A Satellite Ground Station Control System

Yu Du

Kongens Lyngby 2005
IMM-THESIS-2005-86

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

The current ground station is rather complicated or unimplemented according the research of the ground station software. For small student satellites, a simple and flexible ground station control system is necessary to be developed to contact with satellites.

The purpose of the master project is to accomplish a control system which fulfills the automation of tracking satellites, provides the remote access to the ground station, and monitors the system running status.

In this thesis, We shall analyze two existing ground station softwares: Mercury system and JStation. Then we shall present the design and implementation of the control system which has accomplished our goal for the project.

Key Words: Satellite, Ground Station, Control System

Preface

This master thesis has been accomplished at Informatics Mathematical Modelling(IMM), the Technical University of Denmark, from April 15 2005 to October 21 2005. The project constitutes the final work of the requirement for obtaining a Master degree in Computer Systems Engineering at DTU. The project was supervised by Associate Professor Hans Henrik Løvengreen of the IMM department at DTU.

First of all, I would like to express my deep-felt gratitude to my advisor, Hans Henrik Løvengreen for his advice, encouragement, enduring patience and constant support.

I would also like to thank my dearest parents for encouraging and supporting me during my study. Finally, I would like to thank all my friends for always giving me advices and inspiring me in my life.

Lyngby, 21 October 2005

Yu Du
s030972

Contents

List of Figures

Chapter 1

Introduction

1.1 General Introduction to Ground Stations

In the 21st century, satellites have become an essential technology for modern life. Among the important application of satellites are telecommunication, earth and space observation, global resource monitoring, military observation, global position (GPS), micro-gravity science and many others.

Satellites are launched from earth by the space shuttle, from high-flying airplanes, or from ground based rockets. Once lunched, the payloads must reach proper elevation and escape velocity to be boosted into orbit [8]. In order to maintain proper operation, satellites are controlled by a *Ground Station* on earth that sends commands and receives status and telemetry from the satellite. Low Earth Orbiting(LEO) satellites are visible for only a period of time from the point of view of an observer on earth. They can dump data to the Ground Station when they pass by a Ground Station area. Therefore, the Ground station plays a very important role in the communication with the satellites in the orbit. A Mission Control Center might use several ground stations to maintain the communication with satellites. Although a ground station will have a lot of radio hardware to receive and send the microwave signal to and from satellites, the objective of this work is to focus on the software part of the ground station. Figure 1.1 depicts the basic picture of the ground station and the mission

control center.

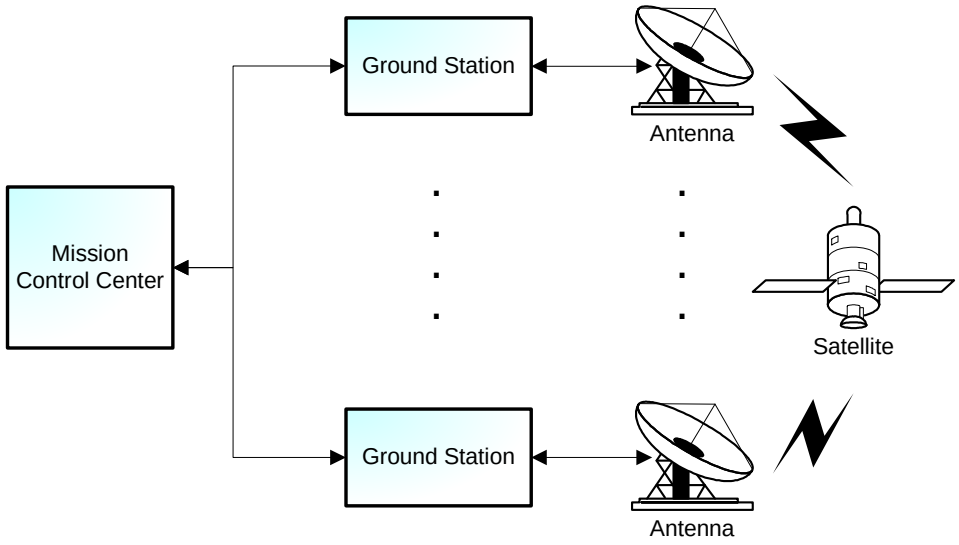

Figure 1.1: The Basic Ground Segment Architecture

1.2 Introduction of DTUsat

DTUsat is a small student satellite designed and built by students from Technical University of Denmark(DTU). In commercial satellites, the reason for launching a satellite is the payload. The rest of satellite, the platform, is built solely to provide the payload with necessary services such as power, communication etc [17]. A university satellite like DTUsat provides the payload and educates the engineers. Also, DTUsat has a lot of potential for public relations, especially for science in DTU generally. DTUsat presents a possibility to counter some of prejudices which exist on engineers and scientists in the public. The purpose of DTUsat is scientific, educational and long term public relations. This master project is done in the context of the DTUsat project.

1.3 Project Scope

The ground station for satellite communication range from large telecommunication stations to small stations for nano-satellites such as small student satel-

lites. The functionality of the ground segment software covers pre-pass command preparation, real time communication with the satellites during a pass, and post-pass telemetry data processing and dissemination.

