is then analyzed and processed according to the recommended activities to counter Jet Lag, resulting in a Jet Lag coping schedule. This schedule is then presented to the user in a format that is familiar and useful.

Casual Use Case 2: Calendar Synchronization

The calendar connects with the application for synchronization. The calendar then receives the Jet Lag coping event schedule calculated and attempts to merge them with the user's existing calendar. Any conflict between the user's prior scheduled events and the newly recommended schedule are resolved by moving the Jet Lag coping events to the nearest available time. The schedule is then updated in the application.

Fully Dressed Use Cases:

Fully Dressed Use Case 1: User Schedule Creation

Primary Actor: User

Goal in context: To create a Jet Lag coping schedule for an upcoming trip.

Scope: Time Traveler Application: User Schedule Creation – The user inputting information into the application, the application processing the information, and creation of a Jet Lag Schedule.

Level: User-goal

Stakeholders and interests:

User: Wants to get a schedule that is easy to read and works coherently with previous events already scheduled.

Preconditions: none

Minimal guarantees: The creation of a schedule that has times and dates for events in which will help with the coping of Jet Lag.

Main success story:

Trigger: User begins using application

1: User inputs information into the application

2: The application verifies the information

3: The application analyzes the information

4: The application creates a schedule

5: The application reports the schedule to the user

Fully Dressed Use Case 2: Calendar Schedule Synchronization

Primary Actor: Calendar

Goal in context: To synchronize the Jet Lag coping schedule with the user's existing schedule, resolve any conflicts found, and change the schedule in the application accordingly.

Scope: Time Traveler Application: Calendar Synchronization – The application synchronizing with the users calendar to further cohesion between app and real world.

Level: User-Application cohesion.

Stakeholders and interests: User

User: Wants to have a cohesive union between Time Traveler app and the real world.

Preconditions: Event data is valid

Minimal guarantees: The user will have their calendar and the schedule created by the application synchronized.

Main success story:

Trigger: The schedule is created

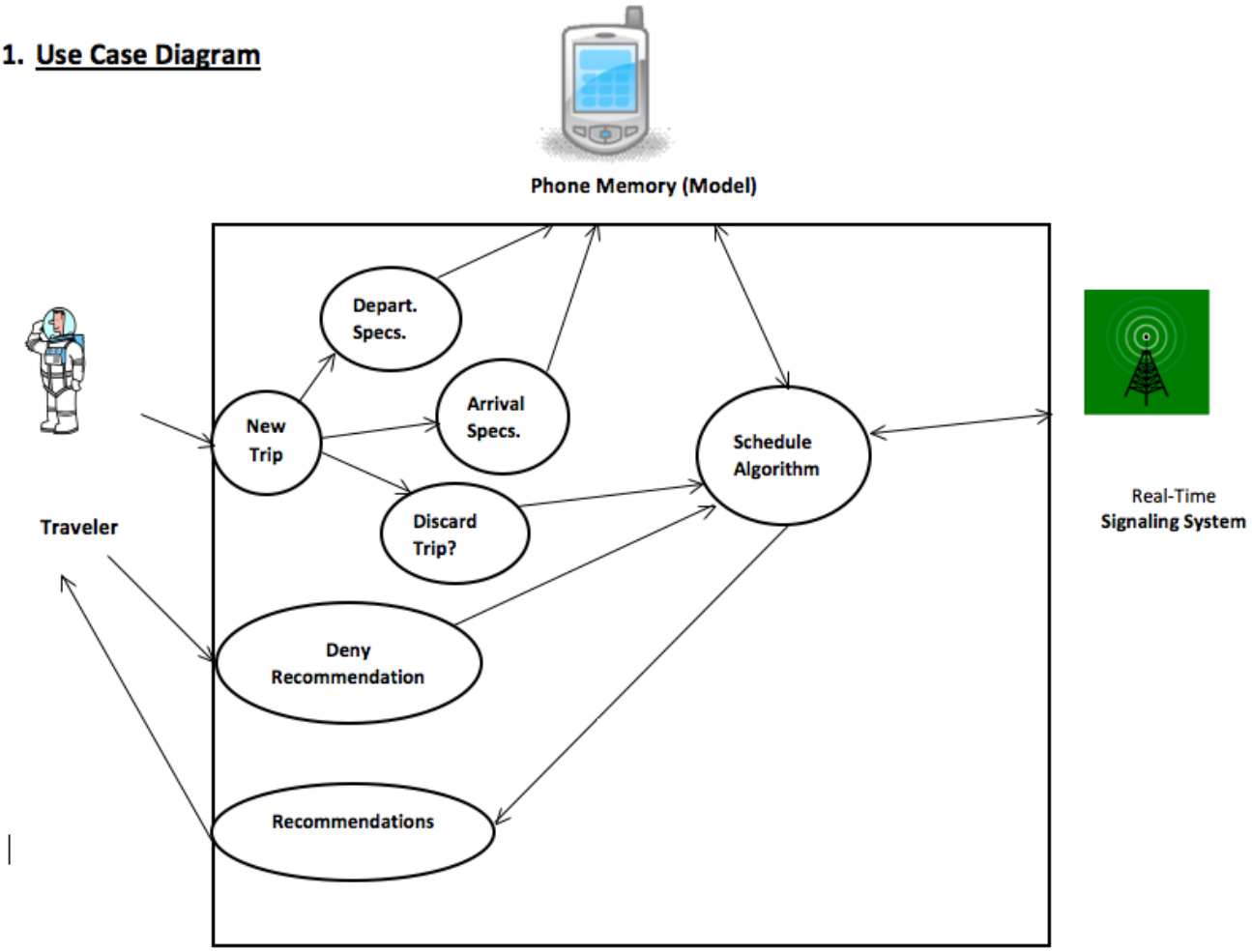
1: The application gets the users schedule from the calendar

