Palantir-B System

Software Requirements and Design Document (SRDD)

9-30-2001

Prepared for:
Bennett

Prepared by

Team D
(**F**ernando, **I**ndu, **L**iang, **U**dai, **C**indy, **T**ony)

Ac3@cec.wustl.edu

http://userfs.cec.wustl.edu/~ac3/

Table of Contents

List of Figures & Tables

Revision History

9/30/01

Replace System Context diagram

Updated requirements

Updated Actors and User-cases

Inserted Use-case diagram

Updated User-interface Section

Added Design Approach Section

Updated Analysis /Design Section

Inserted new State, Activity diagrams

Updated External Storage Section

Updated External Interface Section

1 INTRODUCTION

The Palantir-B system is an on-going project that collects data from the atmosphere and send it back to its end-users. The system consists of 2 parts – Flight System and the Ground Station System. The Flight System controls sensors on a microsatellite sent to the atmosphere by Project Aria, a research program led by the School of Engineering of Washington University. The Flight System is also responsible for self-power management and data transmission back to the Ground Station. The Ground Station System allows user to request data and display them in a graphical user interface.

1.1 Purpose of the System

Palantir-B is to provide its users, which range from K-12 students to researchers around the world, reliable data from the microsatellites of Project Aria for different experiments.

Our potential customers might also be our end-user; for example, atmospheric researchers. While other customers might be an intermediate link to the end-user. This could be school teachers purchasing the software for high school students.

While processing those raw data from the microsatellites and sending them to clients will be straightforward, the challenge of this project is to overcome transmission delay. Furthermore, the solar power supply of the satellite is not stable. Thus, besides dealing with data transmission and multiple clients, our system also need to "power-manage" the satellite.

1.2 Scope of the System

The Palantir-B is a system that builds on top of the sensor packages of the microsatellites from Project Aria. It also provides an interface that abstracts away the complexities of transmission delay.

1.3 Objective and Success Criteria of the Project

The criteria for success in this project is

- To provide communication between the flight system with the ground station system with the constraint of unreliable data transmission
- To have the system be able to power manage itself in all imaginable condition
- To allow the system to be upgraded with minimal effort

1.4 Definition of Terms and Acronyms

- Palantir A – Device for collecting information about an environment, and relaying that information to a ground station
- Palantir B – Next generation Palantir, designed to have greater data collection ability. Does not yet exist in physical form.

1.5 References

- Project Aria Program Description

- Palantir TD Drawings
- SSC01-VIII-6 Paper
- In-class handout (project reqs)

2 SYSTEM CONTEXT

2.1 Current System

The current system consists of everything developed for Washington University's Project Aria. This includes the Palantir CubeSat program and the Palantir Technology Demonstration balloon program.

The CubeSat program is run collectively by Stanford University's Space Systems Development Laboratory (SSDL) and California Polytechnic State University. It's goal is to provide a framework for the design, construction and launch of picosatellites by students.

The Palantir Technology Demonstration, referred to as the "Tech Demo," is an instrument that was to be mounted as a payload on Steve Fossett's balloon. Using a combination of solar cells and batteries for power, this payload would collect measurements of temperature, pressure, altitude, humidity, and location data as well as take pictures of Fossett's flight as he attempts to travel around the world. This data is collected on solid state flash memory within the payload and would periodically be sent to a ground station via a network of amateur radio and satellite network. Unfortunately, in the days leading to the actual deployment of this payload, the power system failed, preventing its deployment.

Two websites with more detailed information are:

http://www.aria.cec.wustl.edu/
http://userfs.cec.wustl.edu/~palantir/

System Design (If this is a component or subsystem of a larger system)

2.2 System (Subsystem) Overview

DESCRIPTION OF NEW SYSTEM (OR SUBSYSTEM)

Palantir-B is the next generation Palantir rebuilding on the Palantir-Technology Demonstrator. The latter was designed in Spring2001 as an information gathering payload to be carried aboard Steve Fossett's Round-The-World balloon flight. The Palantir Technology Demonstrator or Palantir-TD payload would gather atmospheric and relative data such as temperature, pressure, altitude, humidity and location. Palantir-B will go a step further and allow the following data to be collected:

ATMOSPHERIC:

- external temperature, pressure, wind speed, wind direction

GPS:

- latitude, longitude, altitude

VISUAL:

- 360 degree virtual and motion detected images run on an event schedule

OTHER:

- sound and other sensor detected data such as sound from microphone

While Palantir–TD's software was written in C++, Palantir–B will use JAVA as its programming language in order to avoid fatal memory management problems and be ready to run on Linux (and possibly be made ready to run on a StrongARM processor).

Palantir-B will be designed to actively gather data based on an event-based schedule residing in an internal table. The data from the sensors located at various remote sites will be transmitted to the one or more Palantir Microsatellites, which in turn will transmit it to Palantir's Ground Communication Station. Palantir-B's Ground Processing Center will then collect this data from the Communication Station.

The main focus of Palantir-B is on creating robust software for the Ground Processing Center for the processing of data collected from the various sensors monitoring the atmospheric, GPS, visual and other controls. The processed data or information will be relayed back to the Palantir Communication Station, which will then make it available for viewing over the Internet allowing end users such as researchers and K-12 students to have a virtual experience of the remote Palantir sites.

How it will fit into the current domain?

Palantir-B is part of the bigger project known as Project Aria which is a hands on space engineering/science program allowing students to analyze, design, manufacture, launch and operate various space related projects. Palantir-B is one of the projects underway with remote monitoring as its primary operation. The final working implementation of Palantir-B would allow it to be used in other projects such as Palantir-TD where it would be able to replace the basic stamp computer software used in driving the sensors and accumulating and processing the data collected.

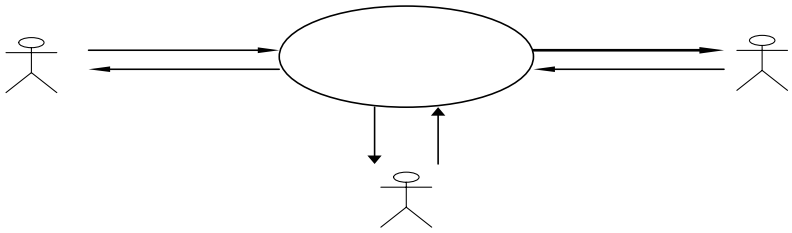
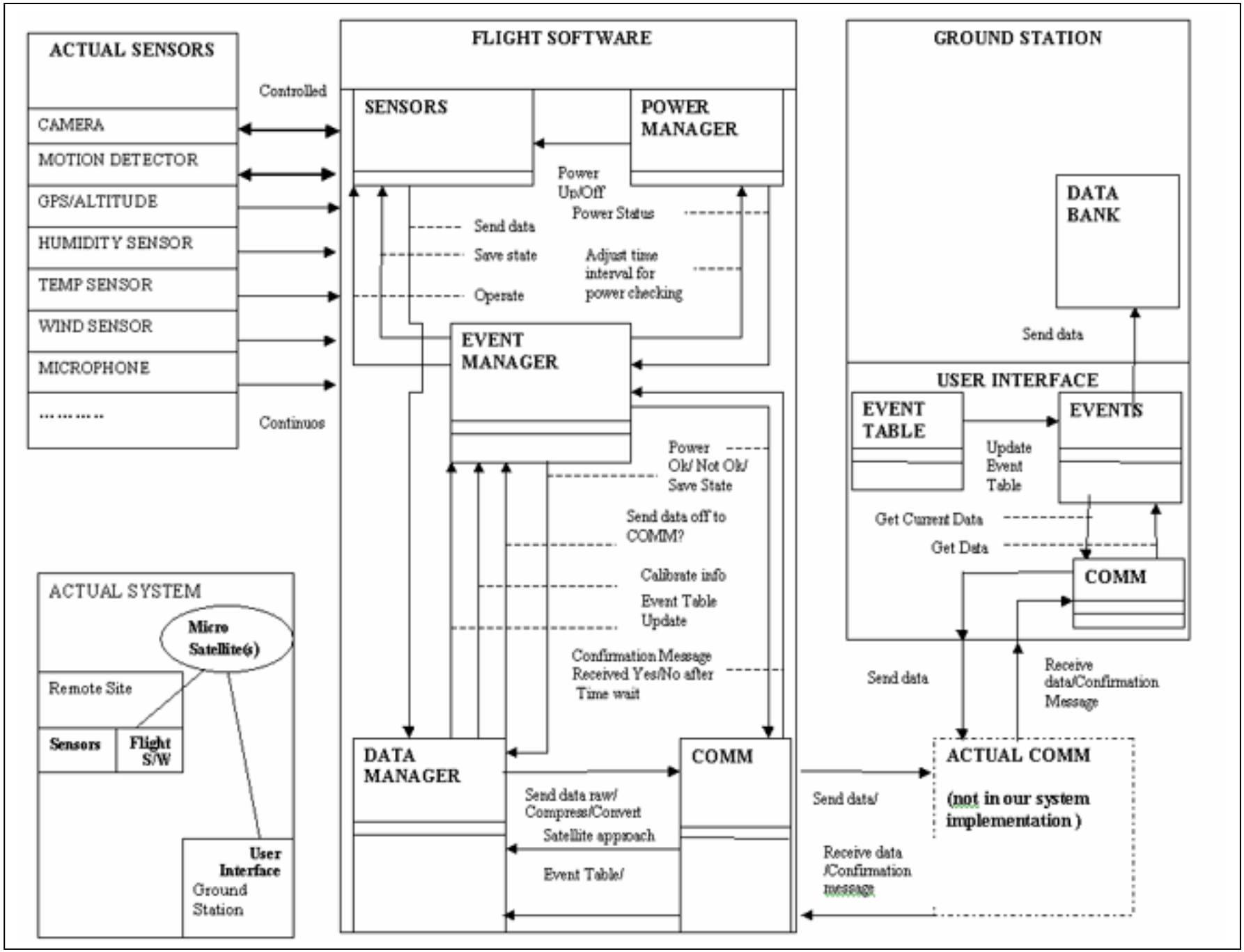