For small student satellites such as DTUsat, currently existing standard software for ground station operation is either not affordable or too complex for the limited purpose and lifetime of the satellite. Therefore, it is of interest to develop a ground station which is simple and flexible enough to be used for different satellites. Some attempts have been made with this goal in mind, such as the Mercury System, but their state of development has not yet reached the maturity necessary for development outside their original domain.

As a simple and flexible architecture for a small scale ground station system, four major components are proposed. A *ground station control system* that establishes the real-time communication path with the satellite, a *satellite mission control center* for uplink and downlink communication with the satellite and *pre and post processing components*.

In this master project a generic ground station is to be developed. The system should allow for basic control of the ground station hardware such as radio and antenna. Further the system should enable an external satellite mission control center to set up *sessions* automatically controlling satellite passes. Also, the system should be able to accommodate different communication protocols used by different satellites including AX-25 based ones. Emphasis should be put on flexibility and robustness of the system.

This master project should be deployed in the DTUsat ground station which is situated in Building 348 at DTU. The implementation of the project is planned to be done using Java running on a Linux platform.

There are six chapters to describe the ground station control system in this thesis.

- Chapter one introduces the basic knowledge of the ground station and outlines the scope of control system.
- Chapter two introduces the fundamental functionalities of ground segment first. Afterwards, this chapter analyzes the two existing free ground station softwares: the Mercury System and JStation, and compares the two systems advantages and differences.
- Chapter three concentrates on the architecture design of the control system. This chapter analyzes the detailed design of each part of functionalities of ground station control system.

- Chapter four describes the implementation of control system using Java language. This chapter illustrates the class diagram of the modules in the control system and analyzes the important technical issues which have been involved in the system implementation.
- Chapter five depicts the testing strategy of the ground station control system including the unit testing, integration testing and on-site testing. This chapter introduces the major test cases for particular functionalities and the test environment.
- Chapter six is the conclusion of the control system. This chapter summaries the achievements of the system, highlights some important issues, and proposes the possible work in the future.
- Appendix A includes all the class diagrams of the project.
- Appendix B includes the source code of the project.
- Appendix C includes the test cases of the project which has been carried out in the testing phase.

Chapter 2

Ground Segment

Global satellite coverage has proven to be extremely desirable in recent years in military and civilian sectors in both communication and exploration application. Increased satellite coverage requires increased capacity of the ground segment which is controlling the main operations in the life circle of satellites. The ground segment provides the means and resources to manage and control the missions, to receive and process the data produced by the instruments. The operational ground segment is in charge of the satellite housekeeping, the mission planning, and telemetry reception from the satellite.

2.1 Functionalities of Ground Segment

Some big ground segments, eg. International Space Station(ISS), have the full ground station coverage around the earth-they can keep the communication with satellites going all the time. But for student satellite project usually only a single , local ground station is available to keep their contact with satellites. Therefore, they only have very limited time to communicate with satellites, normally 2 times per day.

The ground segment performs three major functions to maintain the state of

health and mission readiness of an orbiting satellite. These functions include *pre-pass*, *real-time*, and *post-pass* software [14]. Except for the most basic spacecraft systems, some aspects of each of these will be required in the ground segment. In addition, some very important configuration control procedures have to be adopted to safeguard the integrity of the software part of the ground system.

2.1.1 Pre-pass Phase

The Pre-pass is required to be performed in advance of the satellite passing by the ground station. Generally, there are four different part of functions, namely *Orbit determination and Prediction*, *Observation Planning and Scheduling*, *Command List Generation*, and *Simulation* [14].

Orbit determination and Prediction

- The orbit determination and prediction is the measurements taken to accurately locate the orbiting satellite. Generally, the orbit determination tracks satellites via radiometric techniques, focuses on earth-centered orbits, and relies on intensive numerical calculation. The three measurements are involved to predict the satellite's orbit: measuring the time taken by radio frequency(RF) signals for the round trip journey(station - satellite - station), measuring based on the doppler effect, the frequency shift due to satellite velocity, and measuring antenna orientation with repect to the azimuth and the elevation. For the student satellites, the expensive hardware needed for orbit determination is not available. Instead, they rely on the orbit determination done by *the North American Aerospace Defense Command(NORAD)*[19] in the form of so-called two line elements(TLE).

Observation Planning and Scheduling

- In order to plan the observation schedule for a typical satellite, the ground station needs to know where the satellite will be in the future [14]. The antenna control software needs to know where to find satellite during a pass which means that the ground segment has to point the right position towards the satellite's direction when pass time. The data processing programs need to know where it was when the data was accumulated. The data exchange between these components is summarized in Figure 2.1 [14]. The planning and scheduling provides all the functions required to plan, schedule and command the satellites.