2: The application processes the calendar and checks for conflicts

3: The application resolves conflicts

4: The application merges the new schedule with the calendar

1. Use Case Diagram



User Stories

These user stories are organized in a top down approach where the overarching user story is listed first and from there more specific user stories are listed sequentially.

I. As a user I want a schedule and notifications to help manage Jet Lag

1. As a user I wish to be able to input my departure location, destination, and departure date, as well as my normal sleep and wake times, and whether or not I wish to receive notifications.
2. As a user I want my phone to save the state of the app so that (for example) in the event of a phone-call, when I return to the app I do not lose any information.

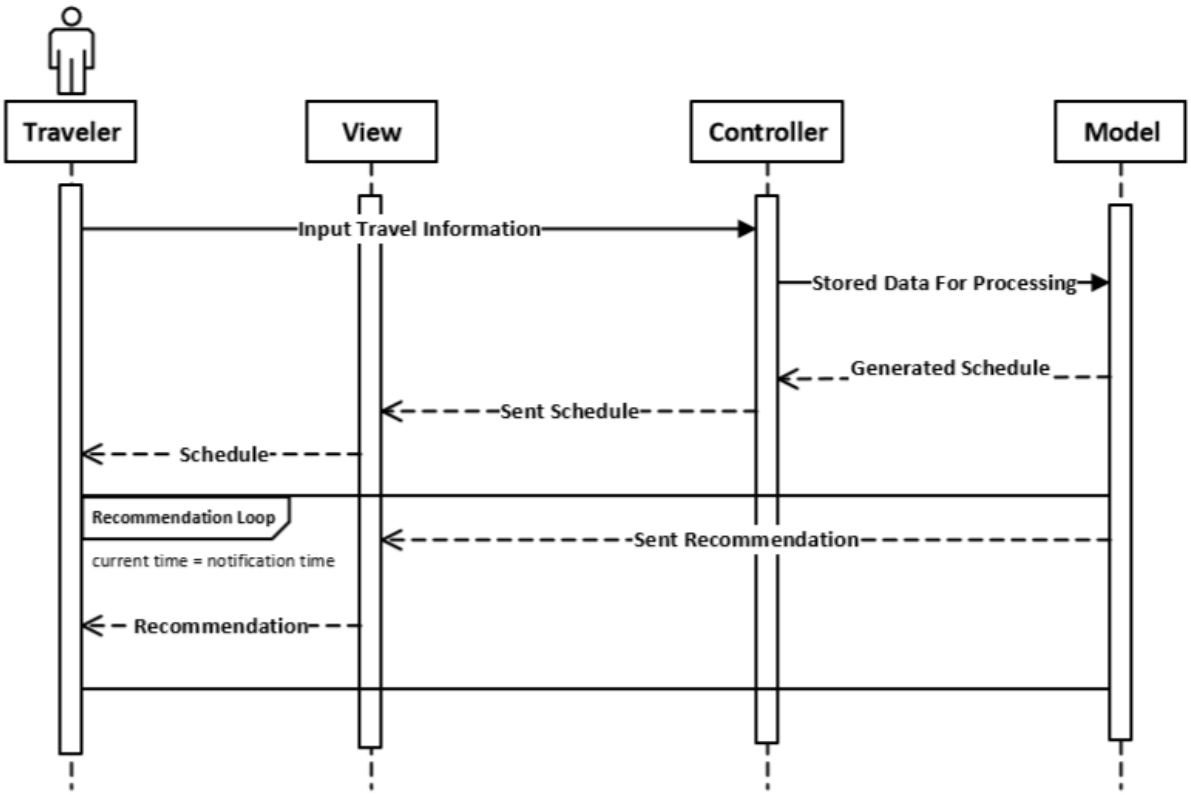
As a user I wish to see my Jet Lag management schedule