Figure 1 - System Context Diagram


ACTUAL SENSORS
CAMERA
MOTION DETECTOR
GPS/ALTITUDE
HUMIDITY SENSOR
TEMP SENSOR
WIND SENSOR
MICROPHONE
.....
Controlled
Continuos
FLIGHT SOFTWARE
SENSORS
POWER MANAGER
Power Up/Off
Power Status
Send data
Save state
Operate
Adjust time interval for power checking
EVENT MANAGER
Power Ok/ Not Ok/ Save State
Send data off to COMM?
Calibrate info
Event Table Update
Confirmation Message Received Yes/No after Time wait
DATA MANAGER
COMM
Send data raw/ Compress/Convert
Satellite approach
Event Table/
GROUND STATION
DATA BANK
Send data
USER INTERFACE
EVENT TABLE
EVENTS
Update Event Table
Get Current Data
Get Data
COMM
Send data
Receive data/Confirmation Message
Send data/
Receive data /Confirmation message
ACTUAL COMM
(not in our system implementation)
ACTUAL SYSTEM
Micro Satellite(s)
Remote Site
Sensors
Flight S/W
User Interface
Ground Station

3 REQUIREMENTS

3.1 Functional Requirements

A) Shall support sensors for:
   1) GPS.
   2) Forward-looking camera, vertical camera with a mirror giving 360-degree image.
   3) External temperature, pressure, humidity, wind speed, wind direction.
   4) Microphone.
   5) Movement detection.
   6) Shall provide easy adaptation for additional sensors.

B) Provide power management.
   1) Monitor power availability.
   2) Shut off or suspend operations if low power is detected.

C) Function according to an event schedule:
   1) Receive event schedule from ground station.
   2) Prioritize schedules.
   3) Override schedules during exceptional situations.
   4) Use schedule as a set of guidelines, not as explicit timeline.

D) Communicate with a ground station:
   1) Comm system that uses TCP/IP over Ethernet.
   2) Will simulate data transmission delays and low data rates

3.2 Nonfunctional Requirements

A) Performance:
   1) Distinguish between relevant and irrelevant data.
   2) Provide text interface for users.

B) Security:
   1) Moderate security required, no need for completely secure data transmission.

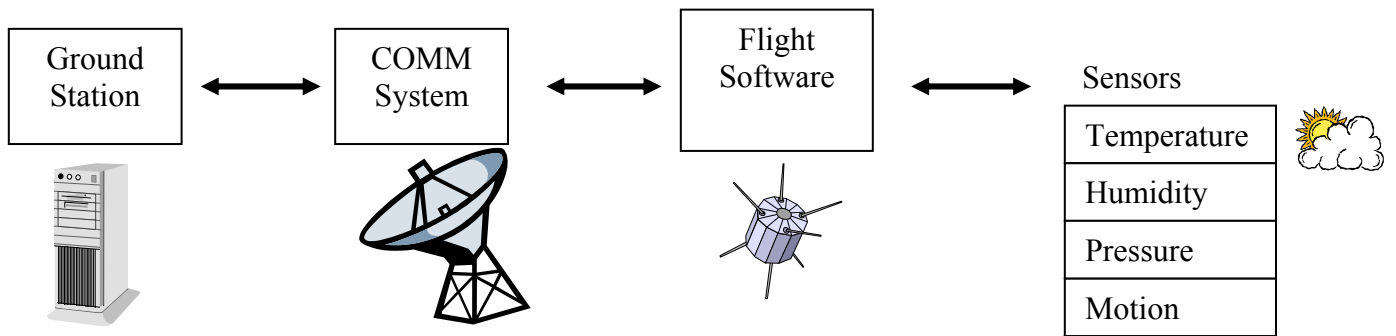
C) Modifiability:
   1) Support new sensors.
   2) Support new event schedule formats, templates.
   3) Support new communications systems.
   4) Support general flight software replacements.

D) Error handling:
   1) Recognition of error types.
   2) Ability to provide event logs, error logs.
   3) Provide troubleshooting support.

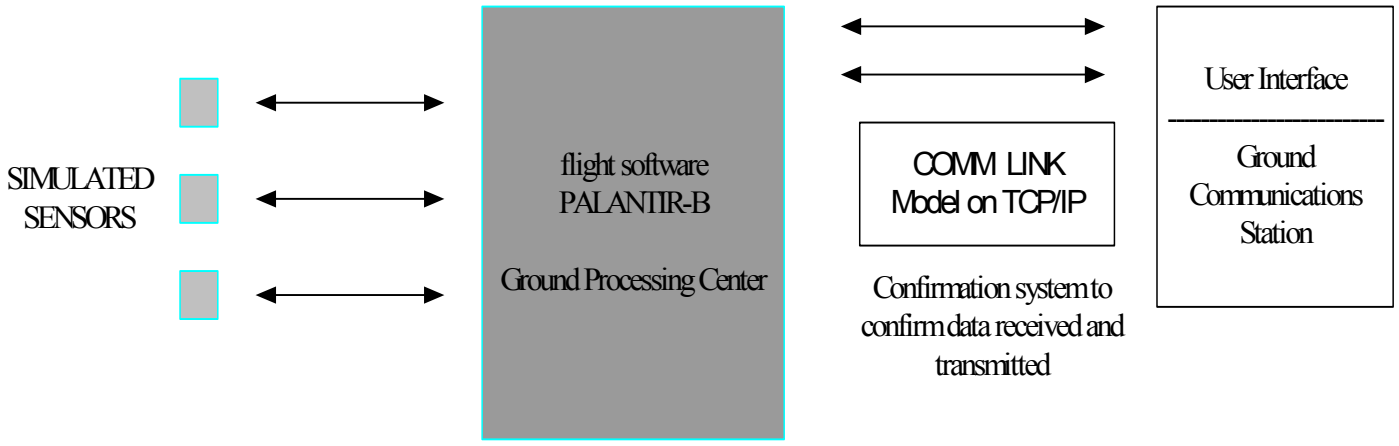
E) Hardware Restrictions:
   1) Should run on a StrongARM processor.

3.3 Pseudo Requirements

A) Platforms:
   1) Linux OS with java support.

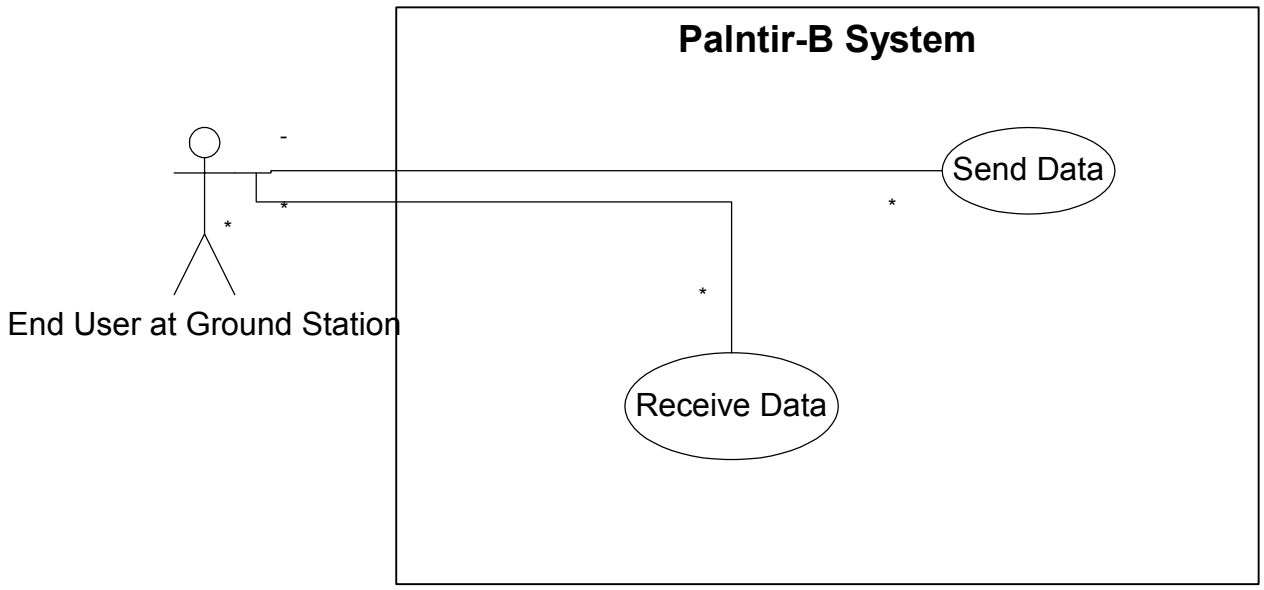


Overall System view



PALANTIR-B SYSTEM CONTEXT DIAGRAM

3.4 Functional analysis



Overall Use Case Diagram

3.4.1 Actors

1. **Actor:** Ground Station

   Purpose : To allow the end user at the Ground Station to interact with our system

   Relations: From the Comm System's perspective the Ground Station uploads and downloads data. From the Flight software's perspective the Ground Station send commands to it as an event table, as a calibration command or a reprogramming command.

