6-4-2015

Dynamic life management assistant (DyLMA)

Arturo Aguilar
Santa Clara University

Ruben Luva
Santa Clara University

David Mora-Barajas
Santa Clara University

Sunny Patel
Santa Clara University

Alejandro Rodriguez
Santa Clara University

SANTA CLARA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: June 4, 2015

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Arturo Aguilar
Ruben Luva
David Mora-Barajas
Sunny Patel
Alejandro Rodriguez

ENTITLED

Dynamic Life Management Assistant (DyLMA)

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Thesis Advisor

6/9/15

Department Chair

Dynamic Life Management Assistant (DyLMA)

by

Arturo Aguilar
Ruben Luva
David Mora-Barajas
Sunny Patel
Alejandro Rodriguez

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 4, 2015

Dynamic Life Management Assistant (DyLMA)

Arturo Aguilar
Ruben Luva
David Mora-Barajas
Sunny Patel
Alejandro Rodriguez

Department of Computer Science and Engineering
Santa Clara University
June 4, 2015

ABSTRACT

Individuals currently face difficulty allocating time to achieve an ideal level of productivity. It is often demanding to manage tasks and responsibilities along with their specific priority level and deadlines. Likewise, managing health and wellness priorities while under the stress of a busy schedule can be just as challenging. The productivity tools of today offer a fragmented mobile experience that relies on a multitude of applications to achieve simple goals. A viable solution is a life-management system for Google Glass that integrates a user's schedule in order to aid productivity and time management. This system manages a series of everyday tasks and calendar events, while as the same time promoting a healthy lifestyle by offering suggestions based on availability and user-defined priorities. The system allows notifications to be displayed conveniently within a user's field of view, ultimately leading to more productive and healthy individuals.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Background and Problem Statement

Individuals face difficulty allocating time for maximum productivity. It is often demanding to manage tasks and responsibilities with physically written reminders, especially if a schedule is saturated with activities. Humans are error-prone; forgetting one task can spawn numerous scheduling conflicts. Physical reminders, such as sticky notes, have spatial limitations and are a nuisance to deal with. For this reason, mobile devices are used instead of these physical reminders. However, mobile applications currently used to keep track of events or tasks cannot prioritize and suggest the most optimal solutions to scheduling issues. Current apps are also incapable of considering an individual's needs other than a task list, such as their personal health.

1.2 Current Solutions

The productivity tools of today offer a fragmented mobile experience that relies on many applications to achieve simple goals. Relying on an iPhone or Android is inconvenient because the device remains in the user's pocket for the majority of the day. Today's apps allow users to view schedules when opened but do not provide guidance on accomplishing tasks. For example, when a user sees a planned dinner period in their schedule, another source such as an application must be used to find food locations. Currently, an intelligent productivity tool that can offer recommendations based on location, daily activities, and important events does not exist. Relying on tools that are spread across multiple services and applications ruins the productivity that could be achieved by a more unified experience.

Today, the only solution is to settle with a combination of multiple products on the market to keep track of schedules. For example, a user must use a calendar app to view his or her schedule as well as use a reminders app to view tasks that need to be completed. There is no widespread,

convenient way to weave these actions despite the similarities they share. Upon deciding on what activity the user will be working on, the user must then switch apps to find information, usually through a map, store, or internet search app. This disconnect only adds to the inconvenience for users due to the multiples transitions between different apps. Current software lacks the proper implementation to offer the functionality that a user wishes he or she could have with such a utility. Often, personal assistants are marketed as having the ability to make everyday tasks less tasking, but ultimately leave much to be desired in the core functionality.

1.3 Proposed Solution

Our solution is to provide a dynamic life-management tool on the Google Glass platform. This tool will not only manage an itinerary for everyday tasks, it will offer suggestions as to what should be done during free time. This ability to suggest will help users stay on task with all duties, including those which are not time specific. This not only solves the issue of having two separate applications to manage calendar schedules and to-do lists, it interweaves their functionality. The personalized assistant will learn from the users' activities, such as their daily tasks, and adjust the priority of activities accordingly. In this way, it can help users complete tasks based on order of importance. Additionally, our solution will help ensure the well-being of the users beyond efficient time management. For example, the app will help promote a healthy lifestyle by encouraging the users to include exercise periods into each day.

The solution thus offers a suggestion system, based on the user's most recent activity list, in order to provide an efficient usage of time. This is achieved by having a user-friendly experience, where notifications can be viewed directly within a user's line of sight. Having the ability to have a unified experience, tears away the psychological barriers tied to the stress-inducing task of managing a schedule, leaving only the productivity aspect to benefit.

Chapter 2

Requirements

We first established a list of functional and nonfunctional requirements based on what was deemed necessary for the system. Functional requirements define the behaviour of the system. They establish what the system will do. Non-functional requirements define the manner in which the system accomplishes the functional requirements. They define the performance and feel of the system.

2.1 Functional Requirements

1. The system will add calendar events by syncing with user's calendars.
   (a) The data will be drawn from Google Calendar.
2. The system will add user tasks by syncing with user's task list applications.
   (a) The system will support Google Tasks.
3. The system will provide dynamic suggestions for the schedule based on user's inputted events.
   (a) The user will be given options which he/she may accept or refuse.
4. The system will provide suggestions based on health goals, such as physical activities.
5. The system will provide additional help for suggestions.
6. The system will generate reminders to keep the user on track.

2.2 Non-functional Requirements

1. The system will be consistent with Google Glass design standards.
   (a) Will not confuse users with an unfamiliar setting

2. The system's use will be intuitive.
    (a) Easy to read and use
3. The system will be responsive.
    (a) The system will minimize load delays and information will appear quickly in response to user interactions.
4. The system will be visually appealing.
    (a) Users should not be distracted by the application, nor should they have difficulty distinguishing important aspects.
5. The system will be secure.
    (a) Users should not be distracted by the application, nor should they have difficulty distinguishing important aspects.
6. Due to the nature of requesting personal data, the system will protect imported information.

2.3 Design Constraints

The system is required to run within a certain environment and with certain limitations. These limitations may take the form of system application limitations, such as caps on usable memory, or limitations on available tools, such as the limited pool of networking libraries. These limitations put a constraint on the design options developers when creating a system. Our Dynamic Life Management Tool must run on Google Glass and be compatible with Google Calendars. It must be able to interact and work in conjunction with a smartphone application running on an Android based Operating System.

Chapter 3

Design Concept

3.1 Conceptual Model

As the system will be displayed on the Google Glass screen, it is necessary that all screens be simple and easy to read. The Glass design imperative is to use white text on a semi-transparent black in order to maintain partial visibility. Information is to be presented with sparse detail unless a user requests more data. With this in mind, the design has two home screens which indicate the status of the user.

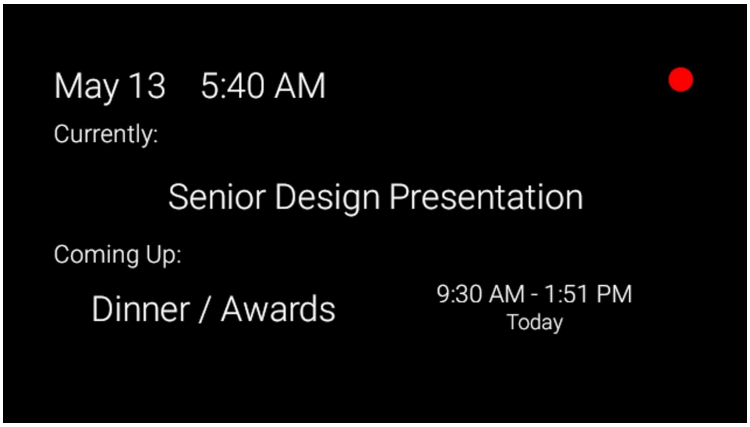

Figure 3.1: Busy State

Figure 3.1 represents one of two possible home screens for the system. This figure demonstrates when a user's status is set to busy; the screen shown is that of a busy status. In the busy status, the user is given little information. Instead, the system presents a condensed, low detail format. This low detail format provides only the necessary details for their current, non-flexible event. This prevents the user from becoming distracted and helps them to stay productive. The current date and time is shown in the top left corner. The status is clearly indicated in the top right hand of the screen as a red light. In the center of the screen the user can see the current event.The status is set

to busy whenever there is an event scheduled during the current time. The bottom portion of the screen shows a quick summary of the next upcoming event.

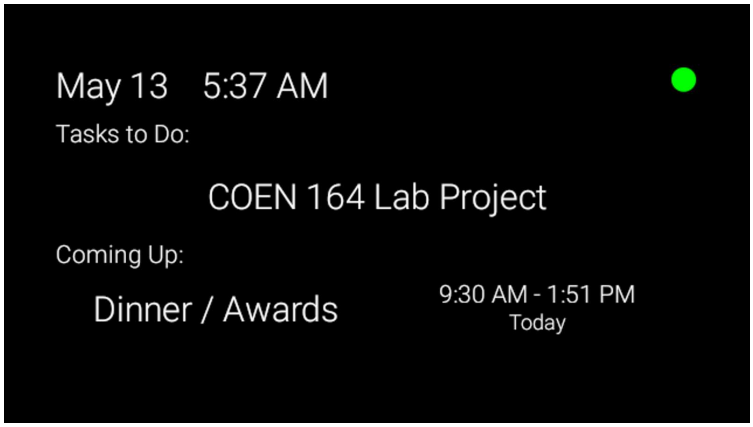

Figure 3.2: Free State

Figure 3.2 represents the other possible home screen for the system. In Figure 3.2, the user's status is free and therefore the screen shows a free status. This screen differs in content from that of the busy status home screen by presenting a suggested Task. One of the similarities to the busy state home screen is that the top left corner of the screen still shows the current date and time. However, the top right corner of the screen now displays the current status as free, as indicated by the green light. When the user is free, the system presents tasks in the middle of the screen.

The idea of having different views is that when the user is not busy, he or she still has different tasks or assignments that need to be completed. These tasks or activities could range from completing a homework assignment to reminding the user to go exercise. The list of activities is presented one at a time and each has three options: to delay the task and see what other tasks are present, to state that the user has completed the task, or to permanently remove the task. The bottom portion of the screen shows what upcoming event is next and at what time it will begin so that the user can gauge how much time he or she has available to complete the required tasks.

Figure 3.3 represents a more detailed view of an event. The user can navigate to the screen in Figure 3.3 from the busy state or free state home screens. From the busy state, the user can access the details of the current event and the next upcoming event. From the free state, the user may only access information about the next upcoming event. The home screen merely shows the title of the current event; in this screen, the user is presented with more details about the event that could otherwise not fit in the home screen. From here, the user can go back to the home screen.

Figure 3.4 represents a more detailed view of a task. The user can only navigate to the screen

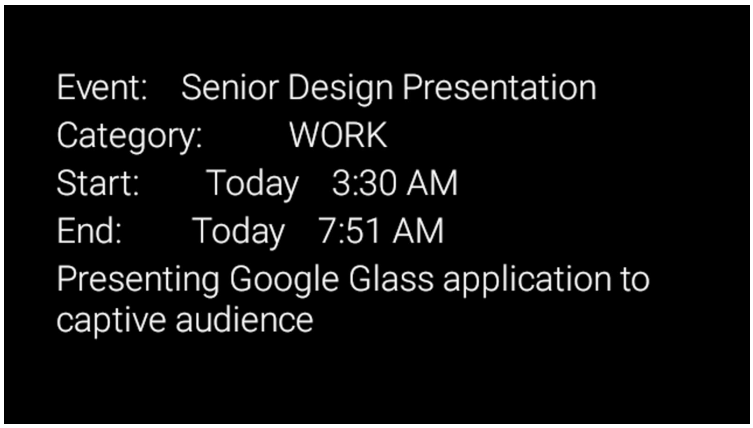

Figure 3.3: More Information for an Event

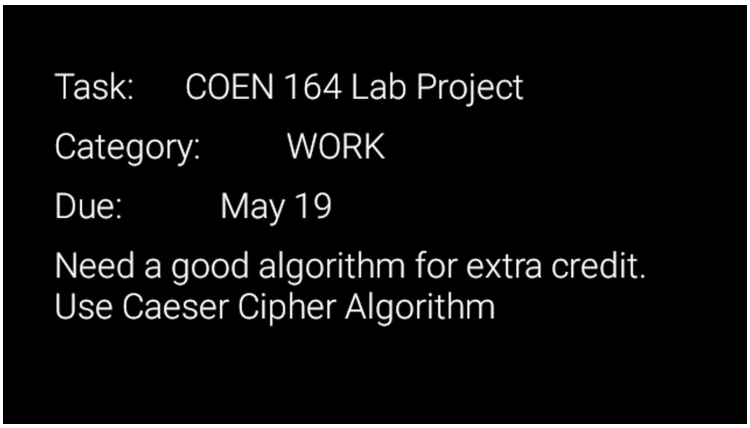

Figure 3.4: More Information for a Task

shown in Figure 3.4 from the free state home screen. The free state home screen shows a task for the user that he or she could select to accomplish. In this screen, the user is presented with more details about the task that cannot fit in the free state home screen. This screen tells the user what needs to be done to complete the task and gives a deadline for when the task needs to be completed. This screen has a set of options for dealing with the given task. The user has a completed option which allows the user to state that the task has been completed. Other options include "remove", which permanently removes the task from circulation, and "skip", which delays this task temporarily and presents a different one. The user can go back to the free state home screen by swiping down on the glass.

3.2 Use Cases

Use cases in our system describe the management of schedules and system recommendations by the user. These use cases define the series of steps required to accomplish our functional requirements. We have identified five use cases to address the functional requirements defined on a previous page. Figure 3.5 gives a high-level view of all use cases defined by our system.

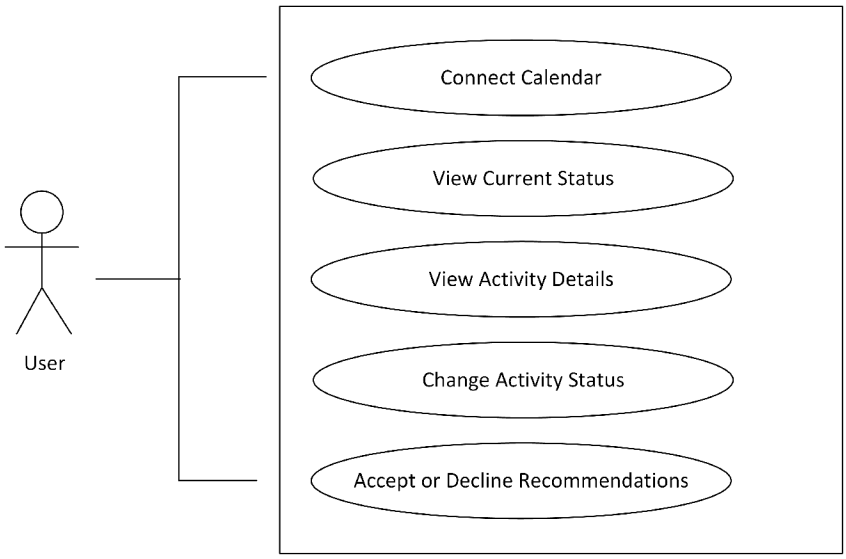

Figure 3.5: Use Case Diagram

The following enumerated list describes each use case in greater detail.

1. **Accept or Decline Recommendation**

   Goal: Change the schedule based on acceptance or rejection of the system's suggestion.

   Actor: User

   Precondition: The system has attained enough information from the user's schedule to intelligently create suggestions.

   Postcondition: The system modifies the schedule based on the implemented modification algorithm.

   Steps:

   (a) User clicks accept or decline when the suggestion notification pops up.

   Exceptions: The system does not know enough about the user to suggest schedule improvements.

2. **View Event Details**

   Goal: The user receives detailed information on a scheduled event rather than just viewing the event's basic information.

   Actor: User

   Precondition: The user has events on their schedule with added detailed information attained through the companion app.

   Postcondition: The user knows more detailed information on a scheduled event.

   Steps:

   (a) Navigate to status screen.

   (b) Highlight event of interest by swiping through events.

   (c) Tap the touchpad to access detailed view.

   Exceptions: None

3. **View Current Status**

   Goal: Allow the user to view his or her current status in regards to schedule. Either Busy or Free.

   Actor: User

   Precondition: User has added data to the calendar and/or user has added tasks he or she would like to accomplish.

   Postcondition: User sees what status he or she is currently in. User will also know if he or she has events coming up.

   Steps:

   (a) Open the Application.

   (b) Look to the top right hand corner of the Screen.

   (c) Home page shows current status of user and shows relevant information based on the current status.

   Exceptions: None

4. **Manage Events**

   Goal: Allow the user to be in control of scheduled events and list of tasks he or she wants to accomplish.

Actor: User

Precondition: The user must have already sync their calendar events.

Postcondition: The user would have added additional information, such as priority and deadline to each task.

Steps:

(a) Users would open the Android companion application in order to view their sync schedule.

(b) They would select a specific task or event and add additional information

(c) They would then save the information and let the Google Glass application resync the new information.

Exceptions: None

5. **Connect Calendar**

Goal: Connect system with Google Calendar to sync.

Actor: User

Precondition: The user must have a Google Calendar with events.

Postcondition: Google Calendar has synced with the system.

Steps:

(a) Log into Google Calendar.

(b) Give our system access to view calendar events.