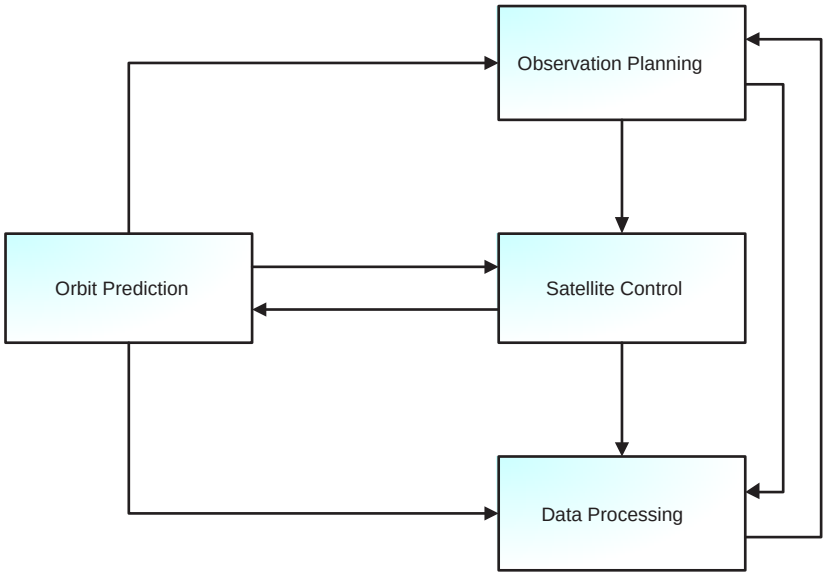

Figure 2.1: Observation Planning

The observation planning and scheduling should process all the requests from the clients, check the observation modes, attitude control or satellite constrains. It will schedule all observations optimized to ensure that the conflicts of observation would not happen in the schedule.

Command List Generation

- A *pass schedule* contains a set of instructions or commands that define the action to be taken during the pass, the tracking data, the new satellite plan and any other data. Normally, the pass schedule will be generated using some default options automatically. However, it is also allowed to construct the command list manually to override some default options by the clients

Simulation

- The spacecraft simulator is very important for testing the satellite's functions to aid problem solving in the event of satellite failures before launching the satellites into space. The simulation includes the capability to simulate both the satellite housekeeping data as well as the instrument

data. It also useful for checking the satisfactory handling of corrupted data. Another important task of simulation is to verify the command list after launching the satellites. The simulation uses the received data from satellites to fit in the command list which has been generated by the mission control center. The validation simulation can check whether the command list has some faults or not after launch.

2.1.2 Real-time Software

The real-time software is active during the whole period of satellite passing by the ground station. It consists of *Antenna tracking*, *Command uplink and verification*, *Data reception*, and *status checking* [14].

Antenna tracking

- The antenna tracking is the capacity for automatically aiming an antenna array towards the satellites position and also maintain accurate orientation in the direction of communication satellites. At the beginning of a satellite pass over the ground station, the antenna is commanded by computer to point towards the horizon of satellite's direction as expected. The antenna tracking system should control antenna resources, such as azimuth position, elevation position and antenna polarity.

Command uplink and verification

- Generally, the tasks of *Command uplink and verification* are to uplink individual and/or strings of commands from the previously generated pass schedule, and to verify the those commands have been correctly received and stored by the satellite on-board handling system. This process helps to detect uplink errors.

Data reception

- During a satellite's ground contact, data from the satellite's data storages is transmitted to the ground over a high speed telemetry (HST) link and stored in a database. Transmitted data from satellites involve not only image data but also housekeeping telemetry data including electric current of the on-board equipment and temperature etc. In general, these data are received by a ground station directly.

Status checking

- Real time data from the housekeeping telemetry frames are received at the ground station, both before and after the uplink of commands to the satellites [14]. These data are a mixture of status flags (eg. Subsystem on/off) and engineering parameter value, and must be checked and analyzed immediately, because these data contain parameters which are critical to both the health and operation of satellites. The command verification data from each frame are used to check that the commands sent to the satellites were received correctly. The housekeeping data is information about the satellite and its health and safety which comprises a set of satellite status indicators and sensor read outs, and the values of a selected group of these parameters can be extracted from each frame and displayed on control consoles at the ground station. If the Mission Control Center(MCC) discovers that the some mistakes occur through the quick health checking, the MCC switches to the failure mode to recover the problems immediately.

2.1.3 Post-pass Software

The immediate post-pass software is comprised by *Health assessment*, *Data processing*, *Orbit determination* and *Data analysis*. The tasks of post-pass software include extraction of the housekeeping and science data for quality control.

Health assessment

- After a pass, the ground station extracts and processed the engineering data in order to assess the success of the observation or experiment schedule on the satellite [14]. The data derived from low-speed telemetry(LST) link includes a set of timing points and a copy of the satellite event data, together with stored housekeeping data. The satellite support teams monitor the satellite's health and performance using the information and instrument which are provided by the health assessment.

Data processing

- Generally, the data from satellites is saved in the computer storage, and then processed to pull out the information of interest. The first task of data processing is further processing of housekeeping data to provide

subsystem engineers with all their products. The second task is processing of science/technology data from the raw telemetry.