2. **Actor:** Flight Software

   Purpose: To allow the collection od data from the sensors and relay it back to the users at the Ground Station

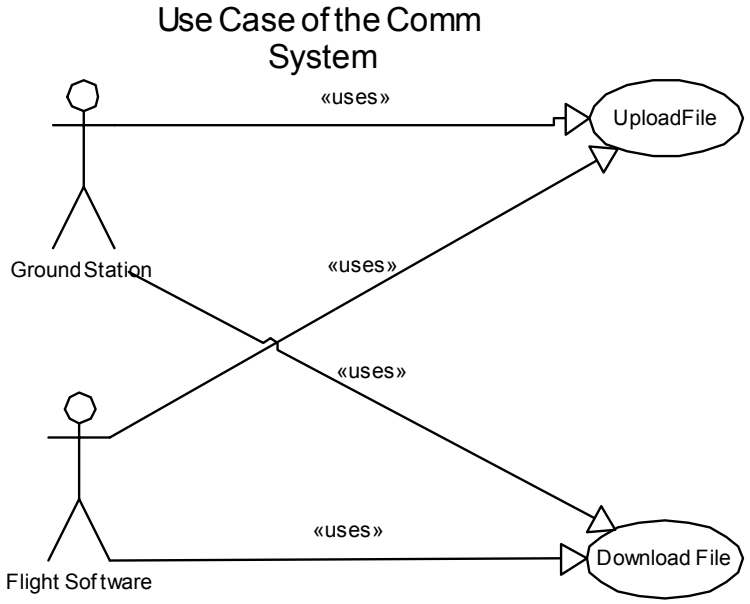
   Relations: From the Comm System's perspective the Flight Software uploads and downloads data. From the Ground Station's perspective the Flight Software sends in data which is saved locally.

3.4.2 Use-Case X

1. **Use-Case Name:** Comm System

   Participating Actors: Ground Station, Flight Software

   Entry Condition: When data is ready to be transmitted between Ground Station and Flight Software



Flow of Events:

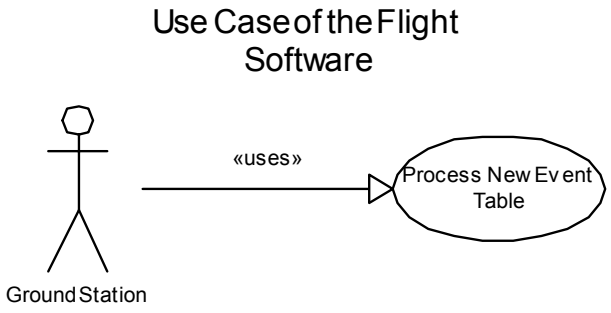
Exit Condition: When data has been sent to the Comm System

2. **Use-Case Name:** Flight Software

   Participating Actors: Ground Station

   Entry Condition: When the user at the ground station send data to the flight software

   Flow of Events:

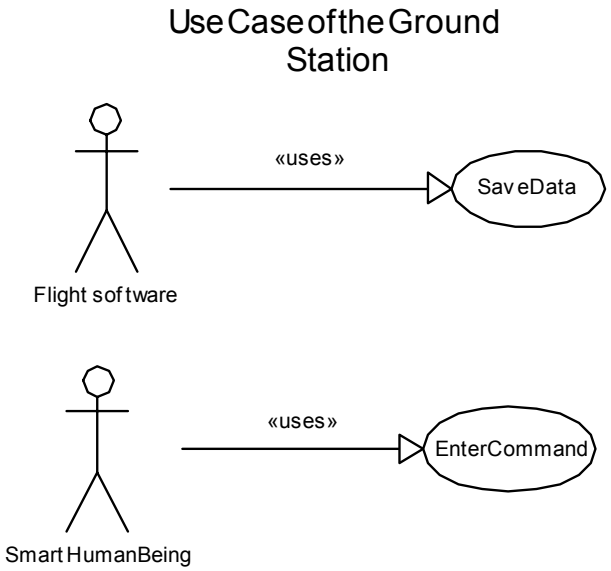


   Exit Condition: When the data has been sent by the flight station

3. **Use-Case Name:** Ground Station

   Participating Actors: Flight Software, Smart Human Being

   Entry Condition: When there is data to be sent to the flight software



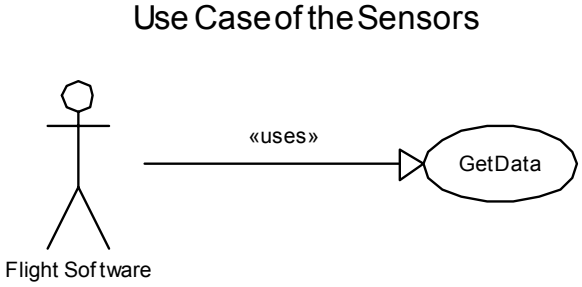
Flow of Events:

Exit Condition: When the flight software has sent back data and it has been saved on the ground station. When the ground station is ready to send the parsed input command from the Smart Human Being.

4. **Use-Case Name:** Sensors

   Participating Actors: Flight Software

   Entry Condition: When the flight software is ready to get data from the sensor based on a scheduled event from the event table.



Flow of Events:

Exit Condition: When the flight software has received data

3.4.2.1 Scenerio X.1

4 USER INTERFACE

The user interface is designed to be intuitive and easily expanded. To this end, some functions defined here may not actually be implemented in this portion of the Palantir project lifecycle. The actual interface is a simple text prompt which the ground user can use to enter commands one at a time. Because the user interface will be replaced in future versions of this product, it does not need to be particularly error-tolerant. All entered commands will be grouped and sent to the first available satelite, and will be executed by the Palantir probe in the order the commands are entered. Note: command ordering assumes that all commands are sent together and are recevied in the same transmission burst. Because commands may be received (and thus executed) out of order, it is recommend that mission-critical commands such as changing out flight software be executed only when successful receipt of the target data has been confirmed.

All commands have been grouped under three headings: get, send, and execute. All syntax is in the form `(GET/SEND) command [options].` Incoming data from the Palantir will be dumped into appropriate files on the groundstation machine, and will not be displayed on the text interface screen. Any instructions to be executed by the Flight Software should be included in the event table.

Get Information

`GET CHECKSUM filename`

This command will cause the Palantir-B to send back the sum of the bytes in the requested file. This command is userful to verify correct delivery of mission-critical data packets, such as those holding flight software updates.

`GET DATALIST [directory] [start date [ end date]]`

By default, this command returns a listing of all the datafiles in the system, including software, event schedules, and captured data, and shows the file names, dates, free space, and any other relevent information. This option can be used to check for new files that may have been lost in transmission, and to perform routine maintenance checks.

-The optional directory option allows the listing to be limited to a particular directory.

-The optional start date and end date options allow the returned listing to be limited to data acquired in the specified date range.

`GET FILE filename`

This requests that the Palantir send back a particular file, which may contain an image, sound, log, or other data.

Send Information

`SEND FILE filename [destination name]`

This option is provided to allow ground teams to update software remotely. The file specified by filename will be sent, and will be placed into the name/location specified by destination name. If no destination name is provided, the file will retain its original filename and will be placed into a default directory onboard the Palantir. If the file already exists on the Palantir, it will be overwritten by the incoming file. Note: users are strongly cautioned against overwriting exiting

files. For software updates, a better strategy is to send the file and request a checksum, and only then swap out the old file with the new using the SWAP command.

`SEND SCHEDULE (APPEND/OVERWRITE) filename`

This command allows the ground station to update the flight software's schedule. Sending with the APPEND option will simply add events, while choosing OVERWRITE will replace the Palantir's onboard schedule with a new, provided schedule. All schedule updates are performed by sending a text file containing schedule events.

Use Case Example A:

This is a typical set of commands for a ground station controller. In this example, the user is adding some events to the Palantir schedule, and is manually retrieving an image.

```
>SEND SCHEDULE APPEND sched05.txt
>GET FILE /camera/img00128.jpg
```

Use Case Example B:

For this example, the ground controller requests a file list. Seeing that there is a junk datafile using up space, the controller deletes it. In addition, he re-enables the camera which had previously been shut down to conserve power.

```
>GET DATALIST
// -- Break of several days to get data -- //
```

Use Case Example C:

This example is a conceptualization of how flight software would be updated, should that need arise.

```
>SEND FILE flightSoft flightSoftNew
>GET CHECKSUM flightSoftNew
// -- Break of several days to get data -- //

>GET CHECKSUM flightSoft
// -- Break of several days to get data -- //
>DELETE flightSoftNew
```

5 DESIGN APPROACH

The design of our system is driven by the event table and other data such as calibration information, which is sent by user at the ground station. Because this system is designed to fulfill an extremely narrrow, specific purpose, it does not have as many external influencing actors as other similar projects. Rather, most of the "actors" and communications take place between components of the system itself.

The key requirements that will drive our design include : modifiability and "event-responsive." Modifability meaning that the system can be modified easily in order to satisfy new demands, such as addition of sensors. "Event-responsive" means that the system does not strictly follow the commands given by the user and is equipped to deal with special events.

Our design will basically be object-oriented to allow modifability and abstract away the complexity of the main system and simplify the interaction with the external hardware. Our design will use threads, which is supported in Java, to allow parallel running of multiple events.

6 DEVELOPMENT PLANNING

This section is used to describe the general development approach. On larger systems, this would be in a separate Software Project Management Plan of Software Development Plan.

This sections should contain the following:

- Master Development Plan – i.e. versions to be development. For each version, identify which requirements and/or use-cases are to be development.
- Development Standards – Idenfity coding, design, documentation, etc. standards that the development team will adhere too.
- Configuration Management Plan – Describe the configuration management approach.
- Development Environment – Describe the development environment to be used. Include languages, tools, facilities, etc.