Exceptions: None

Chapter 4

Design Architecture

4.1 Activity Diagram

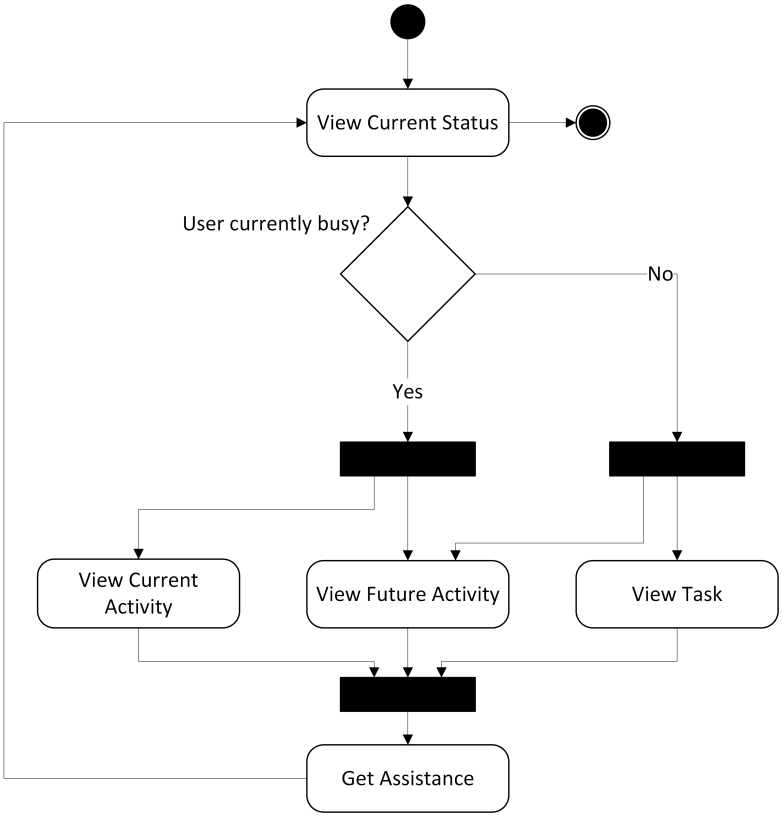

Figure 4.1: User Flow Chart

An activity diagram describes how an end user moves through the system. Figure 4.1 describes the user interactions of a user who has already downloaded the application to their Google Glass device, completed the import and synchronization processes, and is now actively using the application.

The activity diagram in Figure 4.1 shows that users begin by viewing their current status. This status will provide them with some preliminary useful information, at which point the user may be satisfied and end the cycle. As the user progresses down this figure, the user is given more detailed information related to a task. If the user desires more information, their next step is dependent on the system's status. If the user is currently busy, they can either view a future activity or the current activity. If the user is not busy, the user can either view a future activity or review a task. When viewing the details of an activity or task, the user may decide that they want additional assistance. The user could then progress further, which would call the most relevant assistance tool available, such as the Google Maps tool.

4.2 Sequence Diagram

A sequence diagram depicts the interactions between modules. Figure 4.2 is the sequence diagram of this system. The interactions of the system will be launched by the user's interactions with the status screen. This status screen will load different events based on whether the user is currently busy. First, the system will queries the activity section on the status of the user's schedule. The status screen will then be requesting information from the activity and task data structures. If the system receives a busy indication, it requests only two activities. These two activities are the current activity and the next major activity. If the user is not busy the system will request the next activity and a task. The activities and tasks given are determined within the class data structures. A user could then request details. If the details requested are on an activity object, the Activity class is called to present further details. If the details are requested for a task, the Task class is called to present further details. From the detail view, a user may request further assistance. The Status Screen will then ask the Assistance Module to launch another application, such as Google Maps or Yelp, for greater assistance and will be transferred accordingly.

4.3 Data Flow

The system was broken down into different modules in order to create an extensible system and enabled independent development of each tasks. The interactions between modules is modeled in Figure 4.3. Data begins in the Google servers, either in Google Calendar or in Google Tasks. The Calendar Translator module receives updates from the servers and displays the data on the Android Companion App. The translator module was separated to enable future improvement through the swapping or adding of multiple translator modules drawing from different data sources. The app

displays the pulled data as well as allows the tagging of this data with extra information such as the type of activity of priority level. This data is then forwarded in the form of JSON objects to the Google Glass device. The device receives it and inserts it into the Scheduler Manager module. The Scheduler Manager is responsible for evaluating the data it is given and prioritizing it. The module receives requests from the user for data through the status screens and sends it in the form of Java Objects defined by the Scheduler Manager.

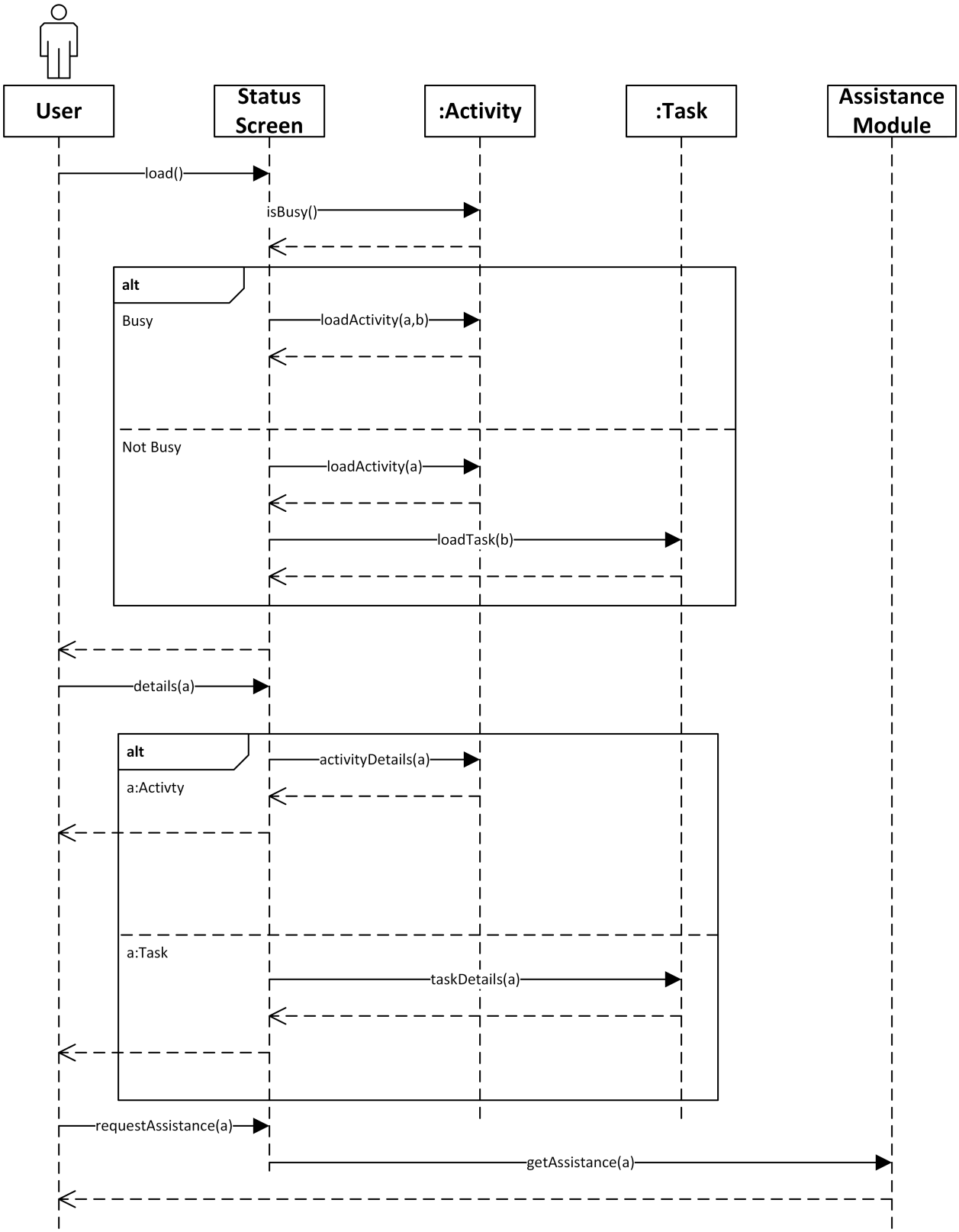

Figure 4.2: System Sequence Diagram

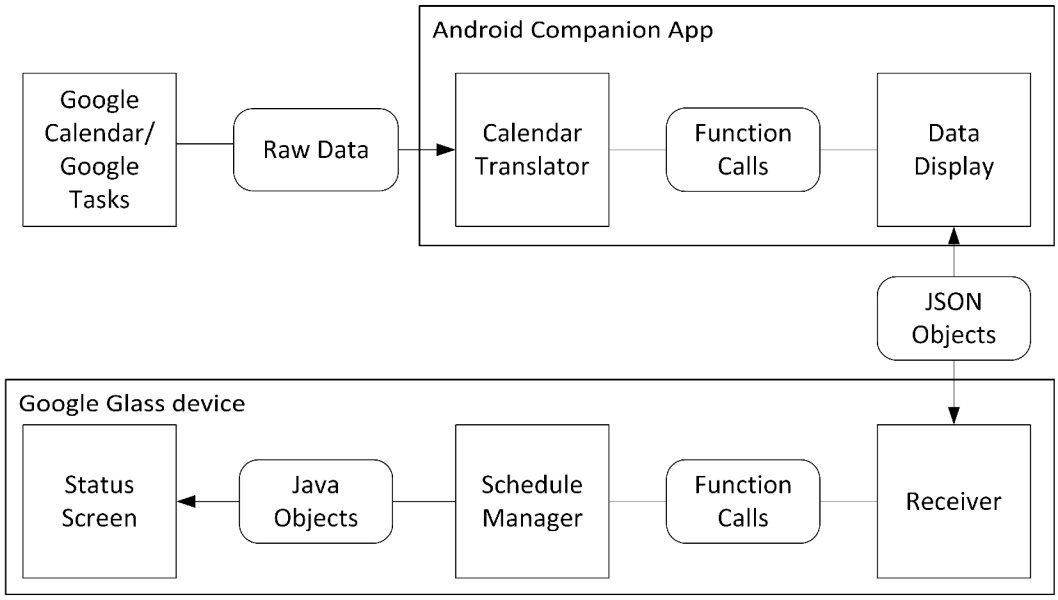

Figure 4.3: Data Flow Diagram

Chapter 5

Design Rationale

5.1 Technologies Used

The system uses a variety of technologies focusing on smartphone applications to fully meet all requirements. The system is implemented using Java , Android SDK, and the Glass Development Kit.

- Java is a programming language that serves as the foundation for the Android Operating System. Java is class-based, object-oriented, and designed to be portable so that it can be coded independently of the platform.
- The Android SDK, known as Android Software Development Kit, is a set of software development tools that allows the creation of applications for Android based systems. Android SDK is built upon and works in conjunction with the Java programming language.
- The Glass Development Kit is an add-on to the Android SDK that allows software called Glassware to be built and run directly on a Google Glass device.

5.2 Design Choices

Our system will be implemented to run primarily on the Google Glass device. This allows for portability and accessibility within the dynamic life management software no matter where the user is or what the user may be doing. The system will work in conjunction with an Android application. Mobile applications have numerous advantages and benefits over regular desktop applications, one of the most important being the seamless experience provided to the users. Mobile applications allow users to perform activities and tasks once limited by a stationary desktop. The use of an application in conjunction with a wearable device exemplifies that the system is beneficial to the users in all

aspects of their lives. No longer will users have to remember to use the system when they want to be productive because our system will remind them to.

The primary purpose of developing a mobile application is to provide an easy experience for users to add tasks/goals around their regular schedules quickly and efficiently. A mobile application is needed in order to work with the Google Glass because typing is not supported and can be extremely time consuming if a future update allows it. A mobile application seems the best choice to solve this issue since a Google Glass typically relies on a mobile phone for internet connectivity.

5.3 Suggestion Algorithm

The Schedule Manager is built around a formula defined heuristically based on the predicted needs of a user. This algorithm takes in a variety of factors and produces a relative importance, or weight, of a particular task. The higher the weight, the sooner the task is presented to the user. There are five pieces of information considered in this weight.

1. **Category** The category of a task is the type of activity to be performed. The system has three categories. Work is a category for academic or career related tasks. These are considered to have the highest importance. The category Exercise includes physical activity meant to maintain a user's health. An example would be time at the gym or yoga. This category is of medium importance and is also used to monitor user health. The last category is Rest. Rest is a relatively low level category for non-strenuous activities such as social gatherings or time to relax. If a task is not characterized by a user, it is automatically placed in the work category.

2. **Due Date** One of the strongest indications of importance is an upcoming due date. The system considers due dates in bucket categories which places items with similar due dates in the same category. It then applies a transformation based on the buckets to come up with a numeric representation of the relative importance of each task.

3. **Category Completion** The system keeps track of the relative completion rates of tasks in each category. A user who is consistently failing to perform their tasks in a specific category needs to be told about future tasks in the category with more ample time to complete them. For example, if a user continuously fails to complete Work category items, they will begin to see tasks in this category a few days sooner than they normally would.This gives them more time to complete the tasks.

4. **Skips** Tasks that are skipped are assumed to be of relatively less importance than otherwise equal tasks. The system keeps track of the number of skips and reduces the importance of these tasks up to a cap.

5. **Priority** Priority is a user-defined category. Tasks are given a numerical importance by the user using the Android Companion app. The ratings span from 1, relatively unimportant, to 5, extremely important. The default importance is 3. Since the priority is directly defined by the user in this system it carries a high level of importance.

To see the derivation of the formula used in the application, please see Appendix A.

Chapter 6

Testing

The testing phase is a key portion in any type of software development because its primary purpose is to find bugs and defects in software. Our test plan consisted of numerous stages and phases which were used all throughout the development stages.

6.1 Interface Testing

The interface-testing phase verifies the applications ease of navigation. Since the Graphic User Interface (GUI) is separated from the backend of the system, we were able to test it independent of the rest of the system. When testing the interface, we navigated the use cases just as a user would. This allowed us to test the flow of our menus as well as how natural the gesture commands felt. Additionally, we were testing for the usability of the layout and the general look of DyLMA.

6.2 Verification Testing

The first stage of testing for the back end was the verification stage. The verification stage involves proving the system is functionally correct. This had four main cases:

- Verify that the user's calendar is being accurately synchronized with the Google Glass application
- Verify that the user is able to navigate through the different tasks and events they have scheduled. These cases tested the level of responsiveness of our system.
- Verify that the system can update with Google calendar, to test the portability.
- Verify the application provides various suggestions based on the user's list of events, checking the usability aspect.

6.3 Integration Testing

The next stage of testing was integration testing. This tested the entire application from end to end for errors and bugs that are exposed when the different components are combined. In this stage, other aspects come into play such as performance and latency. The application needs to be functioning correctly and have low latency as latency negatively impacts a user's experience.

6.4 Validation Testing

Once every component of our software has been compiled, we performed validation testing. Validation testing is a check to see if the right product has been built for the marketplace. A technically proficient product which does not suit the user's needs will not succeed. There were two stages, Alpha and Beta testing. Alpha testing is when a software or product is tested among the internal team of developers or engineers within a corporation. Beta testing is when a community of people outside the corporation is invited to use the product to find potential bugs within the software.

Alpha We have worn the Google Glass running DyLMA and have used it to schedule our days. We have found it to be helpful, but limited by the system's battery life.

Beta Once our product successfully passes Alpha testing, we will invite a handful of users from Santa Clara to help us test our system.

6.5 Test Results

Throughout the rigorous testing of our system and application, we worked to make each of the project's functional and nonfunctional requirements operate smoothly. Users of our application are able to successfully use our system to generate a custom schedule of tasks and events based on priority and maximum efficiency.

The final project passed each of the test cases in the verification. This phase brought out numerous bugs and inconsistencies for adding or removing tasks. We had minor bugs in the system when shifting the tasks based on priority but were able to correct each error. The alpha testing and beta testing phase were beneficial due to the variety of actions produced by real users. We found most of our errors and bugs during or before alpha testing; we found our misunderstandings of how the Glass devices is used by users in alpha and beta testing. Listing each one of these errors out, we were able to debug them individually to result in a complete scheduling system.

Unfortunately, alpha and beta testing revealed the shortcoming of the project's host device. The Google Glass does not lend itself well to extended usage, which our application requires. Due to the short battery life and overheating problems, the application was not as effective as it could have been.

Chapter 7

Social Concerns

7.1 Ethics Analysis

7.1.1 Social and Ethical Ramifications

Ethical Data Collection An ethical responsibility we made sure to adhere to is preserving and securing the data we obtained from the user. From a technical standpoint we must obtain and interpret user data in order for our system to offer and accomplish the services we intend. Having been entrusted with this necessary data, we have taken measures to prevent data leaks.

Preventing Sale of User Data A common practice among companies that work with large amounts of personal metadata is to attempt to commoditize this information, often through the introduction of targeted advertising or the selling of the information to third parties for their use. Google Glass allows for the sale, rent, or providing data to a third party, as long as the third party is not using the information for advertising purposes. Our legal obligation is to request an opt-in for this specific data use. Providing this level of detailed, personal information to a third party after it has been entrusted to us would be unethical. For this reason, only the information that is necessary for our system to function is shared.

Accuracy of Suggestions As our system advises individuals on the best use of their time, there is an ethical obligation to provide users with the best possible information. However, the application is a suggestion based system and relies only on the information provided by the user. It is the user's responsibility to accomplish tasks, whether or not the application has advised them on that subject. It is also the user's responsibility to provide quality data for processing in order to receive quality information. The ethical obligation extends as far as offering suggestions to the user. It still remains their duty to follow through on their responsibilities.