Orbit determination

- Orbit determination is the process whereby the previous best estimate of the satellite's orbit is updated as a result of measurements taken during the ground station pass, and from other measurements. For most general purpose, it is sufficient to determine the satellite along-track position within its orbit to an accuracy of a few kilometers.

Data analysis

- Data analysis software is a important part of software development. It encompasses the software for analysis of tends in the engineering data as well as the complete analysis of all the science/technical data. It includes all the analysis and graphics facilities, as well as the often worldwide distribution of data to the clients.

2.2 MCC and Ground Station

The fundamental purpose of a ground station is to enable communication between ground users and space satellites. Besides the radio frequency(RF) equipment needed for microwave communication, the ground station traditionally have a suite of supporting systems for mission specific data handling needs, such as demultiplexing of data streams, encryption functions, data compression, time tagging, data storage, data quality ensurements and spacecraft ranging [5]. The ground station performs like a communication bridge between the satellites and the ground, sending the commands to satellites as the ground center requests and receiving the telemetry feedback data from satellites when they passe by the ground station horizon. Figure 1.1 shows the basic function of the ground station.

The primary purpose of the Mission Control Center(MCC) is to control the satellites in the different places and distribute the satellite data to the data center. The MCC works mutually with ground stations as discussed in the Chapter 1. Basically, the MCC could have more than one ground station to communicate with satellites, and they might be located at different places. The MCC and the ground stations have different roles and responsibilities in the

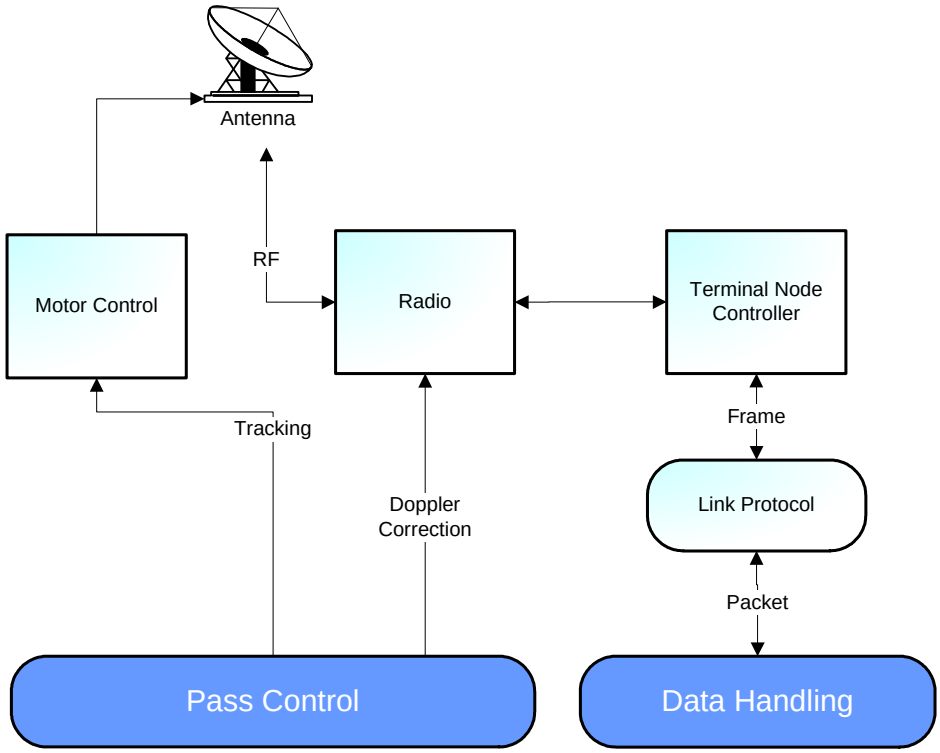

Figure 2.2: Basic Function of Ground Station

ground segment. The work load of pre-pass and post-pass has been done at the MCC side. The basic functions of the ground station shows as Figure 2.2. In the real time phase, the functions of MCC include *Command uplink and verification*, *Data reception*, and *Status check*. The functions of the ground station primarily consists of *Antenna tracking*.

2.3 Mercury System

As part of research program in spacecraft operation, the Mercury system [15] is a ground station control system to support advanced command and telemetry operations developed by a Ph.D student James Cutler in Stanford University's Space Systems Development Laboratory(SSDL). The Mercury system provides command console control, script program based autonomous control, and remote access client for commanding a controlling satellite ground stations via the In-

ternet. The goals/aims of developing the Mercury system are to reduce the cost of operating space missions and to increase mission yields and capabilities.

The initial idea of the research work on Mercury system is to obtain some ideas of designing the satellite control system, because the Mercury system almost outlines all the necessary functionalities for the control system, even though it is not implemented completely. Therefore, the study of the Mercury system was given us a lot of ideas about control systems.

2.3.1 Overview of the Mercury System

The Mercury system is being implemented on SSDL's Orbiting Satellite Carrying Amateur Radio(OSCAR) class amateur radio ground station. The station is used to command and control SSDL's satellites upon their launch into orbit. The purpose of Mercury system is to improve the operational efficiency of the station. Mercury achieve its goal by providing a centralized software interface to control all ground station equipments, routines to automate station operation, and internet gateway to access the centralized interface. Figure 2.3 illustrates the block diagram of the Mercury control system [3].