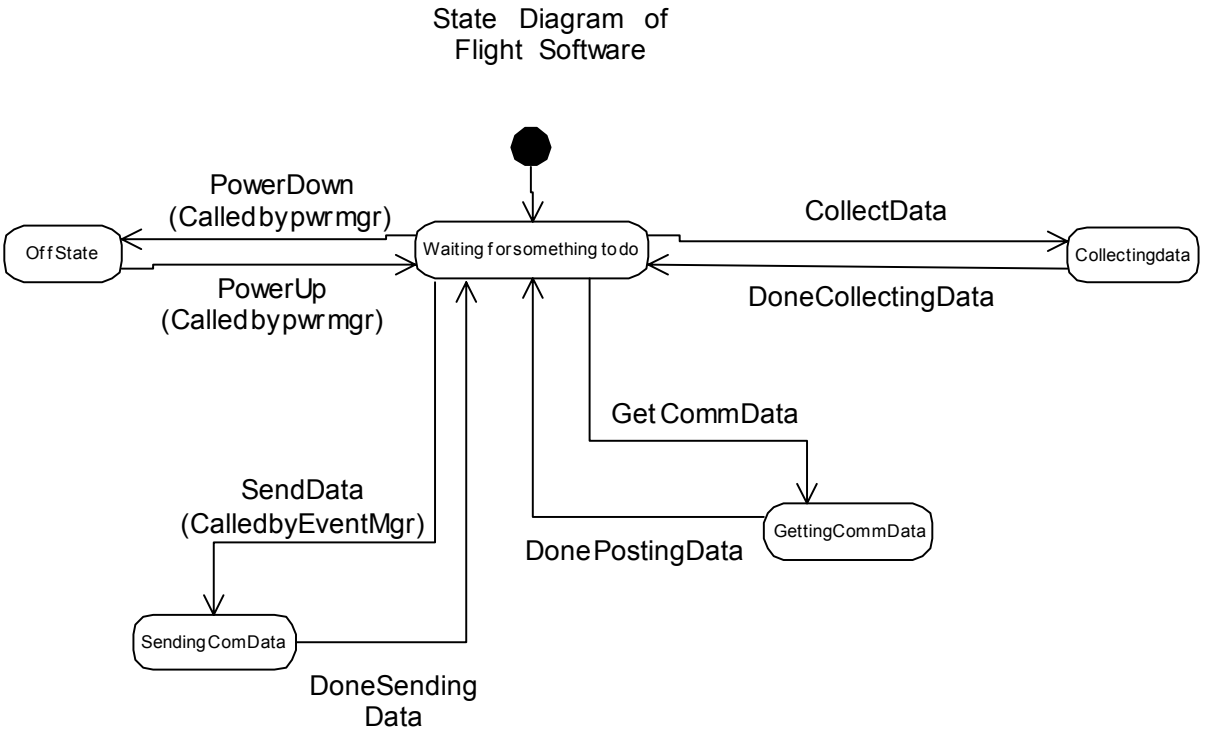
7 FUTURE CAPABILITIES

List any future capabilities which, while not required in the current system, should be considered during design.

8 ANALYSIS/DESIGN

8.1 Analysis/Design Model

8.1.1 Component Descriptions



8.1.1.1 Sensors Class

- **Description and Purpose**

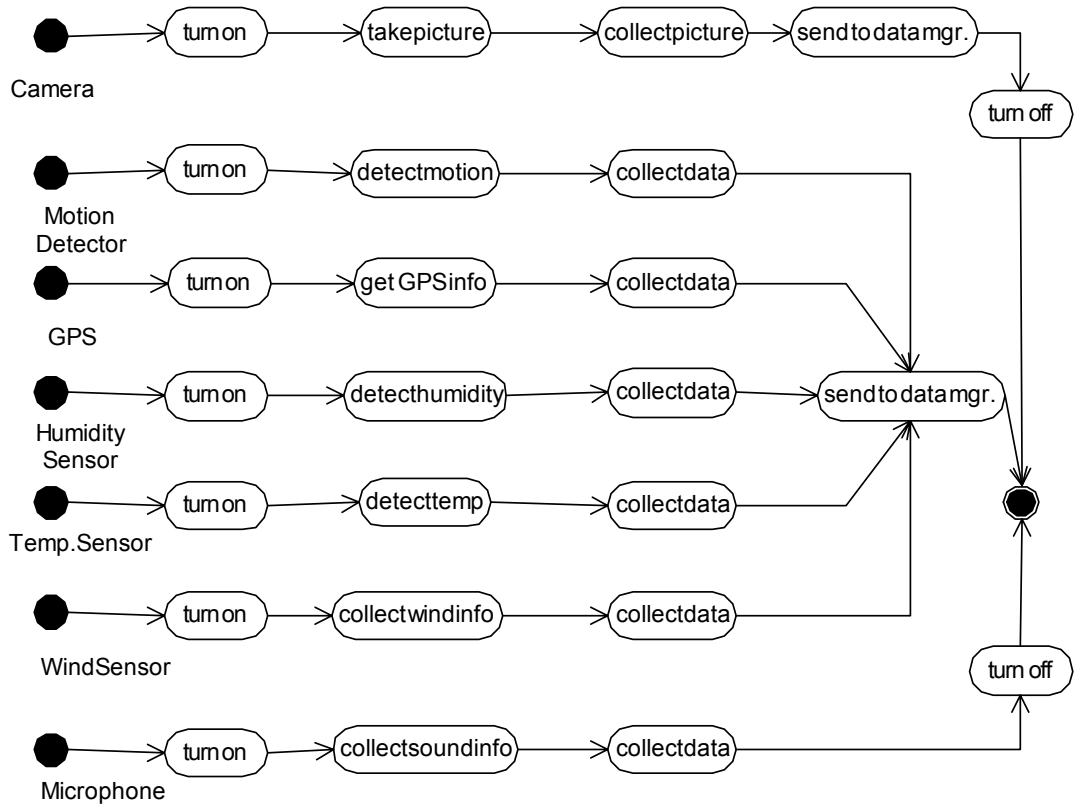
  The Sensors Class will implement a class supporting the use of pre-determined sensors, collection of data from the sensors and the change of calibration of the sesnors when possible (as in the case of Camera). The sensor class will try to minimize what the user of this component should know by hiding the details of sensors mangement for each specific sensor and instead making available simple and recognizable functions as an interface to the general Sensor class.

  - Dynamic
    - The Sensors class will be a dynamic component running its own thread. This would not only allow any new sensors to be added in the future without having a ripple effect on the other components dependent on the Sensors Class but also have several sensors running at the same time

  - Permanent
    - The Sensors class will be a permanent object that exists as long as the program is running

- The Sensors class will be a top-level public class so as to provide easy access by other components.

- **Overall Component States**
  Each sensor thread only has two states, can be either running or dead.
  Overall Component Processing (shown via activity, sequence or collaboration diagram) –



ActivityDiagram

For primary sensor operations

Hardware Component

Wind Sensor
Pressure Sensor
Temperature Sensor
Humidity Sensor
(360deg)Camera
GPS sensor
Microphone

Wind Sensor

Since the Wind sensor will be on all the time and its operating state will not be controlled by the Sensor Class the only point of concern is that of the kind of data being passed.
Get Wind Speed

Pre-Conditions
Wind Sensor: On
Post-Conditions
Wind Sensor: Targeted
I/O
Speed: km/hr/miles/hr

Pressure Sensor
Since the Pressure sensor will be on all of the time and its operating state will not be controlled by the Sensor Class the only point of concern is that of the knid of data being passed.
Get Pressure

Pre-Conditions
Pressure Sensor: On
Post-Conditions
Pressure Sensor: Targeted
I/O
Pressure: (Barometer Reading)inches

Temperature Sensor
Since the Temperature sensor will be on all the time and its operating state will not be controlled by the Sensor Class the only point of concern is that of the knid of data being passed.
Get Temperature

Pre-Conditions
Temperature Sensor: On
Post-Conditions
Temperature Sensor: Targeted
I/O
Temperature: Celsius/Farenheit/Kelvin

Humidity Sensor

Since the Humidity sensor will be on all the time and its operating state will not be controlled by the Sensor Class the only point of concern is that of the knid of data being passed.

Get Humidity

Pre-Conditions
Humidity Sensor: On
Post-Conditions
Humidity Sensor: Targeted
I/O
Humidity: degrees

(360deg)Camera

Since the Camera is one of the sensor devices which will be on all the time but it will be callibrated by the Sensors Class which will be receiving the caliibration instructions from the Ground Station our piint of concern is not only the visual dat being passed but also the ability to monitor the state of the Camera and callibrate it when the power is sufficient and also to be able to shut it off in certain conditions such as night conditons where the visual images being taken and sent by the camera might not be worth taking.

Check Image

Will help to test for night conditions and similar conditions(storm etc)
Pre-Conditions
Camera Sensor: On
Post-Conditions
Camera Sensor: Targeted
I/O
Direction : camera_direction;
Angle: viewpoint angle
Image: jpeg, gif

Power up/off camera

Pre-Conditions
Camera Sensor: Off/On
Post-Conditions
Camera Sensor: Targeted
I/O
Power Up/Off Confirmation: Analog/Digital signal

Power Check

Pre-Conditions
Camera Sensor: Off/On
Post-Conditions
Camera Sensor: Targeted
I/O
Power Sufficient Confirmation: Analog/Digital signal

GPS sensor

Since the GPS will be on all the time and its operating state will not be controlled by the Sensor Class the only point of concern is that of the knid of data being passed.
Get GPS

Pre-Conditions
GPS Sensor: Off/On
Post-Conditions
GPS Sensor: Targeted
I/O
Location: Latitude/Longitude degrees
Altitude: km/miles

Microphone

Since the Microphone sensor will be on all the time and its operating state will not be controlled by the Sensor Class the only point of concern is that of the kind of data being passed.