7.1.2 Product Development Ethics

Preventing Data Theft Integrity is ensured within the product through the use of data encryption and access controls. Data encryption will be used to lock the user's information with a cipher making it difficult to decode in the case of a data breach. All of this encrypted data will be useless without the ability to decrypt the data.

Authentication The implementation of access controls is necessary in order to limit the read and write privileges for the various modules and components in the application. User access to data can be accomplished through the authentication process of the data sources. In order for data to be trusted by the users, they need to feel assured that the application will not undermine the user's data security.

7.1.3 Organizational Ethics

Engineering Obligations It is a moral obligation for a group developing a project to operate within the ethical standards of engineering. Our project must not violate the principles of do no harm, nor violate other established engineering moral codes of conducts. An individual has been appointed to monitor that these ethical principles or obligations are not violated.

Fairness and Equality Team equality has been preserved through a debate and voting process. This ensured that all creativity and novel ideas were taken into consideration during the development process.

7.2 Aesthetics Analysis

Design Aesthetics Our design is significantly influenced by aesthetic and usability considerations due to our platform. Using the Google Glass as our platform, one of the design constraints that we face is making the best utilization of the limited display space. Due to the small nature of the screen, all of the information presented to the user must be easy to read and understand. Often, only the most crucial information can be presented in one moment and the user must be able to infer the rest. Additionally, usability becomes a key issue when we consider the limited user interaction of the device. Most of the input into the device relies upon the swiping mechanism on the side, which only has 5 simple commands possible. The design of our application must make it intuitive to navigate through all of our functionality using only these five buttons, while maintaining consistency with other apps that the system provides.

Inherently Simple Our application uses the fewest possible methods of user input and control in order to maintain usability. Ultimately, our goal was to remove any and all confusion revolving around using the system and make it easy to use within the daily lives of our users. Displaying notifications to the user is done in easy to read fonts and uses a hierarchical layout where each level becomes more and more detailed than the prior level. The purpose of this tiered data display is to only display a relevant amount of information at any time. The system only provides more depth in the case that the user requests more details. The user also has the option to put aside notifications at an early level which prevents the screen from becoming cluttered with unwanted information.

Aesthetics Matter Many of our design decisions are based on our expected users and the environment in which we expect them to operate. Currently, Google Glass users are a mostly young group of individuals. They tend to be highly technology oriented, as it is still in its pioneering phase. This tells us that our users want an elegant design. Technology junkies have a tendency to move on to the next comparable product whenever they are displeased with the visual design. This project required that we maximized the visual appeal of the screens within the limitations that the tiny screen provided. On the other hand, we tried to keep the design minimalist as the device's location on the face makes cluttered apps bothersome.

Usable We have taken design aesthetics into great consideration as we were developing and prototyping our intelligent events application. Having a product or application rich in function

means nothing unless it is simple to use, extremely user friendly, and easy to learn. Such characteristics either draw users towards the product or away, which translates to success or failure. Usability has heavily influenced the design of our project. The limited nature of our user interaction and limited screen space has made this project largely function following form.

Chapter 8

Risk Assesment

In order to evaluate possible risks and their consequences, we created a risk assessment table. This table provides the probability and severity of a risk, the resulting impact, and our mitigation strategies.

Table 8.1: Assessment of Risks and their Mitigation Strategies

Risk	**Consequence**	**Probability (P)**	**Severity (S)**	**Impact (P*S)**	**Mitigation**
Group member getting sick	Personal deadlines get pushed back or parts of the system not finished on time	1	8	8	Have 2 members assigned per task and set early soft deadlines
Bugs	System not working as planned	1	5	5	Have a test plan ready and begin testing in early stages
Time	Parts of the system not being finished	.5	8	4	Prioritize features and set early soft deadlines
Incorrect knowledge of technology used	Time spent learning topics again	.3	7	2.1	Assign project tasks based on member with most experience on related topic/subject
Data Loss	Losing work	.1	8	.8	Keep multiple backups of project and update each other on progress
Group Conflict/ Miscommunication	Lack of unity or togetherness	.15	3	.45	Have frequent meetings and assign lead members in different areas

Chapter 9

Development Timeline

Figures 9.1 to 9.3, starting on page 28, show a Gantt Chart. These figures portray the major components of the project's development cycle. A Gantt chart displays the amount of work done or production that is completed in certain periods of time in relation to the amount planned for in such periods. The chart provides an overview of all the stages and processes the system goes through during development. The life span of this project is three quarters, but some included components have hard deadlines prior to the final deadline. During each quarter of the project's life cycle, time is allocated for individual components of the system. The component assignments are color coded based on the included legend and indicate the personnel responsible for the appropriate project component.

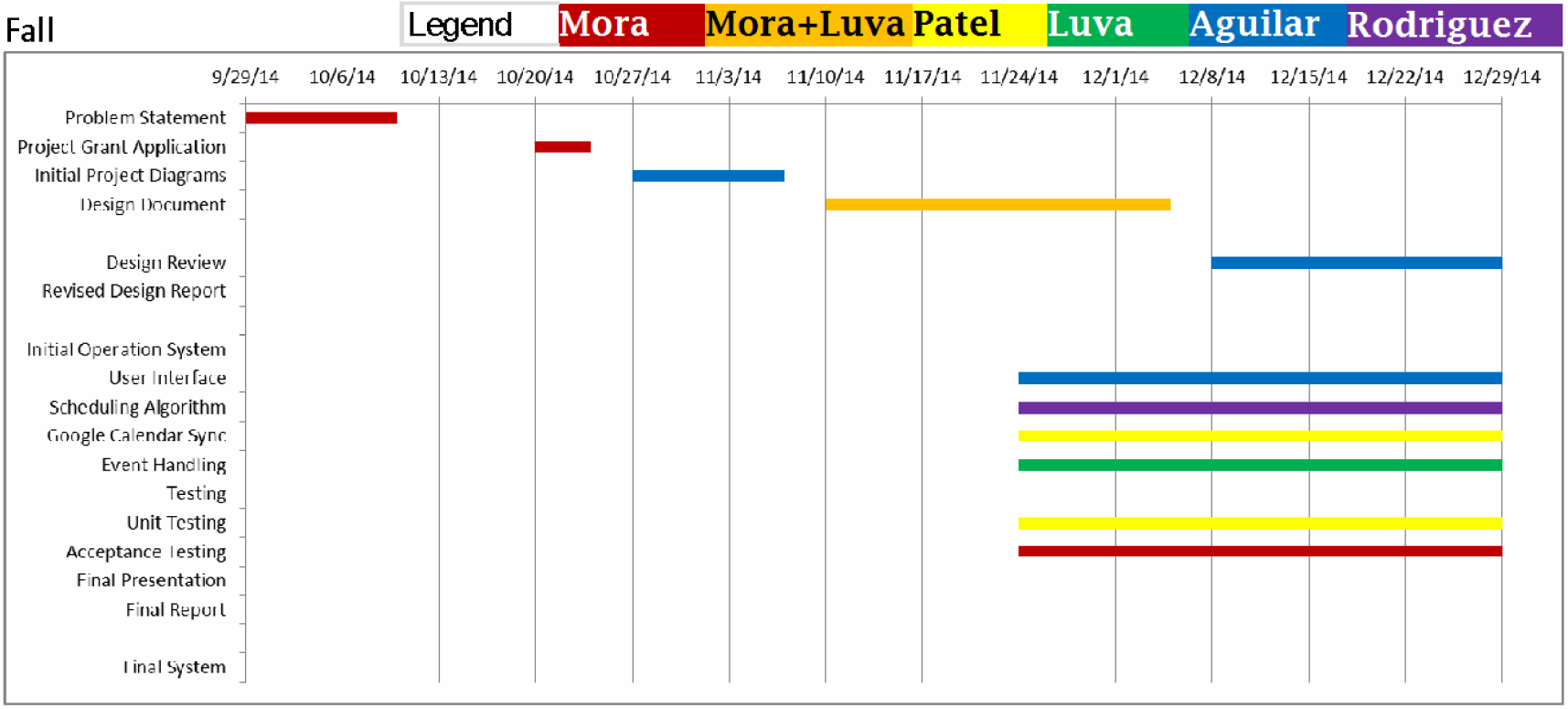

Figure 9.1: Gantt Chart: Fall

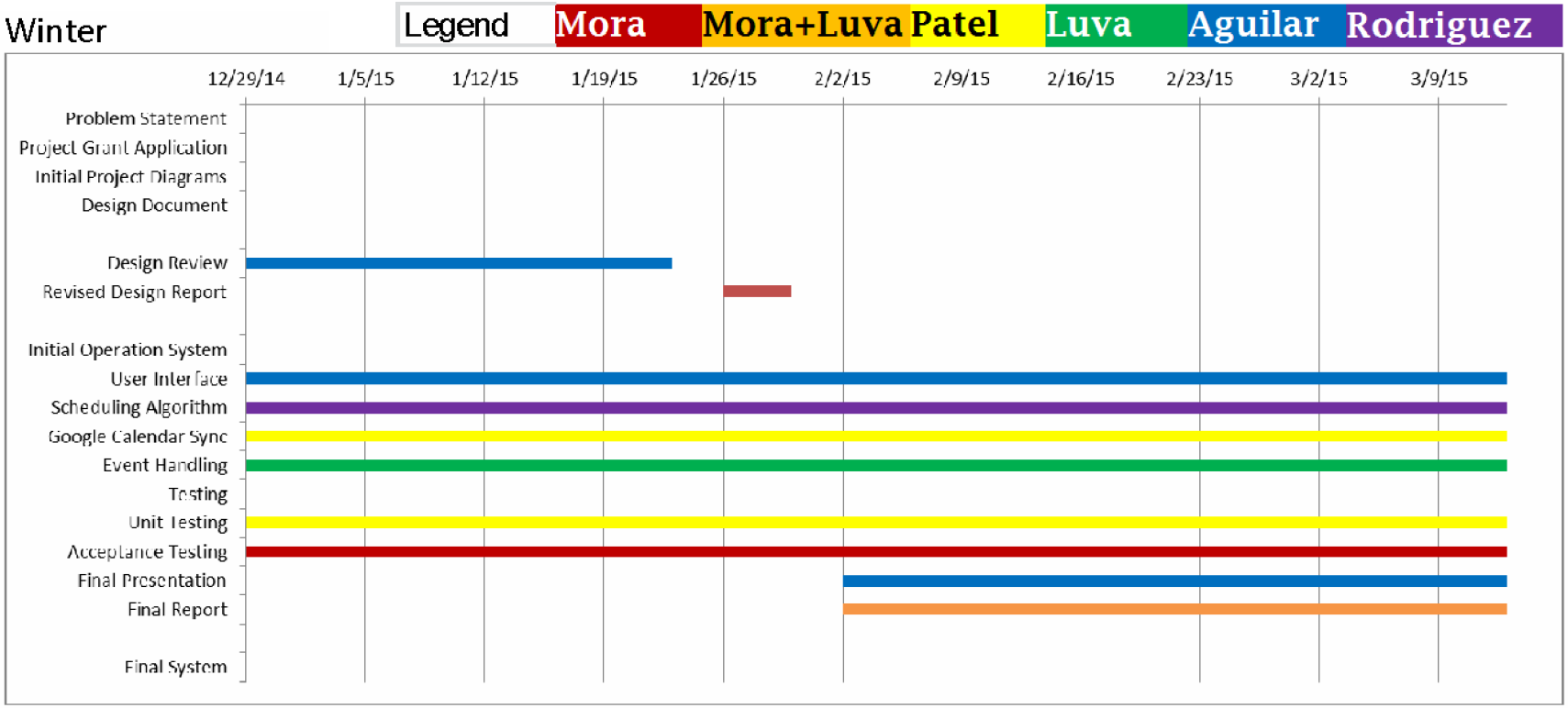

Figure 9.2: Gantt Chart: Winter

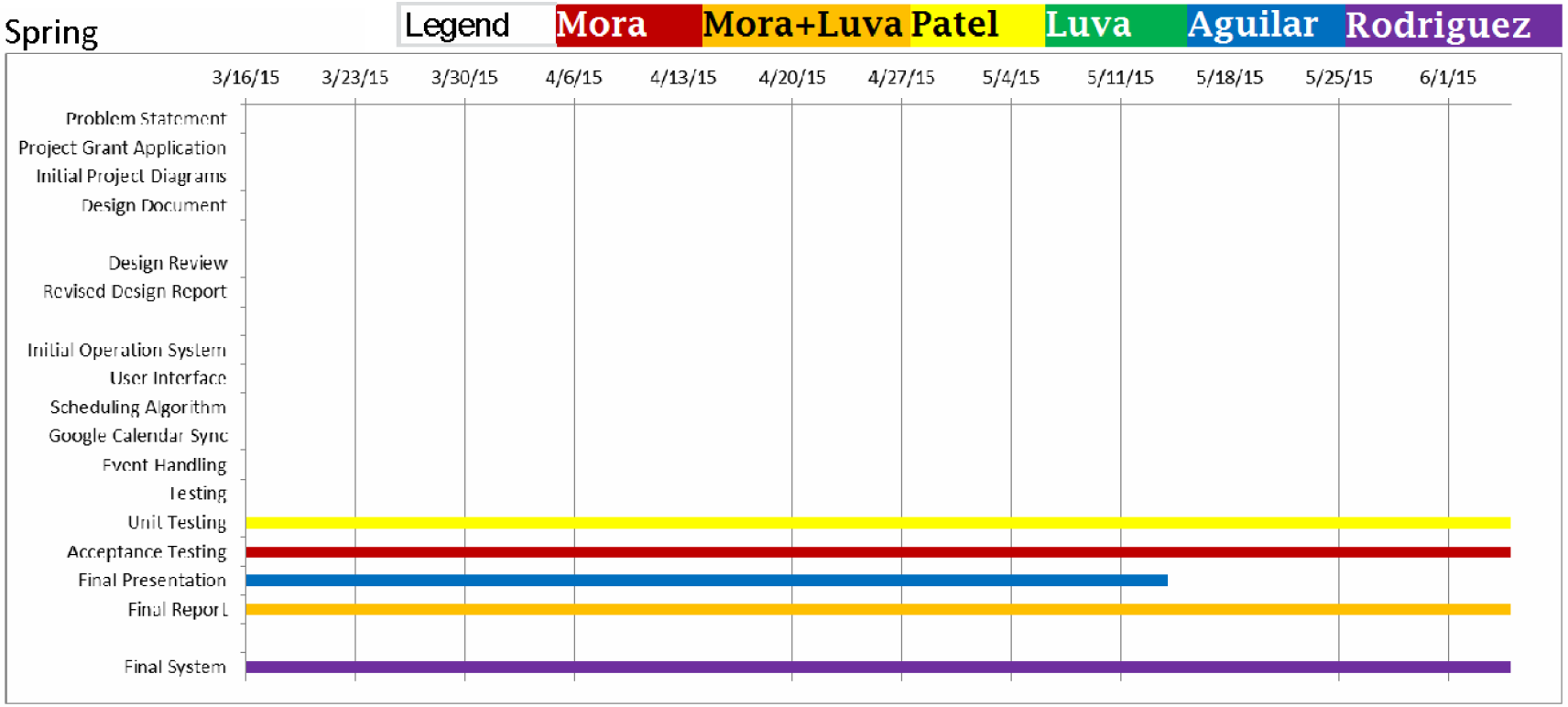

Figure 9.3: Gantt Chart: Spring

Chapter 10

Future of the Project

Due to the uncertainty of the Google Glass project, the future of the project will likely continue on a different device or suitable platform. Because of the modular nature of DyLMA, certain aspects of the project can be expanded upon or redesigned for different platforms relatively easy.

Host Platform The Google Glass no longer seems like a suitable platform for this application. The most natural change to the project would be to migrate the application to another mobile device, such as a wearable. This includes devices/watches running on Android Wear. These devices have a constant visual presence, similar to the Glass, and have the ability to sync with Google services with ease. Smart watches are also available to a larger audience because of their much lower price point than Google Glass. Another potential platform includes the Microsoft HoloLens as it offers a more sophisticated solution than the Google Glass. The HoloLens offers an actual augmented experience, other than simply having another screen in front of your eyes. A move to another device would involve redesigning the Graphical User Interface and modifications to the translator modules but would largely leave the scheduling module intact.

Scheduling Algorithm The scheduling algorithm is not as sophisticated as was originally intended. The priority adjustments need fine-tuning that could be determined through heuristic testing on a wider array of end users. Additionally, the algorithm could be expanded to take into consideration more of the user's information. This was mainly due to the limitations on large scale user testing, which made it difficult to enhance the algorithm for all use cases. This algorithm should be a key target for expansion with any future work on this project.

Chapter 11

Conclusion

11.1 Project State

The end product meets the defined requirements, as well as includes all the initially planned features with some limitations. The project could be introduced to the Google Glassware store and made available. However, this seems like a futile action as there is only a small market of current Glass owners and no guarantee of compatibility with the next iteration of the platform. The application should be expanded further before it forms a marketable product. The heuristic decisions on the algorithm have not been tested enough to say that they fit the expectations of the users.

11.2 Evaluation of Solution

The final application presents users with a set of advantages and disadvantages over currently available solutions. While this product provides a useful combination of several services and a helpful algorithm, it is severely hindered by the device.

11.2.1 Advantages

- The application combines multiple services which were previously spread throughout multiple applications
- The application promotes the user's well-being
- The application's algorithm offers a way to optimize productivity

11.2.2 Disadvantages

- The application can not be run long enough on the device to be useful to the user for a complete day

- The platform for the application is expensive
- Inputting user information is difficult

11.3 Lessons Learned

There is great risk in developing products and services for upcoming and emerging platforms. First of all, there is no guarantee that the platform will be successful and remain in production. The platform that you may be developing on could be discontinued or left unsupported. Additionally, if the platform is yet to be in production, developer documentation may not exist or may be severely limited. This provides a constraint for developing on such a device. However, to be successful with an emerging device, beginning early is key to project completion even after dealing with the risks and constraints associated.