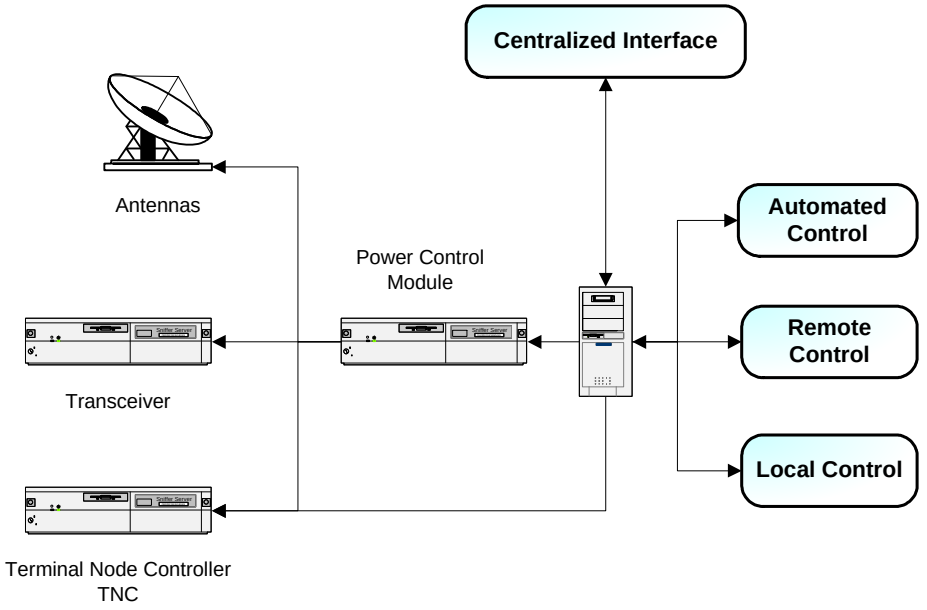

Figure 2.3: Block Diagram of Mercury System

Mercury's *centralized software* controls the drivers for antenna, transceiver, and TNC, provides a user interface to configure the ground station. The operators can control all the ground station equipment from a location.

The centralized software interface enables *automation* of ground station. The automation routines include:

- Moving the antennas to point the properly position during the satellites pass.
- Adjusting transceiver to compensate the doppler shift.
- Processing data collected from the TNC.
- Configuring the station for a particular contact session.
- Monitoring the equipment performance.

The *Internet gateway* allows the operator to access the full functions of ground station control system through the network protocol. It provides the remote access to the users.

2.3.2 Functional Architecture

Mercury is a loosely coupled system to provide internet accessible, mixed-initiative control of ground station. The API called the Ground Station Markup Language(GSML) is introduced to provide basic building blocks to ground station users in Mercury system. The users can compose ground station services by the GSML builder. GSML encapsulates core state information associated with the hardware and services of the system model and provides primitives for actuating ground station elements(similar to remote procedure calls). Mercury system implements the GSML protocol for controlling a hierarchical ground station network. The high level diagram of Mercury system components is shown as Figure 2.4 [15].

The Mercury model is hierarchical and layered according to autonomy level. The *virtual hardware level* control all the low level hardware resource using the

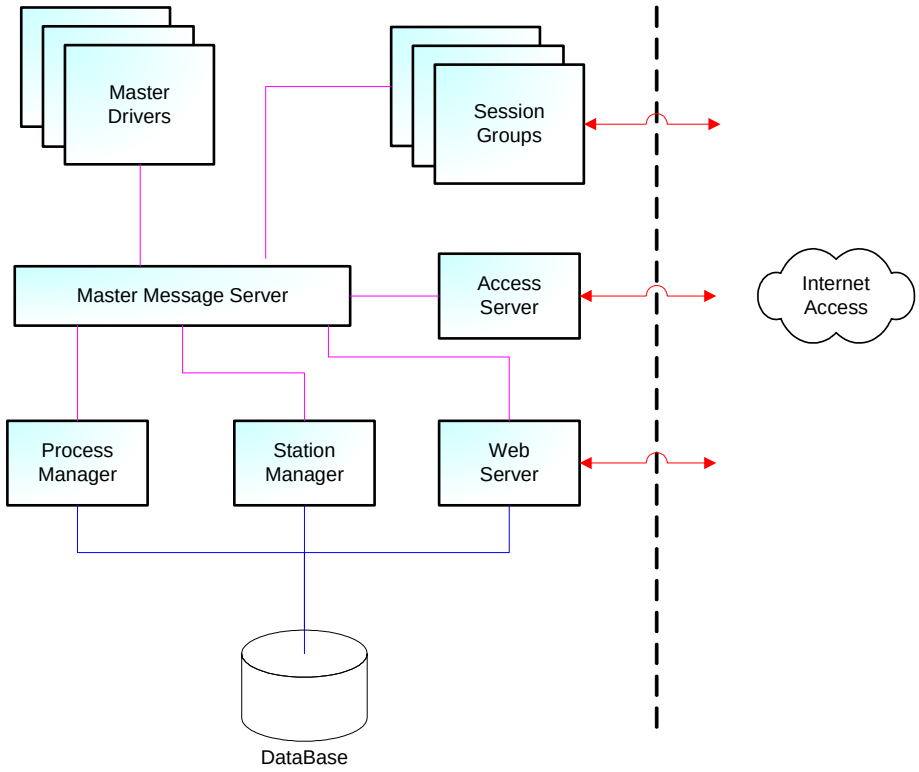

Figure 2.4: Block Diagram of Mercury System

different hardware drivers to manipulate them. The *session level* provides single station automation services. The *network level* catches the services in a ground station network.