Pre-Conditions
Microphone Sensor: On
Post-Conditions
Microphone Sensor: Targeted
I/O
Sound: Radio waves

Motion Detector

The Motion Detector will be turned off / on accordingly based on the environment. For example, in night condition it can be turned off by the flight software.

Pre-Conditions
Motion Detector: On
Post-Conditions
Motion Detector: Targeted
I/O
Signals

8.1.1.1.1 Internal Design

The internal design for the Sensors Class may be a little more complicated than the above overall view. The Senors class will be interacting with other possible componenets such as a Power Management Class and an Events Class both of which are essential for mangaging the data from the sensors as the Power Management Class will make sure that there is sufficient power to manage the sensors and the Events class will schedule the collection/transmission of data. This interaction will be hidden from the user.

8.1.1.2 Power Manager Class

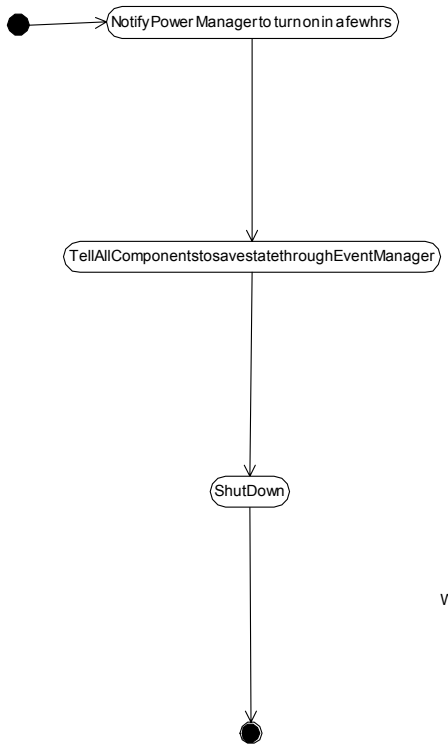
- **Description and Purpose** – The power manager class fundamentally controls the operation of the system. If there is not enough power for the system to maintain it's operation, the power manager will force the system to either use less power or shut down the system temporarily to allow the batteries to recharge. The battery used by the system is smart, meaning it provides this class with remaining power.

  The event manager class notifies the power managerclass when there is no operation scheduled for an extended period of time. During this time, it may be better just to shut down the system for a while to save battery power. For example, if the event manager determines that the current time is 12:00PM, and the next event scheduled at 4:00PM, then it will notify this class. It will then be up to this class to decide whether to shut down the system to conserve battery power.

  Every time the flight software reboots, the power manager is consulted to ensure that there exists enough battery power to run the system reliably. If there isn't enough battery power, then the system is shut down.

  - Dynamic, Static, or Evolving:
    - This class is a dynamic class. It continously monitors the state of the battery subsystem and changes state when the battery level reaches a certain critical level.
  - Temporary, Permanent, or Persistent:
    - This class is permanent since it must monitor the power throughout the lifetime of the system.
  - This is a private class within our application.

- **Overall Component States**
  Since the power manager class is permanent, it has only one state, "running."

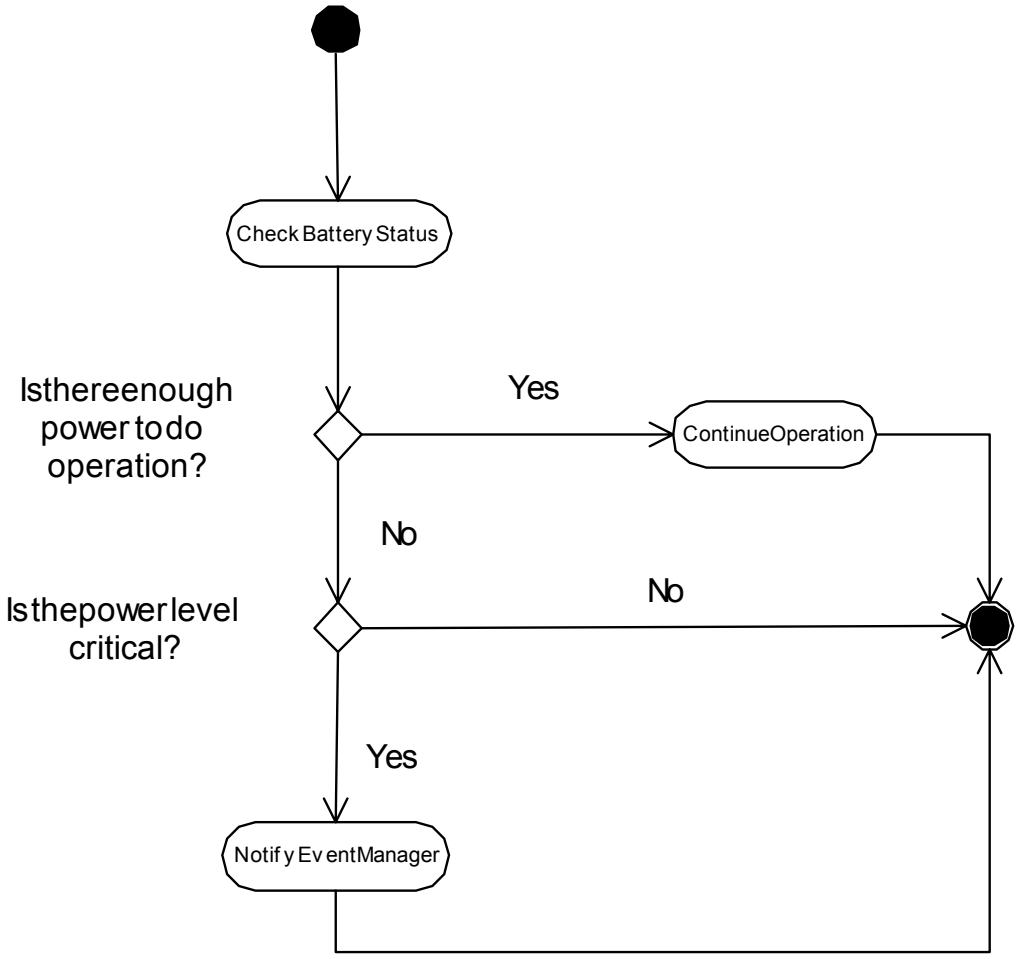


ActivityDiagram

WhenPowerManagerdetects
criticallylowpower

- **Overall Component Processing**

Activity diagram for the power manager to check battery status



- **Hardware Component** – runs on the same hardware as the flight software
- **List of all component procedure, methods or entry points:**

CanRunEvent

Determines if there is enough power to run the event.

Pre-Conditions

The battery exists, the battery is smart

Post-Conditions

We know if we can run the event given the power availability

I/O

In: the event we want to run, out: yay or nay

CanRunApplication

Determines if there is enough power to run the application

Pre-Conditions

The battery exists, the battery is smart

Post-Conditions

We know if we have enough power to run the application

I/O

In: none, out: yay or nay

8.1.1.2.1 Internal Design

The power manager contains a thread that continuously monitors the state of the smart battery. If the state of the battery falls below a certain critical value, it immediately shuts down the system. This scenario is extremely rare because other classes can pass events to pass this class to determine if there is enough power to run the event.

8.1.1.3 Event Manager Class

- **Description and Purpose**
  The event manager class takes in table of events from the ground station and schedule the events in a sequential fashion. It also has a table of internal events, separate from the event table supplied by the user at the ground station. If there is enough power to carry out the event, then the event manager will perform the event by activating the appropriate sensors.

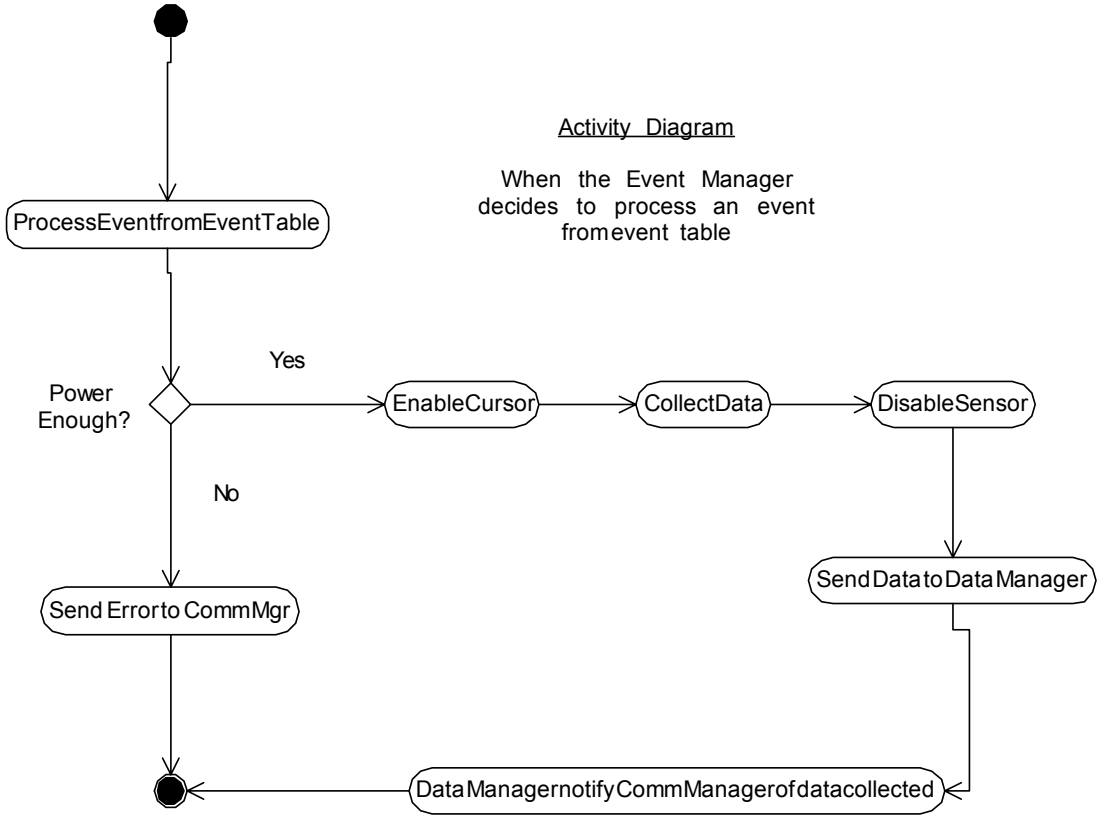
  If there is enough power to perform an event, but carrying out the event takes resources already consumed by another event, then the event with the highest priority takes precedence.