For larger projects, pairwise programming is beneficial for success. The presence of two programmers provides a method for code review during the writing process, thus making code more efficient.

While the product itself is fully functioning, it is disheartening to have a platform fall apart midway through the development process, leaving ambiguity and concern about the future worth of the project. Despite the shortcomings and surprises dealt while working with the Google Glass, the group generally believes that the initial intent for the product was met.

Appendix A

Derivation of the Suggestion Algorithm

The algorithm decides the order in which tasks are given to the user. The higher the weight produced by the algorithm, the more important a task is. Calculation of a weight is based upon category completion percentages, category of task, due date, number of skips, and user-given priority level.

1. The base value is established through the time remaining until the due date. Items with sooner due dates must be given higher final weights.In general, users will not differentiate due dates which are close to each other. This affect is amplified the further the due date is from the current date. This led to the establishment of buckets for due dates as follows:

   Due Date to Bucket Transformation

Days Remaining	0-1	1-3	3-6	6-10	10-15	15+
Bucket	0	1	2	3	4	5

   Those items in bucket zero have the most importance. These are assigned a value of 1. The rest of the buckets must decrease in importance. In determining the relative importance of the buckets, it was assumed that while bucket 0 is much more important than bucket 1, bucket 4 is only marginally more important than bucket 5. A function which exhibits this behavior is an exponential function with a base b where $0 < b < 1$. Additionally, in order to account for items that are past due, the buckets are mirrored onto the negative due dates to achieve:

   $$\omega_0 = .75^{|bucket|}$$

2. The three categories were each assigned relative values. These values are multiplied with the values calculated from the due date. These relative values are used to reduce the importance of certain tasks. Exercise was deemed half as important as Work. Rest was deemed less important than Exercise, but not half as important. For this formula, the Work category to be of full importance so it was assigned a value of 1. This makes Exercise become 0.5. Since rest should be greater than 0.25 but still fairly low, the team settled on 0.35. These numbers are multiplied around the base importance.

   $$\omega_1 = \tau * \omega_0$$

   $$\text{where } \tau = \begin{cases} 1.0 & \text{if } Category \text{ is Work} \\ 0.5 & \text{if } Category \text{ is Exercise} \\ 0.35 & \text{if } Category \text{ is Rest} \end{cases}$$

3. Tasks in categories that are not being completed are given higher priority over tasks that are being completed. This ensures that the task will appear with enough time for a user to complete the task. The available statistic on each category is the percentage of successful completions in each category. A task with 100% success rate in its category should experience no change, while a task with a low success rate should become more important. A transformation which follows these guidelines is:

$$\omega_2 = \omega_1 * (2 - \text{Category Success Rate})$$

4. Items that are being skipped are reduced in priority. The first skip should give the largest reduction in priority, with each further skip becoming less effective. After several skips, the user has essentially declared the task to be very low priority. Using this idea, the effect of skips on the weight is capped after 3 skips. The same transformation of the data is used as for due dates, as a skip is roughly equal to a downgrade by one bucket:

$$\omega_3 = \omega_2 * .75^{skips}$$

5. Priority is a user-defined category. As the only user-defined category, it was given a very high weight in assigning weights. A priority 5 task must have a very high weight, while a priority 1 task should have a relatively small weight. In attempting to determine some constants heuristically with test data, we arrived at the following constants:

Priority to α Transformation

Priority	1	2	3	4	5
α	.35	.75	1	1.75	3

Leading to the equation:

$$Weight = \alpha * \omega_3$$

Which when fully expanded is:

$$Weight = \alpha * \tau * .75^{|bucket|} * (2 - \text{Category Success Rate}) * .75^{skips}$$

Appendix B

Source Code

B.1 Schedule Manager

The scheduler manager works by creating an instance of a Scheduler class and by passing information into this instance. The instance may then be queried to receive the needed information on the user's schedule. The Scheduler class works by sorting Tasks based on their calculated weights. Events are kept in a chronological order as they are fixed in time. Each instance of the Scheduler makes use of an instance of the SuggestionSystem class in order to insert suggested items as well as track information on the data.

B.1.1 Base Classes

The base classes for this system are the Activity, Event, and Task classes. They contain the information for an item in a user's schedule. Event and Task are extensions of an Activity, which is an abstract class. Instances of Task and Event are passed internally in the Scheduler. Activity defines the categories of Events and Tasks. The name of the class had to be changed to ActivityS due to a conflict with a Google Glass predefined class.

Listing B.1: Activity Class

```
1  package com.ld.project.activity;

   import java.util.Calendar;
   import com.ld.project.suggestionSystem.ActivityCategory;

6  // Parent class of Tasks and (Calendar) Events
   public abstract class ActivityS implements java.io.Serializable{
       private static final long serialVersionUID = 1L;

       private String id;
11     private String title;
       private Calendar updated;
       private ActivityCategory type;


16     // constructor default type
       public ActivityS(String _id, String _title){
           this(_id, _title, Calendar.getInstance());
       }
       public ActivityS(String _id, String _title, String _type){
21         this(_id, _title);
           setType(_type);
       }

       // constructor with updated
26     public ActivityS(String _id, String _title,
                        Calendar _updated){
           this(_id, _title,
                   _updated, "work");
       }
31
       // constructor all variables
       public ActivityS(String _id, String _title,
                        Calendar _updated, String _type){
           this.setId(_id);
36         this.setTitle(_title);
           this.setUpdated(_updated);
           this.setType(_type);
       }

41
       // two Activities are the same activity (with possibly updated information)
       // if they have the same id
       @Override
       public boolean equals(Object obj) {
46         if (!(obj instanceof ActivityS))
```

```
            return false;
        if (obj == this)
            return true;

        ActivityS rhs = (ActivityS) obj;
        if(getId().equals(rhs.getId())){
            return true;
        }
        return false;
    }


    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getTitle() {
        if(title == null){
            return "error";
        }

        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public Calendar getUpdated() {
        return updated;
    }
    public void setUpdated(Calendar updated) {
        this.updated = updated;
    }

    public String getType() {

        if(type == ActivityCategory.WORK){
            return "work";
        }
        if(type == ActivityCategory.REST){
            return "rest";
        }
        if(type == ActivityCategory.EXERCISE){
            return "exercise";
        }
        return "error";
    }

    // work, rest, or exercise
    public void setType(String _type) {
        String _typeLower = _type.toLowerCase();
        if(_typeLower.equals("work")){
            type = ActivityCategory.WORK;
            return;
        }
        if(_typeLower.equals("rest")){
            type = ActivityCategory.REST;
            return;
        }
        if(_typeLower.equals("exercise")){
            type = ActivityCategory.EXERCISE;
            return;
        }
    }

}
```

The Task class is used for non-time specific events. Each of these represents a task that needs to be done by a certain date. They are organized inside of the Scheduler class. The most important function in this class is the getWeight() function. Each instance of a Task is capable of determining how "important" it is with a number. The higher the number, the more important. The Task class implements the Comparable interface in order to make it easier to sort in the Scheduler.

Listing B.2: Task Class

```
package activity;

import java.text.SimpleDateFormat;
import java.util.Calendar;

import suggestionSystem.ActivityCategory;
import suggestionSystem.SuggestionSystem;

public class Task extends ActivityS implements Comparable<Task> {

  private Calendar due;
  private String description;
  private int priority;
  private int skips;

  @Override
  // Overriding the compareTo method using weight
  public int compareTo(Task other) {
    Double d = new Double(this.getWeight());
    Double oth = new Double(other.getWeight());
    return oth.compareTo(d);
  }

  // returns near one if important, small if not
  public double getWeight() {

    // ////////////////////////////////
    // Calculate time left and assign value
    // ////////////////////////////////

    // effective current time rounded to nearest hour
    Calendar currTime = Calendar.getInstance();
    int unroundedMinutes = currTime.get(Calendar.MINUTE);
    int mod = unroundedMinutes % 15;
    currTime.add(Calendar.MINUTE, -mod);
    currTime.set(Calendar.SECOND, 0);
    currTime.set(Calendar.MILLISECOND, 0);

    double value = due.getTimeInMillis() - currTime.getTimeInMillis();
    value = Math.abs(value);
    // value into seconds
    value /= 1000;
    // value into minutes
    value = (double) Math.floor(value /= 60);
    // value into hours
    value = (double) Math.floor(value /= 60);

    // six cases of categories:
    // dueDateValue will be .75^|x| where x is a time duration category
    double dueDateValue;
    // 0 - 1 days
    if (value <= 24) {
      dueDateValue = 0;
    }
    // 1 - 3 days
    else if (24 < value && value <= 72) {
      dueDateValue = 1;
    }
    // 3 - 6 days
    else if (72 < value && value <= 144) {
```

```
61        dueDateValue = 2;
      }
      // 6 - 10 days
      else if (144 < value && value <= 240) {
        dueDateValue = 3;
66    }
      // 10 - 15 days
      else if (240 < value && value <= 360) {
        dueDateValue = 4;
      }
71    // 15+ days
      else {
        dueDateValue = 5;
      }
      dueDateValue = Math.pow(.75, dueDateValue);
76
      // ////////////////////////////////////
      // Calculate category constant
      // ////////////////////////////////////

81    double categoryConst;
      if (this.getType() == ActivityCategory.WORK) {
        categoryConst = 1;
      } else if (this.getType() == ActivityCategory.EXERCISE) {
        categoryConst = .5;
86    }
      // ActivityCategory.Rest
      else {
        categoryConst = .35;
      }
91
      // ////////////////////////////////////
      // Calculate completion rate for category
      // ////////////////////////////////////
      double completionRate; // percentage of completed work
96    if (this.getType() == ActivityCategory.WORK) {
        completionRate = SuggestionSystem.getWorkCompleted();
      } else if (this.getType() == ActivityCategory.EXERCISE) {
        completionRate = SuggestionSystem.getExerciseCompleted();
      }
101   // ActivityCategory.Rest
      else {
        completionRate = SuggestionSystem.getRestCompleted();
      }
      // returns value between 2 and 1 for multiplication
106   completionRate = (2 - completionRate);

      // ////////////////////////////////////
      // Calculate skip penalty
      // ////////////////////////////////////
111
      // value between 0 - 1, 1 being no skips
      // cap skips affect at 5 skips;
      double skipValue = Math.min(5, skips);
      skipValue = Math.pow(.75, skipValue);
116
      // ////////////////////////////////////
      // Priority Scale
      // ////////////////////////////////////

121   double priorityScale = this.getPriority();
      // ultra-high priority
      if (priorityScale == 5) {
        priorityScale = 3;
      }
126   // high priority
      else if (priorityScale == 4) {
        priorityScale = 1.75;
```

```
    }
    // average priority
    else if (priorityScale == 3) {
      priorityScale = 1;
    }
    // low priority
    else if (priorityScale == 2) {
      priorityScale = .75;
    }
    // ultra-low priority
    if (priorityScale == 1) {
      priorityScale = .35;
    }

    // ////////////////////////
    // ////////////////////////
    // Final Equation
    // ////////////////////////
    // ////////////////////

    // final priority value
    return priorityScale * dueDateValue * categoryConst * completionRate
        * skipValue;
  }

  // sets updated:
  // with type:
  public Task(String _id, String _title, Calendar _updated,
      String _description, Calendar _due, int _priority, String _type) {
    super(_id, _title, _updated, _type);
    this.setDue(_due);
    this.setDescription(_description);
    this.setPriority(_priority);
    this.resetSkips();
    this.setType(_type);
  }

  // without type:
  public Task(String _id, String _title, Calendar _updated,
      String _description, Calendar _due, int _priority) {
    this(_id, _title, _updated, _description, _due, _priority, "work");
  }

  // does not set updated:
  // with type:
  public Task(String _id, String _title,String _description, Calendar _due,
      int _priority, String _type) {
    super(_id, _title, _type);
    this.setDue(_due);
    this.setDescription(_description);
    this.setPriority(_priority);
    this.resetSkips();
    this.setType(_type);
  }

  // without type:
  public Task(String _id, String _title,  String _description,Calendar _due,
      int _priority) {
    this(_id, _title,_description,  _due, _priority, "work");
  }

  // without type, without updated, default priority
  public Task(String _id, String _title,  String _description,Calendar _due) {
    this(_id, _title,_description,  _due, 3, "work");
  }

  public Calendar getDue() {
    return due;
```

```
  }

  // rounds down to nearest minute
  public void setDue(Calendar due) {
    due.set(Calendar.SECOND, 0);
    due.set(Calendar.MILLISECOND, 0);
    this.due = due;
  }

  public int getPriority() {
    return priority;
  }

  public void setPriority(int _priority) {
    if (_priority == 1) {
      priority = 1;
      return;
    }
    if (_priority == 2) {
      priority = 2;
      return;
    }
    if (_priority == 3) {
      priority = 3;
      return;
    }
    if (_priority == 4) {
      priority = 4;
      return;
    }
    if (_priority == 5) {
      priority = 5;
      return;
    }
  }

  public String getDescription() {
    return description;
  }

  public void setDescription(String description) {
    this.description = description;
  }

  public void resetSkips() {
    skips = 0;
  }

  public void skip() {
    skips += 1;
  }

  @Override
  public String toString() {

    SimpleDateFormat format1 = new SimpleDateFormat("MM-dd, HH:mm");
    String formatted = format1.format(due.getTime());

    return "Task [title=" + this.getTitle() + " | priority=" + priority
        + " | due=" + formatted + " | skips=" + skips + " | weight="
        + getWeight() + "]";

  }

}
```

The Event class is a relatively simple class. This is due to the fact that an event in a calendar is fixed in time. The Event class simply adds a start time, end time, and a description to the Activity

class.

Listing B.3: Event Class

```
package activity;

import java.util.Calendar;

public class Event extends ActivityS {

  private String description;
  private Calendar start;
  private Calendar end;

  @Override
  public String toString() {
    return "Event [title=" + this.getTitle() + "]";
  }

  // with updated:
  // sets type
  public Event(String _id, String _title, Calendar _updated,
      String _description, Calendar _start, Calendar _end, String _type) {
    super(_id, _title, _updated, _type);
    this.setDescription(_description);
    this.setStart(_start);
    this.setEnd(_end);
  }

  // doesnt set type
  public Event(String _id, String _title, Calendar _updated,
      String _description, Calendar _start, Calendar _end) {
    super(_id, _title, _updated);
    this.setDescription(_description);
    this.setStart(_start);
    this.setEnd(_end);
  }

  // without updated:
  // with type
  public Event(String _id, String _title, String _description,
      Calendar _start, Calendar _end, String _type) {
    super(_id, _title, _type);
    this.setDescription(_description);
    this.setStart(_start);
    this.setEnd(_end);
  }

  // doesnt set type
  public Event(String _id, String _title, String _description,
      Calendar _start, Calendar _end) {
    super(_id, _title);
    this.setDescription(_description);
    this.setStart(_start);
    this.setEnd(_end);
  }

  public String getDescription() {
    return description;
  }

  public void setDescription(String description) {
    this.description = description;
  }

  public Calendar getStart() {
    return start;
  }
```

```
  // rounds down to nearest 15 minutes
  public void setStart(Calendar start) {
    int unroundedMinutes = start.get(Calendar.MINUTE);
    int mod = unroundedMinutes % 15;
    start.add(Calendar.MINUTE, -mod);
    start.set(Calendar.SECOND, 0);
    start.set(Calendar.MILLISECOND, 0);
    this.start = start;
  }

  public Calendar getEnd() {
    return end;
  }

  // rounds up to nearest 15 minutes
  public void setEnd(Calendar end) {
    int unroundedMinutes = start.get(Calendar.MINUTE);
    int mod = unroundedMinutes % 15;
    end.add(Calendar.MINUTE, 15 - mod);
    end.set(Calendar.SECOND, 0);
    end.set(Calendar.MILLISECOND, 0);
    this.end = end;
  }
}
```

B.1.2 Scheduler Class

The Scheduler class is the interface through which the features of the Suggestion System are accessed. First, an instance of a Scheduler is created. Insertion of a Task or Event is done through the insertActivity() functions. In order to receive a recommendation, requests are made through getEvent() and getTaskList(). In this iteration, getTaskList() only returns one task at a time. Once a user is in possession of a Task, t, they may interact with it by calling complete(t), skip(t), or remove(t) on the instance of the Scheduler. The Scheduler contains an instance of the SuggestionSystem, to which it reports changes in the data.