Virtual Hardware Level

- The virtual hardware level captures the basic functionalities of low level ground station hardware and enable to command and control the different hardware components. In this level, the Mercury system abstract away the hardware device protocol and present a set of standard interface for the different hardware such as, antenna, radio, power controller in the ground station. To simplify the specification of ground station capabilities, Mercury introduces *pipelines* to group the communication devices. Many ground station have multiple capabilities to configure the diverse devices, such as controlling the several antennas. Therefore, the high level

operation only need to manipulate the pipeline to specify the way to control the devices.

Session Level

- At the session level, Mercury sketches the automation tasks and services of the ground station. The session object has been introduced in the Mercury system. The session object is constructed by the users, to simulate all the necessary operations which reserve the hardware resources and services in a specific interval of time, to maintain the communication channels between the ground station and satellites during the pass. Figure 2.5 is the component view at the session level [5].

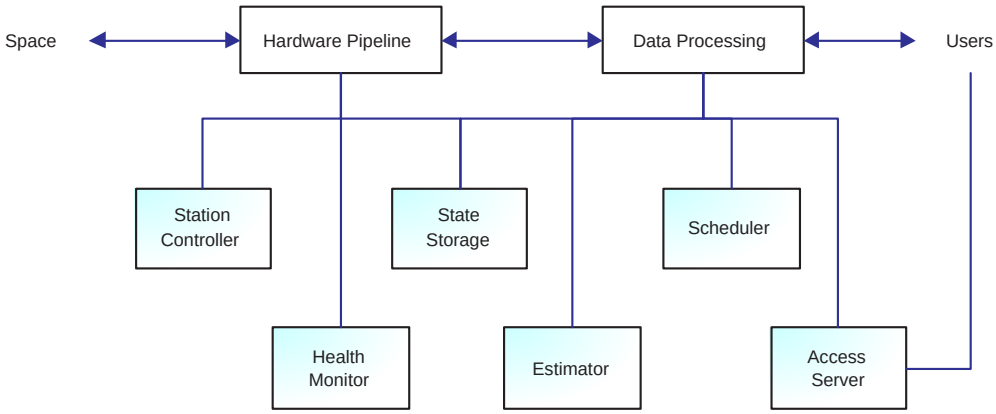

Figure 2.5: Block Diagram of Mercury System

The *scheduling service* accepts reservations for the use of ground station systems. Resource availability and user access priority are taken into account during the scheduling of sessions.

The *station controller* automates execution of satellite contact sessions. It monitors scheduled contact session requests, configures ground station hardware to support the requested automation and data processing services. It performs real time control of station hardware to maintain and optimize communication channels during a satellite pass. The controller also manages health monitor output and recovers from failures automatically or alert on non-recoverable errors.

The *health monitor* consists of sensors, both software and hardware, logging resources for storing telemetry and health assessment functions for detecting failures and performance degradations.

The *estimator* determines the target satellite position, antenna pointing angles, and dropper correction factors.

The *state management* service stores session descriptions and the schedules, session products.

The *remote access* server authenticates remote users and provides secure, encrypted ground station control.

Network Level

- The network level captures the service of a federated ground station network(FGN) which provides global, cross mission support. The purpose of Mercury is to set up the ground station network and make multiple ground stations working together to coordinate the resources to optimize the communication channels. As a result, the ground station network provide the longer contact time and more functional services for the sessions to communicate with satellites.

2.4 JStation System

The Java Satellite Ground Station is an attempt to write a software package to operate the digital store and forward satellite which is written in java and is portable across multiple software and hardware platforms [16]. We did not spend too much time on the JStation system since it does not have any system development documentation and specification. In other words, if you want to know more about JStation, you have to investigate the source code of JStation system. That must be a very time-consumed work. The JStation includes support for the following as stated in the web site.