  If there is not enough power to run the event, the event manager will add an event into its internal event schedule (which takes the highest precedence), telling the ground station of the failure to complete the given event.

  - Dynamic, Static, or Evolving
    - This class is dynamic because it is running its own thread.
  - Temporary, Permanent, or Persistent
    - Permanent – must exist throughout the life of our application. It contains an event schedule which is persistent. This event schedule is a file stored on the flash memory module.
  - This class is a private class within the flight software class.

- **Overall Component States**
  Since the Event manager is a dynamic class runing several threads, it does not exhibit one state at any given point of time.

- **Overall Component Processing**



- **Hardware Component** – The event manager runs on the same processor as the main application.
- **List of all component procedure, methods or entry points:**

  New Event List

  Replace the existing event table with a new one.

  Pre-Conditions

  None

  Post-Conditions

  The old event table is replaced with the new one.

  I/O

  Table of events: a list of event.

8.1.1.3.1 Internal Design

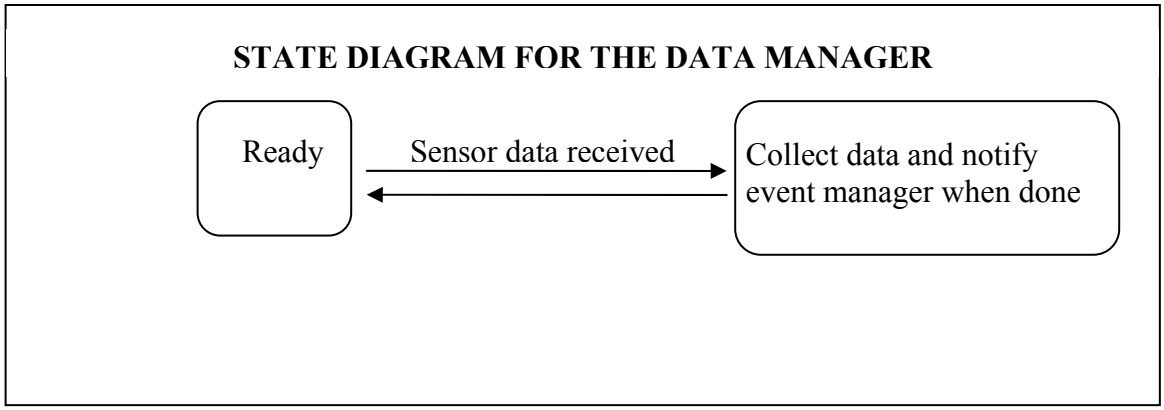
The event manager is a class that contains within it a schedule, which is a priority queue ordered by smallest starting time. The event manager registers itself with the system clock and is notified every time-interval change. This time interval has yet to be determined, but it basically is the finest grain of time resolution that the end user desires. Every time this class is notified of a time change, it queries the priority queue looking for events that need to be scheduled. If there are internal events that need to be scheduled, the class then checks to make sure there are enough resources available to run the events.

If there are, the events are executed. If there isn't, the event may be run depending on whether it has a higher priority than the event currently taking up the resource in question. If the event cannot be run, then an event is added instructing this class to send a failure message to the comm system.

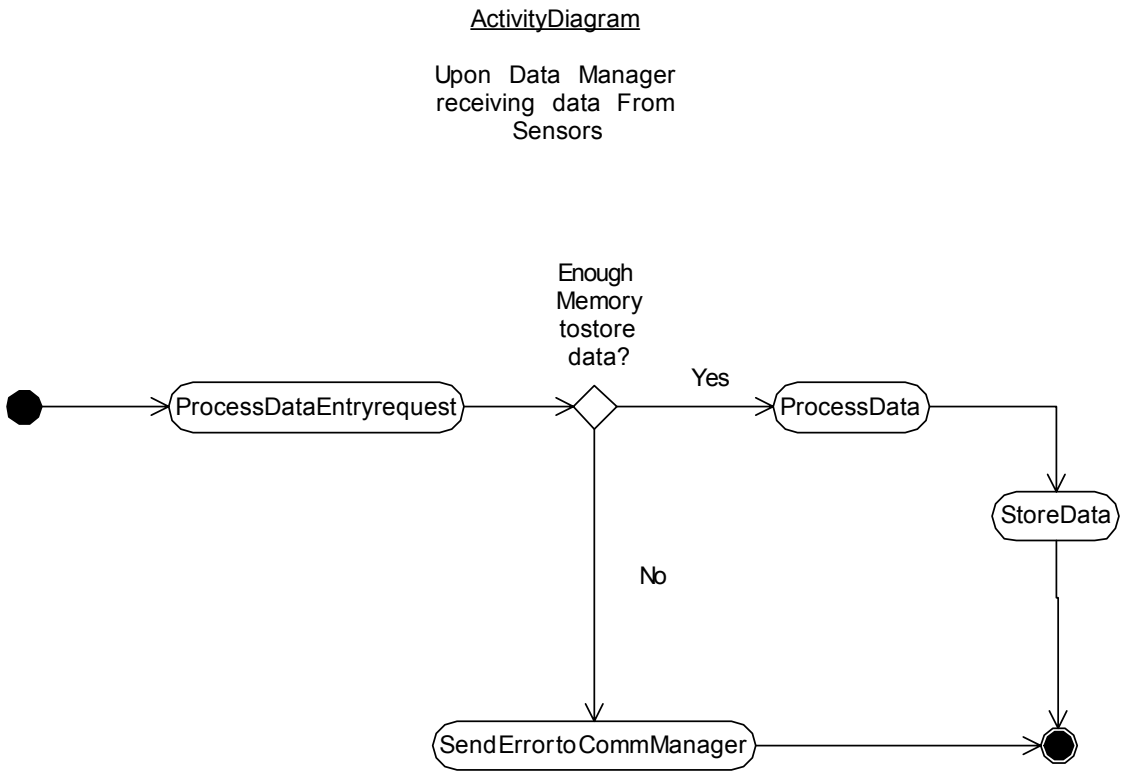
8.1.1.4 Data Manager Class

- **Description and Purpose** – The Data Manager class is responsible for collecting data from the sensors. It is also responsible for telling the Event Manager when it is ready to send data.

  When the sensors are activated through the Event Manager and the Data Manager records the data collected by the sensors, the Data Manager informs the event manager that the data just collected needs to be sent back to the Ground Station. The Event Manager determines whether there is enough power and tells the data manager that it is ok to send the data to the comm manager.
  - Dynamic, Static, or Evolving
    Since this class only performs actions when the event class tells it to, this class is considered static.
  - Temporary, Permanent, or Persistent
    This is a permanent class.
  - This class is also a private class held within the flight software
- **Overall Component States**



- **Overall Component Processing**



- **Hardware Component** – runs on the same hardware resources as the main process.
- **List of all component procedure, methods or entry points:**

  RecordData

  Saves the data gathered by a sensor into persistent storage

  Pre-Conditions

  none

  Post-Conditions

  The data is stored, and the event manager is aware of the data

  I/O

  In: the data to be saved

  Out: an event that tells the flight software to send the data back to the ground station

8.1.1.4.1 Internal Design

The data manager is a class that has listeners registered with all of the sensors. Whenever the sensors send the data manager data, the data manager uses the file manager within it to store the data and then notifies the event manager that it has data that is ready to be sent to the comm system.

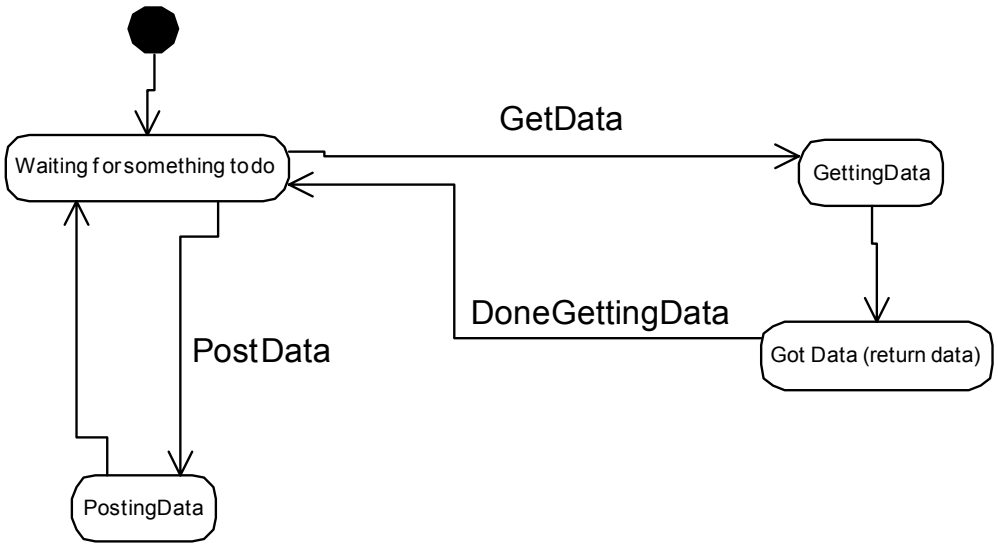
8.1.1.5 Comm System Class

Description and Purpose – The Comm Class will manage the communications between the Ground Station and the Flight Software Package and will be in constant access by the Ground Station and the Event Manager classes in the package. It will be able to send and receive data / messages from the Ground Station using a confirmation mechanism for managing messages and requests to minimize the problems associated with time lag .