Listing B.4: Scheduler class

```
package scheduler;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collections;

import suggestionSystem.ActivityCategory;
import suggestionSystem.SuggestionSystem;
import activity.Event;
import activity.Task;

public class Scheduler {

  // index of the last event/task found
  private static int lastEvent = -1;
  private static int lastTask = -1;

  private ArrayList<Task> tasks = new ArrayList<Task>();
  private ArrayList<Event> events = new ArrayList<Event>();

  private ArrayList<Task> skipTasks = new ArrayList<Task>();
  private ArrayList<Task> currTasks = new ArrayList<Task>();

  private SuggestionSystem sugSys = new SuggestionSystem(this);

  // Initially an empty schedule, the system should make the instance of the
    scheduler persistent
    // outside of this class.
```

```
  Scheduler() {
  }

  public Task complete(Task t) {
    // report to suggestion system:
    sugSys.completed(t);

    // first check if removal area is before or after where we currently
    // are:
    int location = tasks.indexOf(t);
    if (location <= lastTask && lastTask != -1 && location != -1) {
      lastTask--;
    }

    // cannot remove
    if (location == -1) {
      System.out.println("Error, could not remove: " + t);
    }
    // can remove
    else {
      tasks.remove(location);
      currTasks.remove(t);
    }

    // return a new task
    return getTask();
  }

  public Task skip(Task t) {

    // first check if removal area is before or after where we currently
    // are:
    int location = tasks.indexOf(t);
    if (location <= lastTask && lastTask != -1) {
      lastTask--;
    }

    // add to skip list
    skipTasks.add(t);
    // add a skip to it:
    t.skip();

    // cannot remove
    if (location == -1) {
      System.out.println("Error, could not remove: " + t);
    }
    // can remove to skipped list:
    else {
      tasks.remove(location);
      currTasks.remove(t);
    }

    // return a new task
    return getTask();

  }

  public Task remove(Task t) {
    // report to suggestion system:
    sugSys.removed(t);

    // first check if removal area is before or after where we currently
    // are:
    int location = tasks.indexOf(t);
    if (location <= lastTask && lastTask != -1) {
      lastTask--;
    }

```

```
    // cannot remove
    if (location == -1) {
      System.out.println("Error, could not remove: " + t);
    }
    // can remove
    else {
      tasks.remove(location);
      currTasks.remove(t);
    }

    // return a new task
    return getTask();

  }

  public void insertActivity(Task t) {
    // add to end, it gets sorted when you getTask();
    tasks.add(t);

  }

  // call on first use
  public Task[] getTaskList() {
    sugSys.refreshExercise();
    Task[] a = new Task[5];
    for (int i = 0; i < 5; i++) {
      a[i] = getTask();
    }
    return a;
  }

  protected Task getTask() {
    int i;

    sugSys.refreshExercise();

    Collections.sort(tasks);

    // search for the next task
    i = lastTask + 1;
    // look for a task
    while (i < tasks.size()) {
      Task t = tasks.get(i);
      // if this one has already been submitted or it
      if (currTasks.contains(t)) {
        i = lastTask = i + 1;
      } else {
        // if this is it:
        lastTask = i;
        currTasks.add(t);
        return t;
      }
    }

    // did not find any more tasks:
    System.out.println("Reached end");
    lastTask = -1;

    // reintegrate the skipped ones:
    tasks.addAll(skipTasks);
    // clear the skiptasks
    skipTasks = new ArrayList<Task>();

    // if we everything we have has been given:
    if (tasks.size() == currTasks.size()) {
      return null;
    }
    // if we have anything left, get more:
```

```
    else {
      return getTask();
    }
  }

  // returns the next event suggested
  public Event getEvent() {
    int i;

    Calendar currTime = Calendar.getInstance();
    currTime.set(Calendar.SECOND, 0);
    currTime.set(Calendar.MILLISECOND, 0);
    // search for the next event
    i = lastEvent + 1;
    // look for an event
    while (i < events.size()) {
      Event e = events.get(i);
      // if current time is after the event end, remove that event, don't
      // increase the counter
      if (currTime.compareTo(e.getEnd()) > 0) {
        System.out.println("deleted event because its over");
        events.remove(i);
        continue;
      }

      // current event is now either ongoing or after current time, return
      // this one
      lastEvent = i;
      return e;
    }

    // did not find any more events:
    System.out.println("Reached end");
    lastEvent = -1;
    return null;
  }

  public void insertActivity(Event e) {
    int i = 0;

    // only insert non-ended activities
    Calendar currTime = Calendar.getInstance();
    currTime.set(Calendar.SECOND, 0);
    currTime.set(Calendar.MILLISECOND, 0);

    Calendar eventEnd = e.getEnd();

    // if current time is after event end, do not add
    if (currTime.compareTo(eventEnd) > 0) {
      return;
    }

    // report event to the suggestion system
    sugSys.addEvent(e);

    // if empty
    if (events.size() == 0) {
      events.add(e);
      return;
    }

    // search for correct place
    for (i = 0; i < events.size(); i++) {
      // if this date is before or equal to this event, add it here
      if (e.getStart().compareTo(events.get(i).getStart()) < 1) {
        events.add(i, e);
        return;
      }
```

```
    }

    // if at the end
    events.add(i, e);
    return;
  }

  public boolean exercisingToday() {
    // create calendar for end of the day
    Calendar endOfDay = Calendar.getInstance();
    int year = endOfDay.get(Calendar.YEAR);
    int month = endOfDay.get(Calendar.MONTH);
    int day = endOfDay.get(Calendar.DATE);
    endOfDay.set(year, month, day, 23, 59, 59);
    endOfDay.set(Calendar.MILLISECOND, 999);

    // check events:
    // if there are events
    if (events.size() != 0) {
      // events are in order by start time:
      // look for an event today with exercise category
      int i = 0;
      while (events.get(i).getStart().before(endOfDay)) {
        if (events.get(i).getType() == ActivityCategory.EXERCISE) {
          return true;
        }
        i++;
      }
    }

    // create calendar for start of the day
    Calendar beginOfDay = Calendar.getInstance();
    year = beginOfDay.get(Calendar.YEAR);
    month = beginOfDay.get(Calendar.MONTH);
    day = beginOfDay.get(Calendar.DATE);
    beginOfDay.set(year, month, day, 0, 0, 0);
    beginOfDay.set(Calendar.MILLISECOND, 0);

    // check tasks:

    if (tasks.size() != 0) {
      for (int i = 0; i < tasks.size(); i++) {
        Calendar taskDate = tasks.get(i).getDue();
        if (taskDate.before(endOfDay) && taskDate.after(beginOfDay)) {
          if (tasks.get(i).getType() == ActivityCategory.EXERCISE)
            return true;
        }
      }
    }

    // checked both, found none: return false
    return false;
  }

  public boolean isBusy() {
    Calendar currTime = Calendar.getInstance();
    currTime.set(Calendar.SECOND, 0);
    currTime.set(Calendar.MILLISECOND, 0);

    int i = 0;
    // look for an event that is current
    while (i < events.size()) {
      Event e = events.get(i);

      // if current time is after the event end, remove that event, don't
      // increase the counter
      if (currTime.compareTo(e.getEnd()) > 0) {
        System.out.println("deleted event because its over");
```

```
300           events.remove(i);
              continue;
            }

            // events are sorted by start time, so if start time > current time
            if (e.getStart().compareTo(currTime) > 0) {
              return false;
            }
            // start time <= current time, so must be in event, busy
            else {
              return true;
            }
          }

          // got to end of list, not in an event, not busy
          return false;

        }


320   }
```

B.1.3 Suggestion System Class

The Suggestion System works by impacting the data in the Scheduler class to which it belongs. It makes occasional insertions of Exercise events, as well as monitoring completion statistics in each category: work, exercise, and rest. The suggestion system is still fairly rudimentary, currently only making an insert if the user has not exercised at least once in the past day.

Listing B.5: SuggestionSystem class

```
package suggestionSystem;

import java.util.ArrayList;
4 import java.util.Calendar;
import scheduler.Scheduler;
import activity.Event;
import activity.Task;

9 // system math works on assumption that events take continuous attention
// ex: event: work conference that lasts 2 days would mean 48 hours of work, not 2 8
    hour shifts)
public class SuggestionSystem {
  private Scheduler scheduler;
  private static double work = 0;
14  private static double workCompleted = 0;
  private static double rest = 0;
  private static double restCompleted = 0;
  private static double exercise = 0;
  private static double exerciseCompleted = 0;
19  private Calendar lastExercise;
  private ArrayList<Task> addedInTasks = new ArrayList<Task>();
  private ArrayList<Event> addedInEvents = new ArrayList<Event>();

  public SuggestionSystem(Scheduler s) {
24    this.scheduler = s;
    // starting values to prevent heavy changes
    addWork(10);
    addWorkCompleted(10);
    addRest(10);
29    addRestCompleted(10);
    addExercise(10);
    addExerciseCompleted(10);

    // set lastExercise to day before
34    Calendar cal = Calendar.getInstance();
```

```
    cal.add(Calendar.DATE, -1);
    lastExercise = cal;
  }

  public void refreshExercise() {
    // check if they have completed a SUGGESTED exercise today:
    Calendar today = Calendar.getInstance();
    boolean sameDay = lastExercise.get(Calendar.YEAR) == today
        .get(Calendar.YEAR)
        && lastExercise.get(Calendar.DAY_OF_YEAR) == today
            .get(Calendar.DAY_OF_YEAR);
    // if we have exercised today, do nothing
    if (sameDay) {
      return;
    }
    // check if they have/have-scheduled exercise for today
    if (!scheduler.exercisingToday()) {
      // if there is no exercise scheduled today, add one in
      Calendar endOfDay = Calendar.getInstance();
      int year = endOfDay.get(Calendar.YEAR);
      int month = endOfDay.get(Calendar.MONTH);
      int day = endOfDay.get(Calendar.DATE);
      endOfDay.set(year, month, day, 23, 59, 59);
      endOfDay.set(Calendar.MILLISECOND, 999);
      Task t = new Task("Exercise", "Exercise Today", "You do not have exercise
    scheduled for today. Please arrange to hit the gym, go for a jog, "
          + "or to spend some time at the park", endOfDay, 4, "exercise");
      addedInTasks.add(t);
      scheduler.insertActivity(t);
    }

  }

  // makes the changes to the numbers
  public void addEvent(Event e) {
    if (e == null)
      return;
    // catalog type:
    ActivityCategory type = e.getType();
    // get length in hours
    double length = (e.getEnd().getTimeInMillis() - e.getStart()
        .getTimeInMillis()) / 1000 / 60 / 60;
    System.out.println("--------------------" + length);
    if (type == ActivityCategory.EXERCISE) {
      addExercise(length);
    } else if (type == ActivityCategory.REST) {
      addRest(length);
    } else if (type == ActivityCategory.WORK) {
      addWork(length);
    }

    // if this one was one that was added in: delete from our list
    if (addedInEvents.contains(e)) {
      addedInEvents.remove(e);
    }
  }

  public void completed(Task t) {
    // catalog type:
    ActivityCategory type = t.getType();
    if (type == ActivityCategory.EXERCISE) {
      addExercise(1);
      addExerciseCompleted(1);
      lastExercise = Calendar.getInstance();
    } else if (type == ActivityCategory.REST) {
      addRest(1);
      addRestCompleted(1);
    } else if (type == ActivityCategory.WORK) {
```

```
      addWork(1);
      addRestCompleted(1);
    }

    // if this one was one that was added in: delete from our list
    if (addedInTasks.contains(t)) {
      addedInTasks.remove(t);
    }
  }

  public void removed(Task t) {
    // catalog type:
    ActivityCategory type = t.getType();
    if (type == ActivityCategory.EXERCISE) {
      addExercise(1);
      addExerciseCompleted(0);
      lastExercise = Calendar.getInstance();
    } else if (type == ActivityCategory.REST) {
      addRest(1);
      addRestCompleted(0);
    } else if (type == ActivityCategory.WORK) {
      addWork(1);
      addRestCompleted(0);
    }

    // if this one was one that was added in: delete from our list
    if (addedInTasks.contains(t)) {
      addedInTasks.remove(t);
    }
  }


  public double getWork() {
    return work;
  }

  public void addWork(double work) {
    SuggestionSystem.work += work;
  }

  public static double getWorkCompleted() {
    return workCompleted / work;
  }

  public void addWorkCompleted(double workCompleted) {
    SuggestionSystem.workCompleted += workCompleted;
  }

  public double getRest() {
    return rest;
  }

  public void addRest(double rest) {
    SuggestionSystem.rest += rest;
  }

  public static double getRestCompleted() {
    return restCompleted / rest;
  }

  public void addRestCompleted(double restCompleted) {
    SuggestionSystem.restCompleted += restCompleted;
  }

  public double getExercise() {
    return exercise;
  }

```

```
  public void addExercise(double exercise) {
    SuggestionSystem.exercise += exercise;
  }

  public static double getExerciseCompleted() {
    return exerciseCompleted / exercise;
  }

  public void addExerciseCompleted(double exerciseCompleted) {
    SuggestionSystem.exerciseCompleted += exerciseCompleted;
  }

}
```

B.2 Google Glass Application

B.2.1 Google Glass Landing Page

Listing B.6: HomeScreen Class

```
package com.ld.project.dylma;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;
import android.view.Window;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.Toast;

import com.google.android.glass.touchpad.Gesture;
import com.google.android.glass.touchpad.GestureDetector;
import com.google.android.glass.widget.CardBuilder;
import com.google.android.glass.widget.CardScrollAdapter;
import com.google.android.glass.widget.CardScrollView;
import com.ld.project.activity.Event;
import com.ld.project.activity.Task;
import com.ld.project.scheduler.Scheduler;
import com.ld.project.scheduler.SchedulerWrapper;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.ObjectInputStream;

public class HomeScreen extends Activity {

    private CardScrollView mCardScroller;

    private View mView;

    private GestureDetector mGestureDetector;

    public static Scheduler suggestionSource;

    public static Task taskList;
    public static Event eventCurrent;
    public static Event eventFuture;

    @Override
    public boolean onCreatePanelMenu(int featureId, Menu menu){
        if (featureId == Window.FEATURE_OPTIONS_PANEL) {
            getMenuInflater().inflate(R.menu.main, menu);
            return true;
        }
        return super.onCreatePanelMenu(featureId, menu);
    }


    @Override
    public boolean onMenuItemSelected(int featureId, MenuItem item) {
        if (featureId == Window.FEATURE_OPTIONS_PANEL) {
            switch (item.getItemId()) {
                case R.id.select_task:
                    moveToSecondScreen("SelectTask");
                    break;
                /*case R.id.skip_task:
```

```
                Toast.makeText(getApplicationContext(), "SKIP TASK NOT
    IMPLEMENTED", Toast.LENGTH_LONG).show();
                break;*/
            case R.id.select_activity:
                moveToSecondScreen("SelectActivity");
                break;
        }
        return true;
    }
    return super.onMenuItemSelected(featureId, item);
}


public void moveToSecondScreen(String activity){
    Intent resultsIntent = new Intent(this, ResultsActivity.class);
    resultsIntent.putExtra(ResultsActivity.SEARCH, activity);
    startActivity(resultsIntent);

}


private GestureDetector createGestureDetector(Context context) {
    GestureDetector gestureDetector = new GestureDetector(context);

    //Create a base listener for generic gestures
    gestureDetector.setBaseListener( new GestureDetector.BaseListener() {
        @Override
        public boolean onGesture(Gesture gesture) {
            if (gesture == Gesture.TAP) {
                openOptionsMenu();
                return true;
            } /*else if (gesture == Gesture.TWO_TAP) {
                // do something on two finger tap
                return true;
            } else if (gesture == Gesture.SWIPE_RIGHT) {
                // do something on right (forward) swipe
                return true;
            } else if (gesture == Gesture.SWIPE_LEFT) {
                // do something on left (backwards) swipe
                return true;
            } */else if (gesture == Gesture.SWIPE_DOWN){
                finish();
            }
            return false;
        }
    });

    gestureDetector.setFingerListener(new GestureDetector.FingerListener() {
        @Override
        public void onFingerCountChanged(int previousCount, int currentCount) {
            // do something on finger count changes
        }
    });

    gestureDetector.setScrollListener(new GestureDetector.ScrollListener() {
        @Override
        public boolean onScroll(float displacement, float delta, float velocity)
{
            // do something on scrolling
            return true;
        }
    });

    return gestureDetector;
}

@Override
public boolean onGenericMotionEvent(MotionEvent event) {
```

```
        if (mGestureDetector != null) {
            return mGestureDetector.onMotionEvent(event);
        }
        return false;
    }


    protected void onCreate(Bundle bundle) {

        super.onCreate(bundle);

        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        mView = buildView();
        mCardScroller = new CardScrollView(this);
        mCardScroller.setAdapter(new CardScrollAdapter() {
            @Override
            public int getCount() {
                return 1;
            }
            @Override
            public Object getItem(int position) {
                return mView;
            }
            @Override
            public View getView(int position, View convertView, ViewGroup parent) {
                return mView;
            }
            @Override
            public int getPosition(Object item) {
                if (mView.equals(item)) {
                    return 0;
                }
                return AdapterView.INVALID_POSITION;
            }
        });

        // Handle the TAP event.
        mCardScroller.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
    long id) {
                openOptionsMenu();
            }
        });

        if(HomeScreen.suggestionSource == null) {
            HomeScreen.suggestionSource = SchedulerWrapper.getSchedule();
        }

        eventCurrent = suggestionSource.getCurrentEvent();
        eventFuture = suggestionSource.getFutureEvent();

        if(taskList== null) {
            taskList = suggestionSource.getTask();
        }

        mGestureDetector = createGestureDetector(this);
        setContentView(mCardScroller);


    }

    @Override
    protected void onResume() {
        super.onResume();
        mCardScroller.activate();
    }
```

```
/*
    private void readFromFile() {
        if (suggestionSource == null) {
            try {
                FileInputStream fileIn = new FileInputStream("../DyLMA/src/taskList.
    ser");
                ObjectInputStream in = new ObjectInputStream(fileIn);
                suggestionSource = (Scheduler) in.readObject();
                in.close();
                fileIn.close();
            }

            // file not found: this is the first time program loaded: load
            // imaginary users with default values
            catch (FileNotFoundException nF) {
                System.out.println("File not found exception");
                suggestionSource = new Scheduler();

            } catch (IOException i) {
                i.printStackTrace();
                return;
            } catch (ClassNotFoundException c) {
                System.out.println("Schedule not found");
                c.printStackTrace();
                return;
            }

        }
        // customer list exists, not rewriting it
        else {
            System.out
                    .println("customer list already exists, no need to rewrite");
        }

    }
    */


    @Override
    protected void onPause() {
        mCardScroller.deactivate();

        super.onPause();
    }

    /**
     * Builds a Glass styled "Hello World!" view using the {@link CardBuilder} class.
     */
    private View buildView() {
        /*CardBuilder card = new CardBuilder(this, CardBuilder.Layout.TEXT);

        card.setText(R.string.hello_world);
        return card.getView();

        CardBuilder card = new CardBuilder(this);
        card.setText(R.string.app_name);
        card.setImageLayout(CardBuilder.ImageLayout.LEFT);
        card.addImage(R.drawable.logo);
        return card.getView();
        */


        View view1 = new CardBuilder(this, CardBuilder.Layout.COLUMNS)
                .setText(R.string.app_name) //"DyLMA"
                .addImage(R.drawable.logo)
                .getView();
```

```
        return view1;
    }

}
```