- Kiss Protocols
- AX25 Protocols
- FTL0 Protocols
- Directory and Message Downloading

- Message Composer
- Automatic Message Downloading
- Automatic Message Uploading
- Satellite Orbit Propogation
- Satellite Scheduling
- Antenna Rotor Control
- Radio Control

2.5 Summary of Ground Station Softwares

2.5.1 Comparison of Mercury and JStation

The Mercury system has defined a very ambitious picture for the ground station. It coves almost all the functionalities which we currently know about the ground station. The Mercury attempts to provide the federated network of the ground stations to share the resources. The Mercury system lacks the proper documentation specially for the source code in the implementation and only few components are functional. The goal of Mercury system is too far from being reached. But some ideas of Mercury system could be used in our control system like the session level which sketches the automation tasks and services of the ground station. Because the documentation of JStation system is barely existing, we are not clear about the goal and design architecture of JStation currently. For the research of the source code in JStation system, it seems to implement some satellite communication protocols like: KISS protocol, AX-25 Protocol and FTL0 protocol. The implementation of the protocols may be used in our ground station control system. Some implementation of the antenna rotor control and radio control in JStation also might be useful for our system.

2.5.2 Drawback of Current Control Systems

From the analysis of the current ground station control systems, obviously, there are some drawback which are existing in the current control systems. Here, we brief some major drawbacks in current control systems.

Lack of Automation

- The operation of the ground station is very tedious and labor intensive. The operator has to manage the ground station for every pass whenever the time of pass is day or night. Naturally, the requirement of automating the basic operation of ground station has emerged. The automation should involve a central process to control the other processes, handling multiply tasks in the same time.

Lack of Remote Access

- Since the time of satellite pass is limited and all the operations have to be done in the ground station, it is necessary to provide the network interface in the ground station control system. So the operator can access the ground station resources and monitor the status over the internet.

Limited Communication Time with Satellites

- The communication time between the ground station and low earth orbit(LEO) is very small. The limited time restricts the amount of data which are receiving from the satellites. The distributed system of ground stations is used in large space control system. For small student satellites, the ground station should fully utilize the limited pass time of the satellites.

2.5.3 Improvement of Control Systems

From the above analysis of drawbacks, the control system should provide the automation execution in the ground station, the remote access for users and the full utilization of the communication time with satellites. We should focus on this three issues in the control system design.

Chapter 3

The Architecture Design

From the comprehensive discussions in Chapter 2, we have already known the basic functionalities of the ground segment, the mission control center(MCC) and the ground station. For this master project, the real time phase is the essential part as discussed in the scope of the project in Chapter 1. The functionalities of the ground station are the main emphasis of the thesis, specially the control system of the ground station. In this chapter, we will analyze the functionalities of the control system and delineate each of them in details.

3.1 Requirements for a Ground Station

The requirements of the ground station are the important part of the control system because the architecture design of the control system is based on the requirements. In general, the requirements of a ground station is divided into two parts: the services and the control. Figure 3.1 shows the context diagram of the control system.

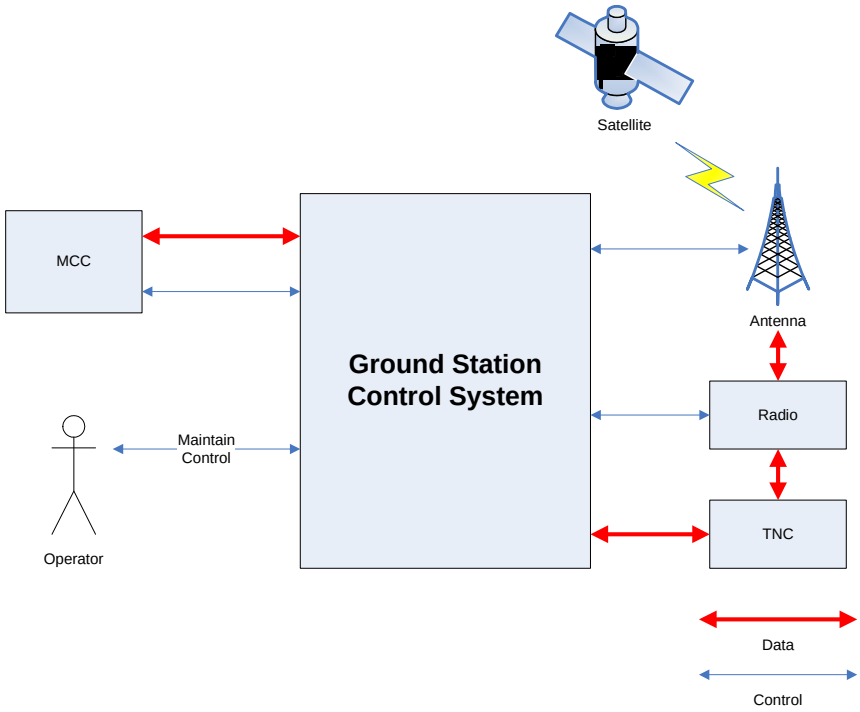

Figure 3.1: Context Diagram of The Control System

3.1.1 Ground Station Services

The ground station services include the following three functionalities: *satellite tracking*, *data and protocol handling*, and *logging*.

Satellite Tracking

- For the satellite tracking, the control system should be able to command the basic ground station hardware such as the antenna and the radio. The satellite tracking should be autonomous to track the satellites based on some configuration files. Due to predictable conditions of the satellite movement in the space, the satellite tracking system can calculate a satellite's position for the given moment using the prediction software. The calculations of the satellite's position can be done based on known orbit parameters such as Two Line Element(TLE) [33]. In order to keep the satellite tracking working precisely, the system should provide Doppler

correction to update the frequency of the radio.