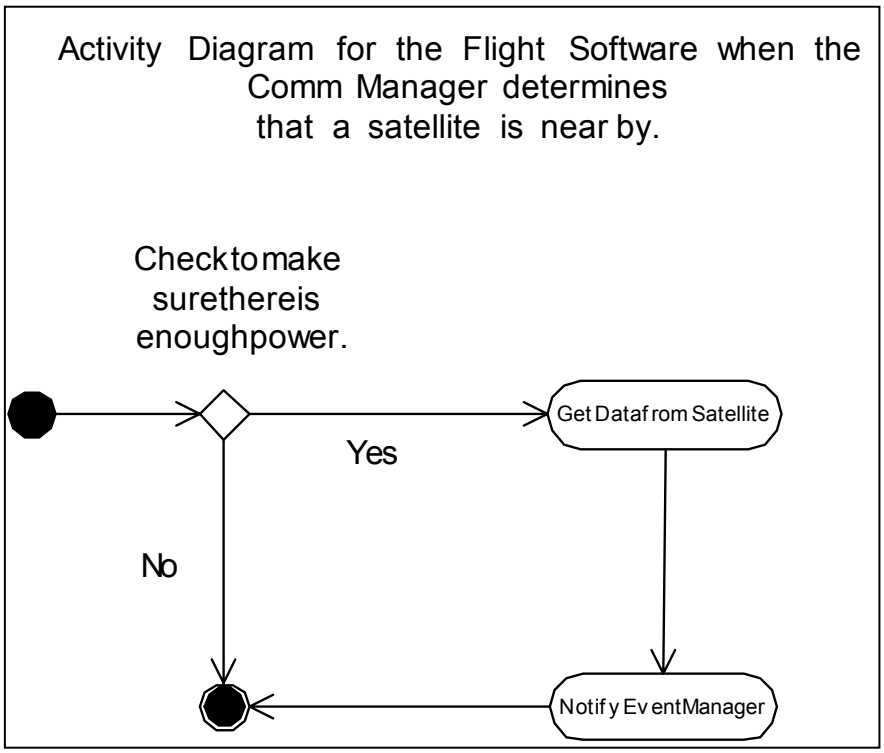
- characteristics of the component including:
  - Static.
    The Comm Class will be a static component so that it cannot change it's own state without another (main) component calling it. This would help to avoid any internal changes take place in the comm system without first notifying the other classes and also help in making sure that all the components accessing the Comm Class will have the same state information about the Comm Class.
  - The Comm Class will be a permanent object.
  - The Comm Class will be a public class allowing easy access to it by the other main classes which might be functioning independently .

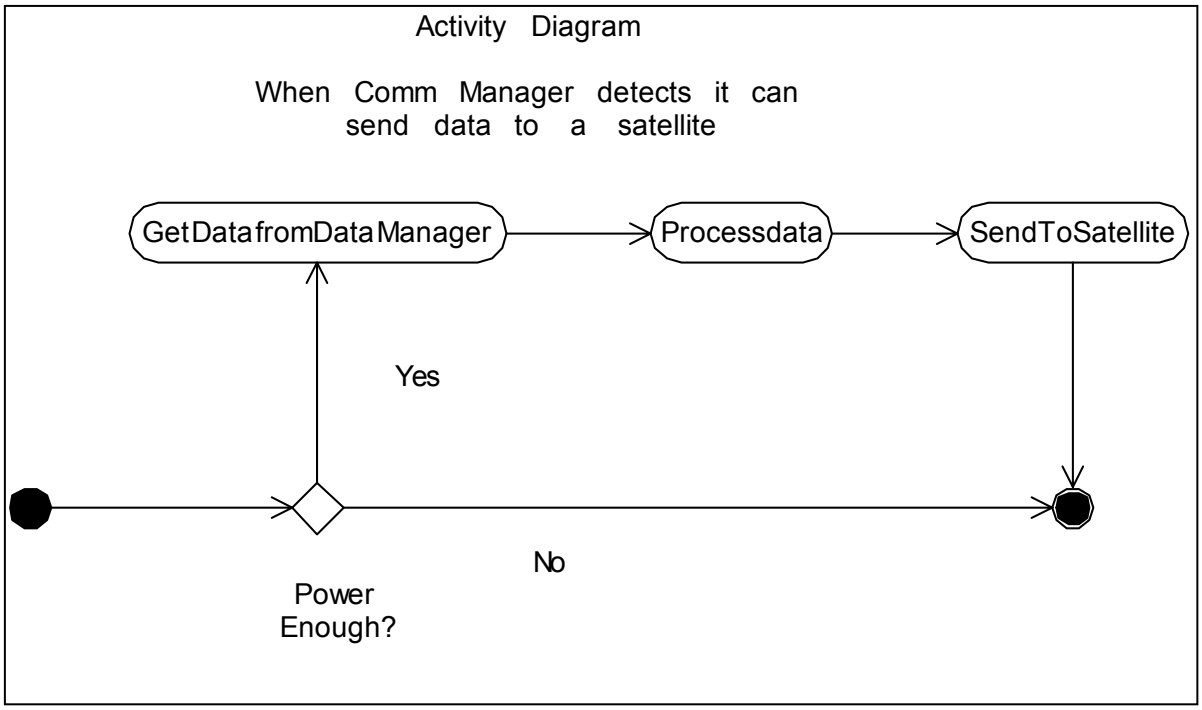
- **Overall Component States**

State Diagram of Comm System



- **Overall Component Processing**





- **List of all component procedure, methods or entry points:**

Send/Receive/Erase/Check(for unsent) data packet
Will allow for the sending/receiving/erasing/checking of data packets currently available to the Comm Class

Pre-Conditions
Data Object State: Available(i.e. communcation is up and running)
Post-Conditions
Data Object State: Targeted
I/O
Data State: State variable corresponding to action

Check Event Schedule
Will allow for the sending/receiving/erasing/checking of data packets
According to the Event schedule currently available to the Comm Class
Pre-Conditions
Event Schedule State : Available(i.e. communcation is up and running)
Post-Conditions
Event Schedule State: Targeted
I/O
Event State: Data packets containing forthcoming event details

Power Check
Will allow Comm class to make sure power is sufficient to proceed with further actions
Pre-Conditions
Power Object State: On/Off
Post-Conditions
Data Object State: Targeted
I/O
Power State: Message indicating power state

8.1.1.5.1 Internal Design

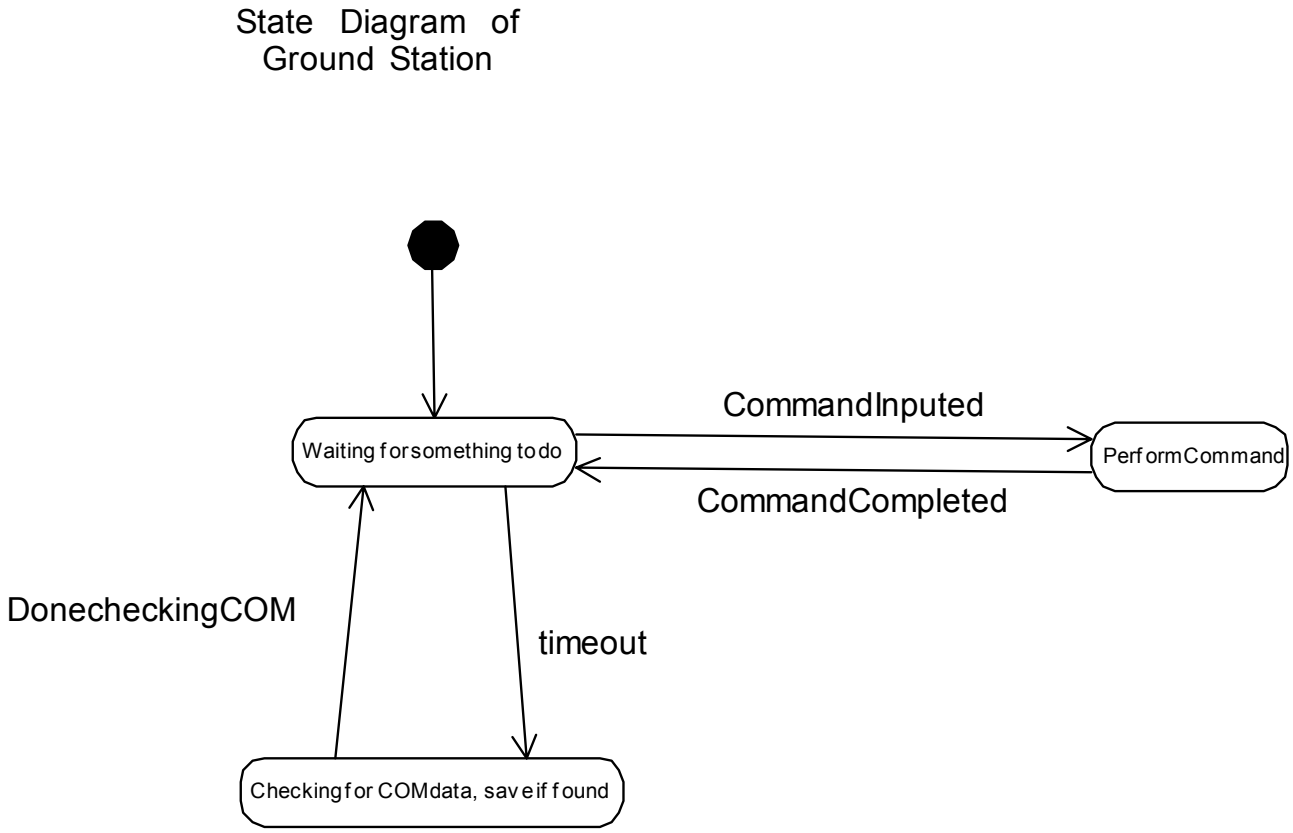
The Comm Class will have sufficient independence to manage data transmission once it is authorized by the other main components such a Power Management Class and an Event Management class but since it can change state only when called to do so by other components its internal design will not be too complicated so as to make the design creates state which are not discernable by other components. For e.g.: If the Comm Class were to change its data packet size internally without the other component knowing about it this might create problems of non-transparency.