B.2.2 User Viewed Home Screens

Listing B.7: ResultsActivity Class

```
package com.ld.project.dylma;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.view.Window;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.HorizontalScrollView;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.glass.touchpad.Gesture;
import com.google.android.glass.touchpad.GestureDetector;
import com.google.android.glass.widget.CardBuilder;
import com.google.android.glass.widget.CardScrollAdapter;
import com.google.android.glass.widget.CardScrollView;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import com.ld.project.dylma.DetailedViewTask;
import com.ld.project.activity.*;
import com.ld.project.scheduler.Scheduler;
/*
    TODO Use temboo-android-sck-core-2.7.0.jar to extract google calendar information
    TODO Find out how to import iCloud info
    TODO All syncs should be in the ASyncTask doInBackground method
 */

public class ResultsActivity extends Activity {
    private int numEvents = 0;
    private int numTasks = 0;

    private String screenType;

    /*Contains all the info that will be displayed on the schedule*/
    private class cardInfo {
        String time;
        String taskname;
        public cardInfo(String x, String y) {
            time = y;
            taskname = x;
        }
    }
    private String[] dummy = new String[10];

    private final SimpleDateFormat mSDF = new SimpleDateFormat("MMM d    h:mm a");

    public static final String SEARCH = "search";
    private String mTask="Android";

```

```
    private CardScrollView mCardScroller;
    private List<View> mCards;
    private TaskAdapter mAdapter;
    private GestureDetector mGestureDetector;
    private ArrayAdapter<String> ad;
    private HorizontalScrollView lv;

    public void replaceCardWithNewOne(){
        cardInfo ci = new cardInfo(mTask, mSDF.format(new Date()));
        updateView(ci);
    }

    public View populateCard(cardInfo ci){

        if(ci.taskname.equals("SelectActivity")){
            screenType = "ActivityScreen";
        }
        else{
            screenType = "TaskScreen";
        }


        for (int i = 0; i < 10; i++) {
            dummy[i] = "Item " + i;
        }




            /* for building tasklist instead of single task
            taskList = HomeScreen.suggestionSource.getTaskList();

            for(int i = 0; i < 5; i++){
                if(taskList[i] != null){
                    numTasks++;
                }
            }
            */
            // for single task



        // creating the card array
        mCards = new ArrayList<View>();


        if(ci.taskname.equals("SelectActivity")) {
            // how many events there are:

            if(numEvents == 1) {
                CardBuilder card = new CardBuilder(this, CardBuilder.Layout.
    EMBED_INSIDE)
                        .setEmbeddedLayout(R.layout.layout);

                View cardView = card.getView();

                TextView date = (TextView) cardView.findViewById(R.id.date);
                date.setText(ci.time);
                TextView status = (TextView) cardView.findViewById(R.id.status);
                status.setText("   ");
                status.setTextColor(Color.RED);
                TextView toDo = (TextView) cardView.findViewById(R.id.current);
                toDo.setText("Currently:");
                TextView title = (TextView) cardView.findViewById(R.id.event_title);
                title.setText(HomeScreen.eventCurrent.getTitle());
                TextView upcoming = (TextView) cardView.findViewById(R.id.upcoming);
```

```
                upcoming.setText("Coming Up:");
                TextView next = (TextView) cardView.findViewById(R.id.next);
                next.setText("No More Events");
                TextView nextTime = (TextView) cardView.findViewById(R.id.nexttime);
                nextTime.setText("(--:--)");
                TextView nextDay = (TextView) cardView.findViewById(R.id.date_dif);
                nextDay.setText("");

                mCards.add(cardView);
                return cardView;
            }

            if(numEvents == 2) {
                CardBuilder card = new CardBuilder(this, CardBuilder.Layout.
    EMBED_INSIDE)
                        .setEmbeddedLayout(R.layout.layout);

                View cardView = card.getView();

                TextView date = (TextView) cardView.findViewById(R.id.date);
                date.setText(ci.time);
                TextView status = (TextView) cardView.findViewById(R.id.status);
                status.setText("    ");
                status.setTextColor(Color.RED);
                TextView toDo = (TextView) cardView.findViewById(R.id.current);
                toDo.setText("Currently:");
                TextView title = (TextView) cardView.findViewById(R.id.event_title);
                title.setText(HomeScreen.eventCurrent.getTitle());
                TextView upcoming = (TextView) cardView.findViewById(R.id.upcoming);
                upcoming.setText("Coming Up:");
                TextView next = (TextView) cardView.findViewById(R.id.next);
                next.setText(HomeScreen.eventFuture.getTitle());

                TextView nextTime = (TextView) cardView.findViewById(R.id.nexttime);
                SimpleDateFormat formatDateTime = new SimpleDateFormat("h:mm a");
                String startTime = formatDateTime.format(HomeScreen.eventFuture.
    getStart().getTime());
                String endTime = formatDateTime.format(HomeScreen.eventFuture.getEnd
    ().getTime());
                // if on the same day, trim off date
                String startDay = startTime.substring(0,5);
                String endDay = endTime.substring(0,5);
                if(startDay.equals(endDay)){
                    endTime = endTime.substring(6);
                }
                nextTime.setText(startTime + " - " + endTime);

                TextView nextDay = (TextView) cardView.findViewById(R.id.date_dif);
                SimpleDateFormat formatDate = new SimpleDateFormat("MM/dd");
                String startDate = formatDate.format(HomeScreen.eventFuture.getStart
    ().getTime());
                String endDate = formatDate.format(HomeScreen.eventFuture.getEnd().
    getTime());

                String today = formatDate.format(Calendar.getInstance().getTime());
                Calendar tomorrowCal = Calendar.getInstance();
                tomorrowCal.add(Calendar.DATE, 1);
                String tomorrow = formatDate.format(tomorrowCal.getTime());

                if(startDate.equals(today)) {
                    startDate = "Today";
                }
                if(endDate.equals(today)){
                    endDate = "Today";
                }
                if(startDate.equals(tomorrow)) {
                    startDate = "Tomorrow";
                }
```

```
                if(endDate.equals(tomorrow)){
                    endDate = "Tomorrow";
                }

                // if it starts and ends on the same day
                if(startDate.equals(endDate)){
                    nextDay.setText(startDate);
                }
                // different days
                else {
                    nextDay.setText(startDate + " - " + endDate);
                }

                mCards.add(cardView);
                return cardView;
            }
        }

        if(ci.taskname.equals("SelectTask")) {
            for (int i = 1; i < 2; i++) {
                CardBuilder card = new CardBuilder(this, CardBuilder.Layout.
EMBED_INSIDE)
                    .setEmbeddedLayout(R.layout.tasklayout);

                View cardView = card.getView();

                TextView date = (TextView) cardView.findViewById(R.id.date);
                date.setText(ci.time);
                TextView status = (TextView) cardView.findViewById(R.id.status);
                status.setText("   ");
                status.setTextColor(Color.GREEN);
                TextView toDo = (TextView) cardView.findViewById(R.id.toDo);
                toDo.setText("Tasks to Do:");
                if(numTasks == 1){
                    TextView title = (TextView) cardView.findViewById(R.id.task_list)
;
                    title.setText(HomeScreen.taskList.getTitle());
                }
                else {
                    TextView title = (TextView) cardView.findViewById(R.id.task_list)
;
                    title.setText("Tasks Completed");
                }
                // sets up the upcoming event
                if(numEvents > 0) {
                    TextView upcoming = (TextView) cardView.findViewById(R.id.
upcoming);
                    upcoming.setText("Coming Up:");
                    TextView next = (TextView) cardView.findViewById(R.id.next);
                    next.setText(HomeScreen.eventFuture.getTitle());
                    TextView nextTime = (TextView) cardView.findViewById(R.id.
nexttime);
                    SimpleDateFormat formatDateTime = new SimpleDateFormat("h:mm a");

                    String startTime = formatDateTime.format(HomeScreen.eventFuture.
getStart().getTime());
                    String endTime = formatDateTime.format(HomeScreen.eventFuture.
getEnd().getTime());

                    // if on the same day, trim off date
                    String startDay = startTime.substring(0,5);
                    String endDay = endTime.substring(0,5);
                    if(startDay.equals(endDay)){
                        endTime = endTime.substring(6);
                    }
```

```
                    nextTime.setText(startTime + " - " + endTime);

                    TextView nextDay = (TextView) cardView.findViewById(R.id.date_dif
);
                    SimpleDateFormat formatDate = new SimpleDateFormat("MM/dd");
                    String startDate = formatDate.format(HomeScreen.eventFuture.
getStart().getTime());
                    String endDate = formatDate.format(HomeScreen.eventFuture.getEnd
().getTime());

                    String today = formatDate.format(Calendar.getInstance().getTime()
);
                    Calendar tomorrowCal = Calendar.getInstance();
                    tomorrowCal.add(Calendar.DATE, 1);
                    String tomorrow = formatDate.format(tomorrowCal.getTime());

                    if(startDate.equals(today)) {
                        startDate = "Today";
                    }
                    if(endDate.equals(today)){
                        endDate = "Today";
                    }
                    if(startDate.equals(tomorrow)) {
                        startDate = "Tomorrow";
                    }
                    if(endDate.equals(tomorrow)){
                        endDate = "Tomorrow";
                    }

                    if(startDate.equals(endDate)){
                        nextDay.setText(startDate);
                    }
                    else {
                        nextDay.setText(startDate + " - " + endDate);
                    }


                }
                else{
                    TextView upcoming = (TextView) cardView.findViewById(R.id.
upcoming);
                    upcoming.setText("Coming Up:");
                    TextView next = (TextView) cardView.findViewById(R.id.next);
                    next.setText("No More Events");
                    TextView nextTime = (TextView) cardView.findViewById(R.id.
nexttime);
                    nextTime.setText("--:--");

                }

                mCards.add(cardView);
                return cardView;

            }
        }
       //mCardScroller.setSelection(0);
        return new View(null);
    }

    @Override
    public boolean onCreatePanelMenu(int featureId, Menu menu){
        if (featureId == Window.FEATURE_OPTIONS_PANEL) {

            if(screenType.equals("ActivityScreen")){
                getMenuInflater().inflate(R.menu.menu_results, menu);
            }
            else {
```

```
                getMenuInflater().inflate(R.menu.menu_results_task, menu);
            }
             return true;
        }
        return super.onCreatePanelMenu(featureId, menu);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        if(screenType.equals("ActivityScreen")){
            getMenuInflater().inflate(R.menu.menu_results, menu);
        }
        else{
            getMenuInflater().inflate(R.menu.menu_results_task, menu);
        }
        return true;
    }


    private GestureDetector createGestureDetector(Context context) {
        GestureDetector gestureDetector = new GestureDetector(context);

        //Create a base listener for generic gestures
        gestureDetector.setBaseListener( new GestureDetector.BaseListener() {
            @Override
            public boolean onGesture(Gesture gesture) {
                if (gesture == Gesture.TAP) {
                    openOptionsMenu();
                    return true;
                } else if (gesture == Gesture.TWO_TAP) {
                    // do something on two finger tap
                    return true;
                } else if (gesture == Gesture.SWIPE_RIGHT) {
                    // do something on right (forward) swipe
                    return true;
                } else if (gesture == Gesture.SWIPE_LEFT) {
                    // do something on left (backwards) swipe
                    return true;
                } else if (gesture == Gesture.SWIPE_DOWN){
                    finish();
                }
                return false;
            }
        });

        gestureDetector.setFingerListener(new GestureDetector.FingerListener() {
            @Override
            public void onFingerCountChanged(int previousCount, int currentCount) {
                // do something on finger count changes
            }
        });

        gestureDetector.setScrollListener(new GestureDetector.ScrollListener() {
            @Override
            public boolean onScroll(float displacement, float delta, float velocity)
    {
                // do something on scrolling
                return true;
            }
        });

        return gestureDetector;
    }

    @Override
    public boolean onMenuItemSelected(int featureId, MenuItem item) {
        if (featureId ==  Window.FEATURE_OPTIONS_PANEL) {
```

```
        Intent detailedIntent;
        switch (item.getItemId()) {
            case R.id.activity_is_complete:
                // complete the task: if more that one, need to identify which
one
                HomeScreen.taskList = HomeScreen.suggestionSource.complete(
HomeScreen.taskList);
                Toast.makeText(getApplicationContext(), "Activity Completed",
Toast.LENGTH_LONG).show();
                replaceCardWithNewOne();
                break;
            case R.id.remove_activity:
                // remove the task: if more that one, need to identify which one
                HomeScreen.taskList = HomeScreen.suggestionSource.remove(
HomeScreen.taskList);
                Toast.makeText(getApplicationContext(), "Activity Removed", Toast
.LENGTH_LONG).show();
                replaceCardWithNewOne();
                break;
            case R.id.skip_activity:
                // skip the task: if more that one, need to identify which one
                HomeScreen.taskList = HomeScreen.suggestionSource.skip(HomeScreen
.taskList);
                Toast.makeText(getApplicationContext(), "Activity Skipped", Toast
.LENGTH_LONG).show();
                replaceCardWithNewOne();
                break;
            case R.id.current_details:
                 detailedIntent = new Intent(this, DetailedView.class);
                detailedIntent.putExtra(SEARCH, HomeScreen.eventCurrent);
                startActivity(detailedIntent);
                break;
            // more info on tasks
            case R.id.task_details:
                 detailedIntent = new Intent(this, DetailedViewTask.class);
                detailedIntent.putExtra(SEARCH, HomeScreen.taskList);
                startActivity(detailedIntent);
                break;
            // more info on future event
            case R.id.event_details:
                 detailedIntent = new Intent(this, DetailedView.class);
                detailedIntent.putExtra(SEARCH, HomeScreen.eventFuture);
                startActivity(detailedIntent);
                break;
        }
        return true;
    }
    return super.onMenuItemSelected(featureId, item);
}



@Override
protected void onCreate(Bundle bundle) {
    super.onCreate(bundle);



    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

    if(getIntent().hasExtra(SEARCH)){
        mTask = getIntent().getStringExtra(SEARCH);
    }

    cardInfo ci = new cardInfo(mTask, mSDF.format(new Date()));
```

```
        /* for deciding which slide to go into
        // populating the list of tasks and events
        numEvents = 0;
        numTasks = 0;
        if(suggestionSource.isBusy()) {
            eventCurrent = suggestionSource.getCurrentEvent();
            eventFuture = suggestionSource.getFutureEvent();
            if(eventCurrent != null && eventFuture != null){
                numEvents = 2;
            }
            else if(eventFuture != null){
                numEvents = 1;
            }
        }
        // not busy: free state
        else {
            // for single task
            taskList = suggestionSource.getTask();
            if (taskList == null) {
                numTasks = 0;
            } else
                numTasks = 1;
            eventFuture = suggestionSource.getFutureEvent();
            if (eventFuture != null) {
                numEvents = 1;
            }
        }

        */

        // populating the list of tasks and events
        numEvents = 0;
        numTasks = 0;
        HomeScreen.eventCurrent = HomeScreen.suggestionSource.getCurrentEvent();
        HomeScreen.eventFuture = HomeScreen.suggestionSource.getFutureEvent();
            if(HomeScreen.eventCurrent != null && HomeScreen.eventFuture != null){
                numEvents = 2;
            }
            else if(HomeScreen.eventFuture != null) {
                numEvents = 1;
            }
        // not busy: free state
            // for single task
        if(HomeScreen.taskList== null) {
            HomeScreen.taskList = HomeScreen.suggestionSource.getTask();
        }
            if (HomeScreen.taskList == null) {
                numTasks = 0;
            } else {
                numTasks = 1;
            }


            updateView(ci);
    }

    private void updateView(cardInfo ci) {
        populateCard(ci);
        mCardScroller = new CardScrollView(this);
        mAdapter = new TaskAdapter();
        mCardScroller.setAdapter(mAdapter);
        mCardScroller.activate();
        // Handle the TAP event.
        mCardScroller.setOnItemClickListener(new
        AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position,
    long id) {
```

```
                openOptionsMenu();
            }
        });

        mGestureDetector = createGestureDetector(this);
        setContentView(mCardScroller);
    }

    /*@Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }*/


// internal task to make the card scroll work
    private class TaskAdapter extends CardScrollAdapter {
        //private List<CardBuilder> mCards;
        //public TaskAdapter(List<> cards){
        //    this.mCards = cards;
        //}
        @Override
        public int getCount() {
            return mCards.size();
        }
        @Override
        public Object getItem(int i) {
            return mCards.get(i);
        }
        @Override
        public View getView(int i, View view, ViewGroup viewGroup) {
            return mCards.get(i);
        }
        @Override
        public int getViewTypeCount() { return 1; }
        @Override
        public int getItemViewType(int position) { return 0; }
        @Override
        public int getPosition(Object o) {
            return mCards.indexOf(o);
        }
    }

    private class calSync extends AsyncTask<Void, Void, Void> {

        @Override
        protected Void doInBackground(Void... params) {

            //TODO: Sync calendars here

            return null;
        }


    }

}
```