- As a user I wish to see my schedule in list format
- As a user I wish to see when to sleep and when to stay awake
- As a user I wish to see when to avoid or seek light
- As a user I wish to see when to take Melatonin

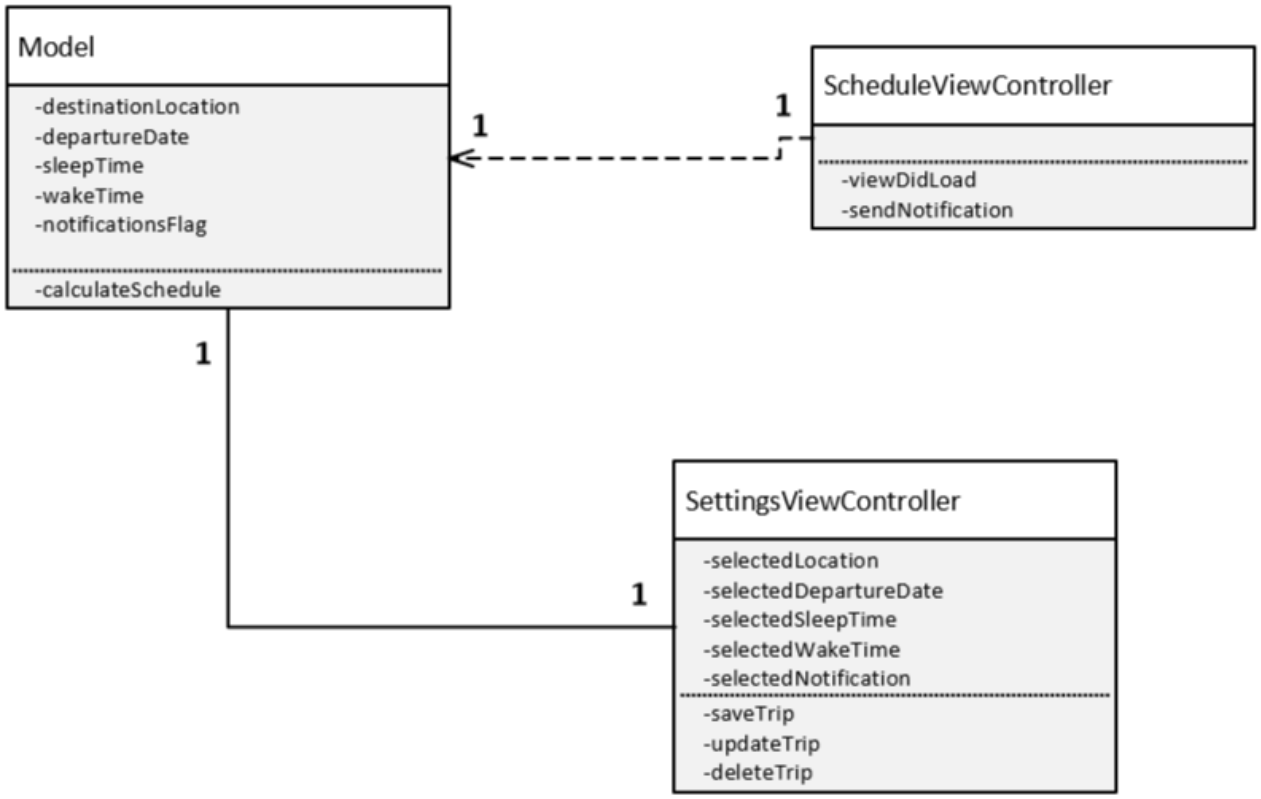
3. As a user I wish to receive notifications based on my schedule
    - As a user I wish to be reminded when to sleep and when to stay awake
    - As a user I wish to be reminded when to avoid or seek light
    - As a user I wish to be reminded when to take Melatonin