8.1.1.6 Ground Station Class

- **Description and Purpose** – In this implementation of the program, the ground station is used to send event schedules to the flight software and to receive data from the flight software via the comm system.

- Dynamic, Static, or Evolving
  Static
- Temporary, Permanent, or Persistent
  Permanent – since we are in a distributed environment, this application must always be running on the user side of the application.
- This is a public class.

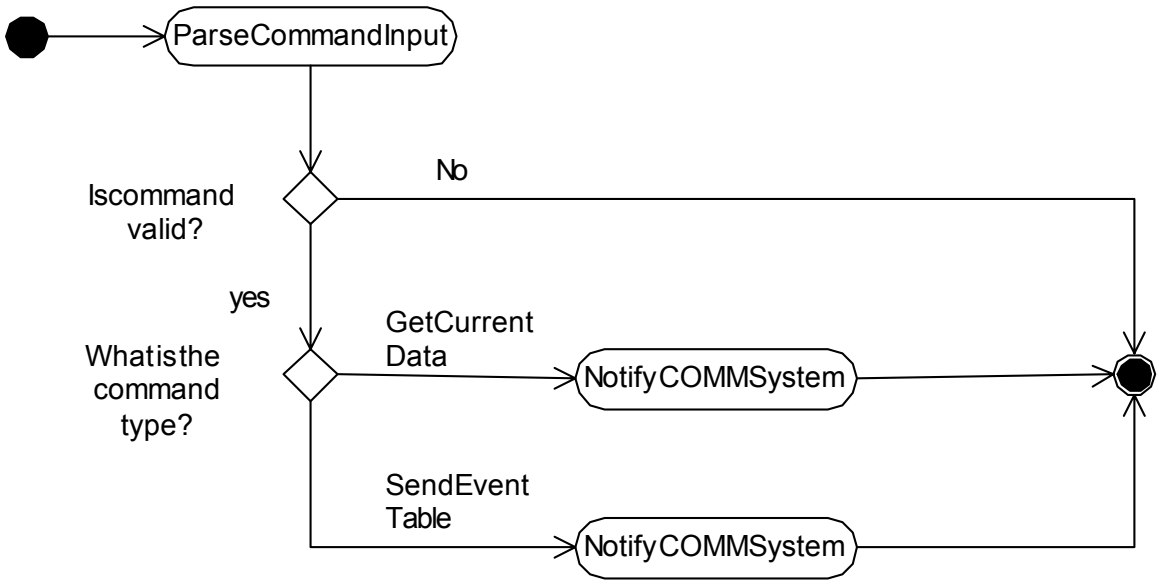
- **Overall Component States**



-

- **Overall Component Processing**

Activity Diagram for the ground station



- **Hardware Component** – Runs on the local user's system.
- **List of all component procedure, methods or entry points:**

AddEvent

Append an event to an event table.

Pre-Conditions

Event table exists.

Post-Conditions

Event table updated

I/O

Out: the event(s)

RemoveEvent

Remove an event from an event table before sending it off to the flight software

Pre-Conditions

Event table exists.

Post-Conditions

Event table updated

I/O

Out: the event(s) to be removed

Send Event table

Send the entire event table constructed using AddEvent and RemoveEvent

Pre-Conditions

Event Table ready to be sent

Post-Conditions

None

I/O

Out: Event table sent

GetCurrentData

Fetches the data collected by the flight software to us.

Pre-Conditions

The comm. Manager exists

Post-Conditions

The ground station has the data collected by the FS.

I/O

In: the data collected by the flight software

8.1.1.6.1 Internal Design

The ground station consists of a very simple GUI that allows users to input commands to the system. These commands are parsed by a command parser which determines what the user wants to do and passes the appropriate command to the comm center. The ground station has a comm which determines when the next satellite crosses over and checks for incoming data when one does.

8.1.2 External storage

In our system, we collected data by the flight software and we sent data back to ground station.

8.1.2.1 External Storage : ground station part

The ground station is assumed to exist for the operation of the flight software and the following features are assumed by the ground station's internal storage, which is external to the flight software.

Storage Method – Identify how the data is stored such as a file, special hardware device, or database.

Characteristic:

- Low_price harddisk allow big storage space and complex mangement of data
- Need user_friendly interface

Method:

- Data saved as files in different directory
- Can keep data long-period
- Can save data by other media
- Can use some DBMS to control data

- Hardware Used
  - Mainly be harddisk
  - Can use multiple media such as CD etc

- Data Format - Describe the format of data stored including data structures. For example, the data is stored in 32-Byte records made up of the following data elements or the data is stored Data_Packet format where Data_Packet is defined here or in the data dictionary.
- Analog data:
  - Data source: sensors which collects analoy data, such as humidity, pressure, motion sensors.
  - Through A/D convertion, those data will transferred to digitial data.
  - Data structure: float point (double)
- Digital data:
  - Data source: sensors which collects digital data, such as temperature or GPS etc;
  - Data structure: float point (double)
- Image data
  - Data source: CCD_camera
  - Data structure: .jpeg
- Audio data
  - Data source: microphone
  - Data structure: .ave
- Other – Include any other information relevant to understanding the interface and to designing software to communicate using the interface. Don't be afraid to include additional information.
  - Currently, the user interface will be very simple. It will be mainly in command line style to pass data to user and user manage directory directly by unix command.
  - Future function: GUI for user to manage and handle data. Multiple diagram and tools for use to "visiualization" data and image.
  -

8.1.3 External Interface Descriptions

8.1.3.1Ground Station Interface:

- Purpose of the Interface:
  - To communicate with the flight software through the comm system.

- Conditions under which it is used:
  - End user at ground station wants to ask the comm system if there is new data available from the flight software.
  - End user at ground station wants to create new event schedules
  - End user at ground station wants to send event schedules, calibration info or updates to the flight software through the comm system.

- Data items to be passed in and out:
  - The ground station will send and receive Data_Packets to and from the comm system.
  - Data_Packets obtained from the comm system will contain data that will be timed, dated, and classified as visual, text, audio, flight software error

messages, flight software status messages, etc., and will have a minumum and maximum total size.
- Data_Packets sent to the comm system will contain event schedule data, and software update data.

- Communications Method :
  - The ground station interface will communicate with a simulated comm system via ethernet using TCP/IP.

- Data Frequency:
  - Data requests by the ground station interface can be sent at:
    - Present time,
    - Any number of seconds, minutes, hours, or days in the future,
    - Or on an automated schedule (e.g. every minute, hour, day, etc.)

8.1.4 Data Dictionary

All identified data items should be described here. Each data description should include type, range, and any constraints.

Sensor / Camera Data:

Temperature – in Calvin. A float will be used. The value ranges from –200C to 200C, subject to change when we research on the specific environment in which palantirB will operate on.

Pressure,Humditiy,Windspeed - A float will be used for each of the measurement. It is subjec to change when more precise values are needed.

WindDirection - 3 integers representing the degree, minutes, and seconds of the wind direction from the North. The degree ranges from 0 to 360 / or / 0 to 2 pi radians.

GPS - 2 sets of 3 integers, one set representing the degree, minutes and seconds in the longitude and the other represents the degree, minutes, and seconds in the latitude.

Camera - images in jpeg format sliced up into a fixed number of sections.

Microphone - the audio data will be in 64-kbr MP3 compression format. There will only be a limited amount of space allocated for recording.

Sensor / Camera Communication:

Specific commands to sensor / camera for turning on and off (to conserve power) or take images in the Data Manager Interface. These packets of data will adhere to the interface supplied by the sensor

Software Upgrade Data / New sensor interface data:

These data will use compression algorithm to minimize the time of transmission (such as zip). The only constraint on its size will be that after it has been downloaded, there will be sufficient space left on Palantir-B to extract, install, while holding onto data from

sensors that are to be transmit back to the ground station later. This is not too big of an issue because we are implementing java and we only need to send the source code.

Periodic Data:

Battery Life: Assuming we are using a "smart battery," our power manager receives a percentage of the remaining battery life in the form of a small integer (e.g. we can treat 98.5 percent as an integer of 985)