B.2.3 Detailed Views on Google Glass

Listing B.8: Detailed View For Event

```
package com.ld.project.dylma;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.TextView;

import com.google.android.glass.touchpad.Gesture;
import com.google.android.glass.touchpad.GestureDetector;
import com.google.android.glass.widget.CardBuilder;
import com.google.android.glass.widget.CardScrollAdapter;
import com.google.android.glass.widget.CardScrollView;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import com.ld.project.activity.*;


public class DetailedView extends Activity {

    private String mTask="Android";
    public static final String SEARCH = "search";
    List<View> mCards;
    private CardScrollView mCardScroller;
    private TaskAdapter mAdapter;
    private GestureDetector mGestureDetector;
    private Event displayEvent;


    @Override
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

        if(getIntent().hasExtra(SEARCH)){
            displayEvent = (Event) getIntent().getSerializableExtra(SEARCH);
        }

        populateCard();
        mCardScroller = new CardScrollView(this);
        mAdapter = new TaskAdapter();
        mCardScroller.setAdapter(mAdapter);
        mCardScroller.activate();
        // Handle the TAP event.
        mCardScroller.setOnItemClickListener(new
                                                AdapterView.OnItemClickListener
    () {

                                                    @Override
                                                    public void onItemClick(
    AdapterView<?> parent, View view, int position, long id) {
                                                        openOptionsMenu();
                                                    }
                                                });

        mGestureDetector = createGestureDetector(this);
```

```
        setContentView(mCardScroller);
    }

    private GestureDetector createGestureDetector(Context context) {
        GestureDetector gestureDetector = new GestureDetector(context);

        //Create a base listener for generic gestures
        gestureDetector.setBaseListener( new GestureDetector.BaseListener() {
            @Override
            public boolean onGesture(Gesture gesture) {
                if (gesture == Gesture.TAP) {
                    //do something if one finger tap
                    return true;
                } else if (gesture == Gesture.TWO_TAP) {
                    // do something on two finger tap
                    return true;
                } else if (gesture == Gesture.SWIPE_RIGHT) {
                    // do something on right (forward) swipe
                    return true;
                } else if (gesture == Gesture.SWIPE_LEFT) {
                    // do something on left (backwards) swipe
                    return true;
                } else if (gesture == Gesture.SWIPE_DOWN){
                    finish();
                }
                return false;
            }
        });

        gestureDetector.setFingerListener(new GestureDetector.FingerListener() {
            @Override
            public void onFingerCountChanged(int previousCount, int currentCount) {
                // do something on finger count changes
            }
        });

        gestureDetector.setScrollListener(new GestureDetector.ScrollListener() {
            @Override
            public boolean onScroll(float displacement, float delta, float velocity)
    {
                // do something on scrolling
                return true;
            }
        });

        return gestureDetector;
    }

    public void populateCard() {

        // creating the card array
        mCards = new ArrayList<View>();

        for (int i = 1; i < 2; i++) {
            CardBuilder card = new CardBuilder(this, CardBuilder.Layout.EMBED_INSIDE)
                    .setEmbeddedLayout(R.layout.detailedlayout);

            View cardView = card.getView();

            TextView asdf = (TextView) cardView.findViewById(R.id.title);
            asdf.setText(displayEvent.getTitle());


            asdf = (TextView) cardView.findViewById(R.id.category);
            String categ = displayEvent.getType();
            asdf.setText(categ.toUpperCase());

            SimpleDateFormat formatDate = new SimpleDateFormat("MMM_d");
```

```
            SimpleDateFormat formatTime = new SimpleDateFormat("    h:mm a");


            String today = formatDate.format(Calendar.getInstance().getTime());
            Calendar tomorrowCal = Calendar.getInstance();
            tomorrowCal.add(Calendar.DATE, 1);
            String tomorrow = formatDate.format(tomorrowCal.getTime());

            String startDate = formatDate.format(displayEvent.getStart().getTime());
            String endDate = formatDate.format(displayEvent.getEnd().getTime());

            if(startDate.equals(today)) {
                startDate = "Today";
            }
            if(endDate.equals(today)){
                endDate = "Today";
            }
            if(startDate.equals(tomorrow)) {
                startDate = "Tomorrow";
            }
            if(endDate.equals(tomorrow)){
                endDate = "Tomorrow";
            }


            asdf = (TextView) cardView.findViewById(R.id.start);
            asdf.setText(
                    startDate + formatTime.format(displayEvent.getStart().getTime()))
    ;

            asdf = (TextView) cardView.findViewById(R.id.end);
            asdf.setText(
                    endDate + formatTime.format(displayEvent.getEnd().getTime()));

             asdf = (TextView) cardView.findViewById(R.id.description);

            asdf.setText(displayEvent.getDescription());
            mCards.add(cardView);
        }
    }

    /*@Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_detailed_view, menu);
        return true;
    }*/

    /*@Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }*/

    private class TaskAdapter extends CardScrollAdapter {
        //private List<CardBuilder> mCards;
        //public TaskAdapter(List<> cards){
        //    this.mCards = cards;
        //}
```

```
        @Override
        public int getCount() {
            return mCards.size();
        }
        @Override
        public Object getItem(int i) {
            return mCards.get(i);
        }
        @Override
        public View getView(int i, View view, ViewGroup viewGroup) {
            return mCards.get(i);
        }
        @Override
        public int getViewTypeCount() { return 1; }
        @Override
        public int getItemViewType(int position) { return 0; }
        @Override
        public int getPosition(Object o) {
            return mCards.indexOf(o);
        }
    }
}
```

Listing B.9: Detailed View for Tasks

```
package com.ld.project.dylma;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.TextView;

import com.google.android.glass.touchpad.Gesture;
import com.google.android.glass.touchpad.GestureDetector;
import com.google.android.glass.widget.CardBuilder;
import com.google.android.glass.widget.CardScrollAdapter;
import com.google.android.glass.widget.CardScrollView;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import com.ld.project.activity.*;


public class DetailedView extends Activity {

    private String mTask="Android";
    public static final String SEARCH = "search";
    List<View> mCards;
    private CardScrollView mCardScroller;
    private TaskAdapter mAdapter;
    private GestureDetector mGestureDetector;
    private Event displayEvent;


    @Override
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

        if(getIntent().hasExtra(SEARCH)){
            displayEvent = (Event) getIntent().getSerializableExtra(SEARCH);
        }

        populateCard();
        mCardScroller = new CardScrollView(this);
        mAdapter = new TaskAdapter();
        mCardScroller.setAdapter(mAdapter);
        mCardScroller.activate();
        // Handle the TAP event.
        mCardScroller.setOnItemClickListener(new
                                                AdapterView.OnItemClickListener
    () {

                                                    @Override
                                                    public void onItemClick(
    AdapterView<?> parent, View view, int position, long id) {
                                                        openOptionsMenu();
                                                    }
                                                });

        mGestureDetector = createGestureDetector(this);
        setContentView(mCardScroller);
    }

```

```
    private GestureDetector createGestureDetector(Context context) {
        GestureDetector gestureDetector = new GestureDetector(context);

        //Create a base listener for generic gestures
        gestureDetector.setBaseListener( new GestureDetector.BaseListener() {
            @Override
            public boolean onGesture(Gesture gesture) {
                if (gesture == Gesture.TAP) {
                    //do something if one finger tap
                    return true;
                } else if (gesture == Gesture.TWO_TAP) {
                    // do something on two finger tap
                    return true;
                } else if (gesture == Gesture.SWIPE_RIGHT) {
                    // do something on right (forward) swipe
                    return true;
                } else if (gesture == Gesture.SWIPE_LEFT) {
                    // do something on left (backwards) swipe
                    return true;
                } else if (gesture == Gesture.SWIPE_DOWN){
                    finish();
                }
                return false;
            }
        });

        gestureDetector.setFingerListener(new GestureDetector.FingerListener() {
            @Override
            public void onFingerCountChanged(int previousCount, int currentCount) {
                // do something on finger count changes
            }
        });

        gestureDetector.setScrollListener(new GestureDetector.ScrollListener() {
            @Override
            public boolean onScroll(float displacement, float delta, float velocity)
    {
                // do something on scrolling
                return true;
            }
        });

        return gestureDetector;
    }

    public void populateCard() {

        // creating the card array
        mCards = new ArrayList<View>();

        for (int i = 1; i < 2; i++) {
            CardBuilder card = new CardBuilder(this, CardBuilder.Layout.EMBED_INSIDE)
                    .setEmbeddedLayout(R.layout.detailedlayout);

            View cardView = card.getView();

            TextView asdf = (TextView) cardView.findViewById(R.id.title);
            asdf.setText(displayEvent.getTitle());


            asdf = (TextView) cardView.findViewById(R.id.category);
            String categ = displayEvent.getType();
            asdf.setText(categ.toUpperCase());

            SimpleDateFormat formatDate = new SimpleDateFormat("MMM_d");
            SimpleDateFormat formatTime = new SimpleDateFormat("____h:mm_a");
```

```
            String today = formatDate.format(Calendar.getInstance().getTime());
            Calendar tomorrowCal = Calendar.getInstance();
            tomorrowCal.add(Calendar.DATE, 1);
            String tomorrow = formatDate.format(tomorrowCal.getTime());

            String startDate = formatDate.format(displayEvent.getStart().getTime());
            String endDate = formatDate.format(displayEvent.getEnd().getTime());

            if(startDate.equals(today)) {
                startDate = "Today";
            }
            if(endDate.equals(today)){
                endDate = "Today";
            }
            if(startDate.equals(tomorrow)) {
                startDate = "Tomorrow";
            }
            if(endDate.equals(tomorrow)){
                endDate = "Tomorrow";
            }


            asdf = (TextView) cardView.findViewById(R.id.start);
            asdf.setText(
                    startDate + formatTime.format(displayEvent.getStart().getTime()))
    ;

            asdf = (TextView) cardView.findViewById(R.id.end);
            asdf.setText(
                    endDate + formatTime.format(displayEvent.getEnd().getTime()));

             asdf = (TextView) cardView.findViewById(R.id.description);

            asdf.setText(displayEvent.getDescription());
            mCards.add(cardView);
        }
    }

    /*@Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_detailed_view, menu);
        return true;
    }*/

    /*@Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }*/

    private class TaskAdapter extends CardScrollAdapter {
        //private List<CardBuilder> mCards;
        //public TaskAdapter(List<> cards){
        //    this.mCards = cards;
        //}
        @Override
        public int getCount() {
            return mCards.size();
```

```
200         }
            @Override
            public Object getItem(int i) {
                return mCards.get(i);
            }
205         @Override
            public View getView(int i, View view, ViewGroup viewGroup) {
                return mCards.get(i);
            }
            @Override
210         public int getViewTypeCount() { return 1; }
            @Override
            public int getItemViewType(int position) { return 0; }
            @Override
            public int getPosition(Object o) {
215             return mCards.indexOf(o);
            }
        }
    }
```

B.3 Android Application

B.3.1 Google Calendar Connection

The following code is part of the Android application and it is significant because it establishes a connection from the application to the user's Google account and it also manages the connection to the internal database. The GoogleCalendarConnection prompts the user to input his/her email address and password and then the user is asked if the application has permission to view and edit the user's tasks and calendar events. After receiving permission, the code will generate the correct retrieve commands to retrieve all of the user's entered tasks in Google and all the users events that are scheduled for the next twenty-four hours. The information is retrieved and packaged into JSON objects, which are then compacted into JSON arrays. The retrieval of the data from Googles servers is done through an Asynchronous task so that the main UI thread is not blocked and the user does not lose interaction abilities with the application. After the data is retrieved, the data that has been packaged into JSON objects in JSON arrays will be also stored into the devices internal memory via an SQLite database. This is where the second class becomes significant. An instance of the second class, MySQLiteHelper, is the manager of the database stored internal to the device running the application. The data stays persistent and will always be available to the application. The connection to Googles calendar will be made when the application first runs, when updates are made through the application, or when data has been added to the Google account via another means, for example, when the user adds data through a desktop computer. The code is packaged into a separate class to simplify the use of the Google server connection. An instance of the GoogleCalendarConnection class will serve as a sort of meta-Api because it acts as a quasi-Api for the Apis Google already provides. It was created in this fashion to further simplify the data transfer between Google servers and the Android application. The instance of the GoogleCalendarConnection class has the ability to push and pull data from both the internal database and from Google Servers.