IV Architecture & Design

Sequence Diagram

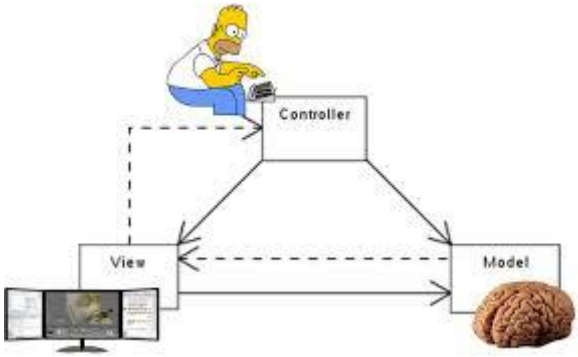


Class Diagram



Big Picture (Top Down)

In iOS development, an environment that involved a touch screen and a single user the MVC design pattern the best fit and mandatorily enforced. MVC stands for Model, View, and Controller. Its functionality can be explained by this simple picture:



Model: Notifies view(s) and controllers when there is a change of state.

View: Generate a user interface based on what the controller communicate

Controller: User interacts with the controller which in turn sends commands to the model and possibly to the view

Major Components

Our majors components are the model, view, and controller as mentioned above. The model manages our schedule algorithm as well as the data. The view generates the interface to our user and provides them a way to input and view otherwise raw data. Controller takes the user input and then hands of that information to the model.

Frameworks

We used three different frameworks within our application.

The iOS Cocoa Touch framework provides special GUI controls, buttons, and full-screen views. It also matches UIKIT which is another framework that we made use of. UIKIT gives you a comprehensive collection of HTML, CSS, and JS components. It made it very intuitive for us to create a user interface. For testing purposes we used XCTest which provided a very user friendly way to run tests and see if they pass or not.

V Future Plans

Personal Reflection

All in all, we are very happy with the development of the project in the time frame that was given to us. XP has provided us with a very efficient methodology to develop a fully functional and most of all dynamic application. While none of us had any experience with iOS development, we all feel like we have gained valuable skills.

Future Plans

Our team would like to add the following features to the application in the future:

Destination location: We would like the destination location picker to be more intuitive. Additionally, allowing destination selection from a map.

Data Visualization: We would like to add a more professional way to represent our data. We would ideally have two modes of data representation marked by portrait and landscape view. It is our idea to add a clock in portrait mode which shades the hours of the various events. In contrast, we would show the entire schedule as a time line in landscape mode.

Info Button: We would furthermore like to add an "Info Button". The purpose of this button is to provide the user with more information on how to prepare for sleep or wake time. For example, if the user is supposed to sleep in two hours, our app could inform them when to put on glasses to avoid blue light or when to take in melatonin.

The Time Traveler application has been created to be very dynamic which allows future developers to add many features.

VI Appendix

Installation

As mentioned above, the software is an iOS application and will be available for download on the app store onto any Mac product such as tablets, phones and laptops. More specifics on software requirements are listed below:

- iPhone 3G or higher
- 2nd generation iPod touch or higher
- any iPad model
- iTunes 7.7 or higher
- iOS 2.0 or higher

First, go to the app store in order to enter a search engine to find the Time Traveler application. Once you have found the Time Traveler application in the store, you are able to install it on your device without charge.

Operations Manual

Start the app by tapping on the Time Traveler icon. In order to get to the settings page either swipe right or tap the menu button. Now you are on the schedule-input page. From here you can enter your travel information to generate your schedule. Your travel information includes the destination time zone, departure date, sleep time, wake time, and notifications options. You are able to reset all of the information including your current schedule. The save button needs to be pushed for a schedule to be generated. Without the button press your travel information will not save. You can swipe the screen back to the

home screen which will now display the schedule to follow. Notice that the top picture will update based on if you should be sleeping. If you should be sleeping, it will show a night sky. If you are supposed to stay awake, it will display a day picture. If nothing is currently planned it will show an airplane. This happens the day of travel as well as when you have not set a schedule.