Listing B.10: GoogleCalendarConnection Class

```
public class GoogleCalendarConnection {
    private HttpTransport transport;
    private JsonFactory jsonFactory;
    private Calendar calClient;
    com.google.api.services.tasks.Tasks taskClient;
    private String accountName;
    private Context _context;

    private String Scopes; // contains both CalendarScopes.Calendar and TasksScopes.
    TASKS

    private JSONArray listEvents;       // array of json objects that holds info
    about each event
    private JSONArray listTasks;        // array of json objects that hold info about
     each task

    private ProgressDialog dialog;
    private Intent intent;
    //private final AsyncTask<Void, Void, String> aTask;

    // connects to google calendar and google tasks
    // retrieves the day's event's
    // retrieves the user's tasks that are due within the next 2 weeks
    public GoogleCalendarConnection(final Context context, final GoogleApiClient
    mGoogleApiClient, final ProgressDialog _dialog, Intent _intent) throws
    ExecutionException, InterruptedException {
        _context = context;
        accountName = Plus.AccountApi.getAccountName(mGoogleApiClient);
        listTasks = new JSONArray();
        listEvents = new JSONArray();
        Scopes = "https://www.googleapis.com/auth/calendar" + " " + "https://www.
    googleapis.com/auth/tasks";
```

```
        dialog = _dialog;
        intent = _intent;

        final AsyncTask<Void, Void, String> aTask = new AsyncTask<Void, Void, String
    >() {
            @Override
            protected void onPreExecute(){
                dialog.setMessage("Pulling Data..");
                dialog.show();
            }


            @Override
            protected String doInBackground(Void... params) {
                String token = null;
                String taskToken = null;

                try {
                    transport = new NetHttpTransport();
                    jsonFactory = new JacksonFactory();

                    GoogleAccountCredential credential = GoogleAccountCredential.
    usingOAuth2(context, Collections.singleton(Scopes));
                    credential.setSelectedAccountName(accountName);

                    try {
                        token = credential.getToken();
                    } catch (UserRecoverableAuthException e) {
                        //startActivityForResult(e.getIntent(), 1);
                        _context.startActivity(e.getIntent().addFlags(Intent.
    FLAG_ACTIVITY_NEW_TASK));
                    } catch (GoogleAuthException e) {
                        e.printStackTrace();
                    }

                    // creating client to retrieve calendar events
                    calClient = new Calendar.Builder(transport, jsonFactory,
    credential)
                            .setApplicationName("Google-CalendarAndroid")
                            .build();

                    java.util.Calendar cal = java.util.Calendar.getInstance();
                    cal.add(java.util.Calendar.DATE, 0);
                    cal.set(java.util.Calendar.HOUR_OF_DAY, 0);
                    cal.set(java.util.Calendar.MINUTE, 0);
                    cal.set(java.util.Calendar.SECOND, 0);
                    Date begToday = cal.getTime();

                    java.util.Calendar cal2 = java.util.Calendar.getInstance();
                    cal2.add(java.util.Calendar.DATE, 0);
                    cal2.set(java.util.Calendar.HOUR_OF_DAY, 23);
                    cal2.set(java.util.Calendar.MINUTE, 59);
                    cal2.set(java.util.Calendar.SECOND, 59);
                    Date endToday = cal2.getTime();

                    DateTime dtBegToday = new DateTime(begToday, TimeZone.getDefault
    ());
                    DateTime dtEndToday = new DateTime(endToday, TimeZone.getDefault
    ());


                    String pageToken = null;
                    JSONObject obj;
                    do {
                        Events events = calClient.events().list(accountName)
                                .setPageToken(pageToken)
                                .setTimeMax(dtEndToday)
                                .setTimeMin(dtBegToday)
```

```
                                .execute();

                        List<Event> items = events.getItems();
                        for (Event event : items) {
                            obj = new JSONObject();
                            try {
                                obj.put("Id", event.getId());
                            } catch (JSONException e) {
                                Log.i("MyApp-Events", "Error retrieving ID");
                                e.printStackTrace();
                            }
                            try {
                                obj.put("Title", event.getSummary());
                            } catch (JSONException e) {
                                Log.i("MyApp-Events", "Error retrieving Title");
                                e.printStackTrace();
                            }
                            try {
                                obj.put("Details", event.getDescription());
                            } catch (JSONException e) {
                                Log.i("MyApp-Events", "Error retrieving Details");
                                e.printStackTrace();
                            }
                            try {
                                obj.put("StartTime", event.getStart());
                            } catch (JSONException e) {
                                Log.i("MyApp-Events", "Error retrieving Start Time");
                                e.printStackTrace();
                            }
                            try {
                                obj.put("EndTime", event.getEnd());
                            } catch (JSONException e) {
                                Log.i("MyApp-Events", "Error retrieving End Time");
                                e.printStackTrace();
                            }
                            listEvents.put(obj);
                        }

                        pageToken = events.getNextPageToken();
                    } while (pageToken != null);


                    // creating client to retrieve tasks
                    taskClient = new com.google.api.services.tasks.Tasks.Builder(
transport, jsonFactory, credential)
                            .setApplicationName("Google-TasksAndroid").build();

                    com.google.api.services.tasks.model.Tasks t = taskClient.tasks().
list("@default")
                            .setShowCompleted(false)
                            .setShowDeleted(false)
                            .setDueMin(String.valueOf(dtBegToday))
                            .setFields("items")
                            .execute();

                    JSONObject o;
                    if(t.size() > 0 && (!t.isEmpty()))
                        for (Task a : t.getItems()) {  // error when empty
                            Log.i("MyApp-Tasks", a.getTitle());
                            o = new JSONObject();
                            try {
                                o.put("Id", a.getId());
                            } catch (JSONException e) {
                                Log.i("MyApp-Tasks", "Error retrieving ID");
                                e.printStackTrace();
                            }
                            try {
                                o.put("Title", a.getTitle());
```

```
                            } catch (JSONException e) {
                                Log.i("MyApp-Tasks", "Error retrieving Title");
                                e.printStackTrace();
                            }
                            try {
                                o.put("Description", a.getNotes());
                            } catch (JSONException e) {
                                Log.i("MyApp-Tasks", "Error retrieving Description");
                                e.printStackTrace();
                            }
                            try {
                                o.put("DueDate", a.getDue());
                            } catch (JSONException e) {
                                Log.i("MyApp-Tasks", "Error retrieving Due Date");
                                e.printStackTrace();
                            }
                            try {
                                o.put("Status", a.getStatus());
                            } catch (JSONException e) {
                                Log.i("MyApp-Tasks", "Error retrieving status");
                                e.printStackTrace();
                            }
                            try {
                                o.put("Priority", "3");
                            } catch (JSONException e) {
                                Log.i("MyApp-Tasks", "Error inserting Default
    priority of 3");
                                e.printStackTrace();
                            }
                            listTasks.put(o);
                        }
                    } catch (IOException e) {
                        Log.i("My Activity", "Error Found when connecting to google
    calendar");
                        e.printStackTrace();
                    }


                    return token;
                }

                @Override
                protected void onPostExecute(String token) {

                    Log.i("FINISHED CONNECTING", "FINISHED CONNECTING");
                    if(dialog.isShowing()){
                        dialog.dismiss();
                    }
                    MySQLiteHelper db = new MySQLiteHelper(_context.getApplicationContext
    ());

                    if(listEvents.length() > 0){
                        for(int j = 0; j < listEvents.length(); j ++) {
                            if(j == 0 || j == 1){

                            }

                            try {
                                db.insertEventToDB(listEvents.getJSONObject(j));
                            } catch (JSONException e) {
                                e.printStackTrace();
                            }
                        }
                    }

                    if(listTasks.length() > 0){
                        for(int i = 0; i < listTasks.length(); i ++) {
                            try {
```

```
                        db.insertTaskToDB(listTasks.getJSONObject(i));
                    } catch (JSONException e){
                        e.printStackTrace();
                    }
                }
            }

            db.close();
        }

    };
    aTask.execute();
}


public String getAccountName(){
    return accountName;
}


public JSONArray getListOfTasks(){
    if(listTasks.length() > 0){
        return listTasks;
    }else{
        Log.i("MyApp", "listTasks is null");
        return null;
    }
}


public JSONArray getListOfEvents(){
    if(listEvents.length() > 0)
        return listEvents;
    else {
        Log.i("MyApp", "listEvents is null");
        return null;
    }
}


public void addCalEntry(String _title, String _details, EventDateTime _start,
EventDateTime _end){

    Event event = new Event();
    event.setSummary(_title);
    event.setDescription(_details);
    event.setStart(_start);
    event.setEnd(_end);

    try {
        calClient.events().insert("primary", event).execute();
    } catch (IOException e) {
        e.printStackTrace();
        Log.i("AddCalEntry", "Error Adding Calendar Entry");
    }

}


public void addCalEntryDetails(String _id, String _details){
    Event event = null;

    try {
        event = calClient.events().get("primary", _id).execute();
    } catch (IOException e) {
        e.printStackTrace();
        Log.i("AddCalEntryDetail", "Error Retrieving Event before updating");
    }
```

```
290         event.setSummary(_details);

            try {
                calClient.events().update("primary", event.getId(), event).execute();
            } catch (IOException e) {
295             e.printStackTrace();
                Log.i("AddCalEntryDetail", "Error updating event details to server");
            }
        }


300
        public void addTaskEntry(String _title, String _details, DateTime _due){
            Task task = new Task();
            task.setTitle(_title);
            task.setNotes(_details);
305         task.setDue(_due);

            try {
                taskClient.tasks().insert("@default", task).execute();
            } catch (IOException e) {
310             Log.i("AddTask", "Could not add Task Entry");
                e.printStackTrace();
            }

        }
315


        public void addTaskEntryDetails(String _id, String _details){
            Task task = null;
            try {
320             task = taskClient.tasks().get("@default", _id).execute();
            } catch (IOException e) {
                e.printStackTrace();
                Log.i("AddTaskDetails", "Error Retrieving Task before updating");
            }
325
            task.setNotes(_details);

            try {
                taskClient.tasks().update("@default", task.getId(), task).execute();
330         } catch (IOException e) {
                e.printStackTrace();
                Log.i("AddTaskDetails", "Error updating Task details to Server");
            }


335
        }

        // saves to google calendar
        public void completeTaskEntry(String _id){
340         Task task = null;
            try {
                task = taskClient.tasks().get("@default", _id).execute();
            } catch (IOException e) {
                e.printStackTrace();
345             Log.i("AddTaskDetails", "Error Retrieving Task before updating");
            }

            // status either: needsAction or completed
            task.setStatus("completed");
350
            try {
                taskClient.tasks().update("@default", task.getId(), task).execute();
            } catch (IOException e) {
                Log.i("AddTaskDetails", "Error updating Task status to Server");
355             e.printStackTrace();
            }
```

```
    }


}




/*
    Event Table Columns
    eventID      text
    title        text
    description  text
    startTime    text
    endTime      text


    Task Table Columns
    taskID       text
    title        text
    description  text
    dueDate      text
    status       text
    priority     INTEGER
 */
```

Listing B.11: MYSQLiteHelper Class

```
public class MySQLiteHelper extends SQLiteOpenHelper {

    // Database Version
    private static final int DATABASE_VERSION = 1;
    // Database Name
    private static final String DATABASE_NAME = "Dilemma";


    public MySQLiteHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //SQL statement to create dilemma table
        String CREATE_TASKS_TABLE = "CREATE TABLE tasks ( " + "id INTEGER PRIMARY KEY
    AUTOINCREMENT, " + "taskID TEXT, " + "title TEXT, " + "description TEXT, " + "
    dueDate TEXT, " + "status TEXT, " + "priority INTEGER )";
        String CREATE_EVENTS_TABLE = "CREATE TABLE events ( " + "id INTEGER PRIMARY
    KEY AUTOINCREMENT, " + "eventID TEXT, " + "title TEXT, " + "description TEXT, " +
     "startTime TEXT, " + "endTime TEXT )";

        //create dilemma table
        db.execSQL(CREATE_TASKS_TABLE);
        db.execSQL(CREATE_EVENTS_TABLE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        //Drop older dilemma table if existed
        db.execSQL("DROP TABLE IF EXISTS tasks");
        db.execSQL("DROP TABLE IF EXISTS events");

        //create fresh tasks table
        this.onCreate(db);
    }


    public JSONArray getAllTasks() throws JSONException {
```

```
        JSONArray taskList = new JSONArray();
        JSONObject obj;

        String selectQuery = "SELECT * FROM tasks";
        SQLiteDatabase db = this.getReadableDatabase();

        Cursor c = db.rawQuery(selectQuery, null);

        if(c.moveToFirst()){
            do{
                obj = new JSONObject();

                obj.put("Id", c.getColumnIndex("taskID"));
                obj.put("Title", c.getColumnIndex("title"));
                obj.put("Description", c.getColumnIndex("description"));
                obj.put("DueDate", c.getColumnIndex("dueDate"));
                obj.put("Status", c.getColumnIndex("status"));
                obj.put("Priority", c.getColumnIndex("priority"));

                taskList.put(obj);
            } while (c.moveToNext());
        }

        return taskList;
    }


    public void insertTaskToDB(JSONObject o) throws JSONException {
        if (o == null) {
            Log.i("Dilemma", "Task JSONObject is empty");
            return;
        }

        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();
        values.put("taskID", o.get("Id").toString());
        values.put("title", o.get("Title").toString());
        values.put("description", o.get("Description").toString());
        values.put("dueDate", o.get("DueDate").toString());
        values.put("status", o.get("Status").toString());
        values.put("priority", o.get("Priority").toString());

        //insert row to table
        db.insert("tasks", null, values);
    }


    public void deleteTask(String taskId){
        SQLiteDatabase db = this.getWritableDatabase();

        db.delete("tasks", "taskID=?", new String[]{taskId});
    }


    public void updateTask(JSONObject o) throws JSONException {
        if (o == null) {
            Log.i("Dilemma", "Task JSONObject is empty");
            return;
        }

        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();

        values.put("taskID", o.get("Id").toString());
        values.put("title", o.get("Title").toString());
        values.put("description", o.get("Description").toString());
```

```
        values.put("dueDate", o.get("DueDate").toString());
        values.put("status", o.get("Status").toString());
        values.put("priority", o.get("Priority").toString());

        db.update("tasks", values, "taskID = " + o.get("Id").toString(), null);

    }


    public JSONArray getAllEvents() throws JSONException {
        JSONArray eventList = new JSONArray();
        JSONObject obj = new JSONObject();

        String selectQuery = "SELECT * FROM events";
        SQLiteDatabase db = this.getReadableDatabase();

        Cursor c = db.rawQuery(selectQuery, null);

        if(c.moveToFirst()){
            do{
                obj = new JSONObject();

                obj.put("Id", c.getColumnIndex("eventId"));
                obj.put("Title", c.getColumnIndex("title"));
                obj.put("Details", c.getColumnIndex("description"));
                obj.put("StartTime", c.getColumnIndex("startTime"));
                obj.put("EndTime", c.getColumnIndex("endTime"));

                eventList.put(obj);
            } while (c.moveToNext());
        }

        return eventList;
    }


    public void insertEventToDB(JSONObject o) throws JSONException {
        if (o == null) {
            Log.i("Dilemma", "Event JSONObject is empty");
            return;
        }

        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();
        values.put("eventID", o.get("Id").toString());
        values.put("title", o.get("Title").toString());
        values.put("description", o.get("Details").toString());
        values.put("startTime", o.get("StartTime").toString());
        values.put("endTime", o.get("EndTime").toString());

        db.insert("events", null, values);
    }


    public void deleteEvent(String eventId){
        SQLiteDatabase db = this.getWritableDatabase();

        db.delete("events", "eventID=?", new String[]{eventId});
    }


    public void updateEvent(JSONObject o) throws JSONException {
        if (o == null) {
            Log.i("Dilemma", "Event JSONObject is empty");
            return;
        }
```

```
173         SQLiteDatabase db = this.getWritableDatabase();

            ContentValues values = new ContentValues();

            values.put("eventID", o.get("Id").toString());
178         values.put("title", o.get("Title").toString());
            values.put("description", o.get("Details").toString());
            values.put("startTime", o.get("StartTime").toString());
            values.put("endTime", o.get("EndTime").toString());

183         db.update("events", values, "eventID = " + o.get("Id").toString(), null);
        }


        public int numTasksInDb(){
188         SQLiteDatabase db = this.getReadableDatabase();

            Cursor cursor = db.rawQuery("SELECT * FROM tasks", null);

            return cursor.getCount();
193     }

        public int numEventsInDb(){
            SQLiteDatabase db = this.getReadableDatabase();

198         Cursor cursor = db.rawQuery("SELECT * FROM events", null);

            return cursor.getCount();
        }

203     public void closeDB(){
            SQLiteDatabase db = this.getReadableDatabase();
            if(db != null && db.isOpen()){
                db.close();
            }
208     }

}
```

B.3.2 Bluetooth Connection

Listing B.12: Bluetooth Data Transfer

```
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.content.Intent;
import android.os.Handler;
import android.os.Message;
import android.util.Log;

import org.json.JSONException;
import org.json.JSONObject;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Set;
import java.util.UUID;

public class DataTransfer{
    //Constants
    private final static int REQUEST_ENABLE_BT = 1; //1
    private static final UUID MY_UUID = UUID.fromString("
    00001101-0000-1000-8000-00805f9b34fb");

    //Instance variables
    private BluetoothAdapter mainBluetoothAdapter;
    private BluetoothDevice mainDevice;
    private ConnectThread connectionThread;
    private ConnectedThread dataThread;
    private Handler mainHandler;


    public DataTransfer(Context _context) { //Testing out the constructor to fix the
    errors with the unknown symbol.
        mainBluetoothAdapter = BluetoothAdapter.getDefaultAdapter(); //Obtains
    bluetooth adapters

        //Preliminary bluetooth checks
        if (mainBluetoothAdapter == null)
            Log.w("Error", "The device does not support bluetooth");
        if (!mainBluetoothAdapter.isEnabled()) {
            Intent enableBluetoothIntent = new Intent(BluetoothAdapter.
    ACTION_REQUEST_ENABLE);
            _context.startActivity(enableBluetoothIntent.addFlags(Intent.
    FLAG_ACTIVITY_NEW_TASK));
        }

        //Obtains all paired devices
        Set<BluetoothDevice> pairedDevices = mainBluetoothAdapter.getBondedDevices();
        if (pairedDevices.size() > 0 || !pairedDevices.isEmpty()) {
            for (BluetoothDevice device : pairedDevices)
                mainDevice = device;
        }

        //Defines the Handler to handle the data by converting the byte array to a
    string.
        mainHandler = new Handler() {
            @Override
            public void handleMessage(Message message) {
                byte[] writeBuffer = (byte[]) message.obj;

                JSONObject testingTransfer = new JSONObject();
                try {
```

```
                    testingTransfer.put("Test", "correct");
                } catch (JSONException e) {
                    e.printStackTrace();
                }

                int start = (int) message.arg1;
                int end = (int) message.arg2;

                switch (message.what) {
                    case 1:
                        String writeMessage = new String(writeBuffer);
                        writeMessage = writeMessage.substring(start, end);
                        break;
                }
            }
        };
        if(!(mainDevice == null)) {
            connectionThread = new ConnectThread(mainDevice);
            connectionThread.start(); //Begins the alternate thread for the bluetooth connection.

            dataThread = new ConnectedThread(connectionThread.getMainSocket());
            dataThread.start(); //Begins alternate thread for bluetooth transfer
        }
        else{
            Log.i("TEST_DATA", "NO_DEVICES_FOUND");
        }
    }


    //Inner class to define a new connection thread to handle data transfer
    private class ConnectThread extends Thread {
        //Instance variables
        private final BluetoothSocket mainSocket;
        private final BluetoothDevice connectedMainDevice;

        public ConnectThread(BluetoothDevice device) {
            BluetoothSocket socketAttempt = null;
            connectedMainDevice = device;

            try {
                socketAttempt = device.createRfcommSocketToServiceRecord(MY_UUID);
            } catch (IOException e) {
                Log.w("Error", "Failed_to_create_the_socket.");
            }

            mainSocket = socketAttempt;
        }

        public void run() {
            mainBluetoothAdapter.cancelDiscovery(); //Makes sure the device is not searching for devices.

            try {
                mainSocket.connect();
            } catch (IOException eConnect) {
                Log.w("Error", "Failed_to_connect_socket");
                try {
                    mainSocket.close();
                } catch (IOException eClose) {
                    Log.w("Error", "Failed_to_close_socket_after_a_failed_connect.");
                }
                //return;
            }
        }

        public void cancel() {
            try {
```

```
                mainSocket.close();
            } catch (IOException e) {
                Log.w("Error", "Failed to close socket.");
            }
        }

        public BluetoothSocket getMainSocket() {
            return mainSocket;
        }
    }

    //Inner class to handle the data transfer
    private class ConnectedThread extends Thread {
        //Instance variables
        private final BluetoothSocket connectedSocket;
        private final InputStream mainInputStream;
        private final OutputStream mainOutputStream;

        public ConnectedThread(BluetoothSocket socket) {
            connectedSocket = socket;
            InputStream attemptInput = null;
            OutputStream attemptOutput = null;

            try {
                attemptInput = connectedSocket.getInputStream();
                attemptOutput = connectedSocket.getOutputStream();
            } catch (IOException e) {
                Log.w("Error", "Failed to obtain input or output stream");
            }

            mainInputStream = attemptInput;
            mainOutputStream = attemptOutput;
        }

        public void run() {
            byte[] buffer = new byte[1024]; //New serial buffer
            int counter = 0;
            int numberOfBytes = 0;

            while (true) {
                try {
                    numberOfBytes += mainInputStream.read(buffer, numberOfBytes,
    buffer.length - numberOfBytes);

                    for (int i = counter; i < numberOfBytes; i++) {
                        mainHandler.obtainMessage(1, counter, i, buffer).sendToTarget
    ();
                        counter++;
                        if (i == numberOfBytes - 1) { //If all the bytes are read,
    then it resets the counters.
                            numberOfBytes = 0;
                            counter = 0;
                        }
                    }
                } catch (IOException e) {
                    Log.w("Error", "Failed to read from the input stream");
                    break;
                }
            }
        }

        public void write(byte[] bytes) {
            try {
                mainOutputStream.write(bytes);
            } catch (IOException e) {
                Log.w("Error", "Failed to write to output stream");
            }
        }
```

```
        public void cancel() {
            try {
193             connectedSocket.close();
            } catch (IOException e) {
                Log.w("Error", "Failed to close data connected socket.");
            }
        }
198 }
}
```