Data and Protocol Handling

- In order to communicate with different satellites, the control system should have the data and protocol handling which can talk with the low level protocols to establish the communication channel with the satellite, and with high level protocols which are used to set up the network connection to transfer the packages with the mission control center(MCC). The protocol can use some external programs to set up a protocol stack, such as an *AX-25* [20] based protocol stack. In order to establish a communication between the low level protocol and the high level protocol, a data translation system should be used to accomplish this goal in the data and protocol handling service.

Logging

- The control system should provide an integrated logging mechanism to log the information including normal events, debug events and execution events. The logging system also should be configurable to choose to log information to a file, console or the database on the given level and module.

3.1.2 Ground Station Control

Automatic Control

- The control system should establish an external mission control center(MCC) to create the automatic pass schedule known as the session. The session includes all the properties needed to communicate with satellites like the start time and the end time. The control system should be able to execute the session automatically when the session is ready to track the satellite. If the MCC wants to send more than one session in a short time, the control system should have the scheduling mechanism to handle multiple sessions.

Manual Control

- Besides the automatic control, the control system should provide another more flexible manual control in which the operators can command the

ground station by themselves. The operator should be allowed to monitor the status of the control system, such as the antenna position and the radio frequencies.

From the above analysis of the requirements for a ground station, the basic functionalities of the ground station control system should include satellite tracking, protocol handling, logging, pass schedule execution and some external interfaces.

3.2 Satellite Tracking

The satellite tracking system provides the interfaces with the antenna controller and radio controller to track the satellite when it passes overhead. So the satellite tracking system responds to the tracking goal by correctly configuring the appropriate hardware and software to provide the requested communication services. Some orbit prediction software must be used to calculate the pass time and position of satellites. The satellite tracking system should prepare the antenna to point towards the predicted position right before the satellite pass, and track the satellite during the pass. In the satellite tracking system, the antenna driver must be commanded to follow exactly the predicted path of the satellites passing the ground station, and the radio driver must be able to change the frequency of the radio and complete the Doppler correction of the radio. The satellite tracking system diagram is shown as Figure 3.2.

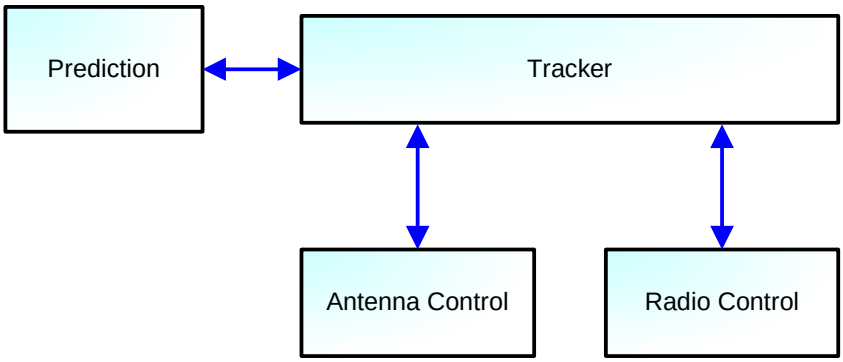

Figure 3.2: Satellite Tracking System Diagram

3.2.1 Prediction

Orbit prediction is an ability to predict the orbit of a satellite around the earth. To calculate the orbit of each satellite, the ground station must reference it from a known position in the space. The ground control facility needs to accurately determine a satellite's position in order to acquire and downlink data. The prediction component provides calculations for azimuth/elevation antenna pointing angle. The orbit prediction software is also used to calculate the pass time of the satellite. The tracking system has to adopt an external prediction program using two line elements(TLE) to calculate the orbit of the satellite in order to decide the antenna angle.

3.2.2 Antenna Control

The antenna control must have the driver for the corresponding antenna to control the ground station antenna resources, such as azimuth position, elevation position and antenna polarity. The purpose of the antenna control is to set azimuth angle, azimuth error window, elevation angle, elevation error window, and polarity direction etc.

3.2.3 Radio Control

The driver for the particular radio must be programmed to control the ground station radio resources in the radio control. The purpose of the radio control is to enable the setting of radio parameters, such as radio transmit frequency, radio transmit mode, radio receive frequency, radio receive mode, squelch level, output power level, input attenuator level, preamp power, and IF filter width etc.

3.3 Data and Protocol Handling

The protocol component of the control system should be able to deploy different communication protocols used by different satellites including AX-25 based ones. There should be one protocol instance for each protocol in the control system and the system should be able to use the correct protocol for the communication with the satellite. The data handling component processes the incoming real time data and converts the raw data.

The protocol component sets up the communication channel to receive the command list from the mission control center(MCC), and also sends the command list to the protocol stack which is prepared to talk with the Terminal Node Controller(TNC). On the other hand, the TNC receives the telemetry data from satellites during the pass, and then sends the data to the protocol component. Afterwards, the protocol component can send the data back to the MCC for the data processing. The block diagram of data pass is shown as Figure 3.3.

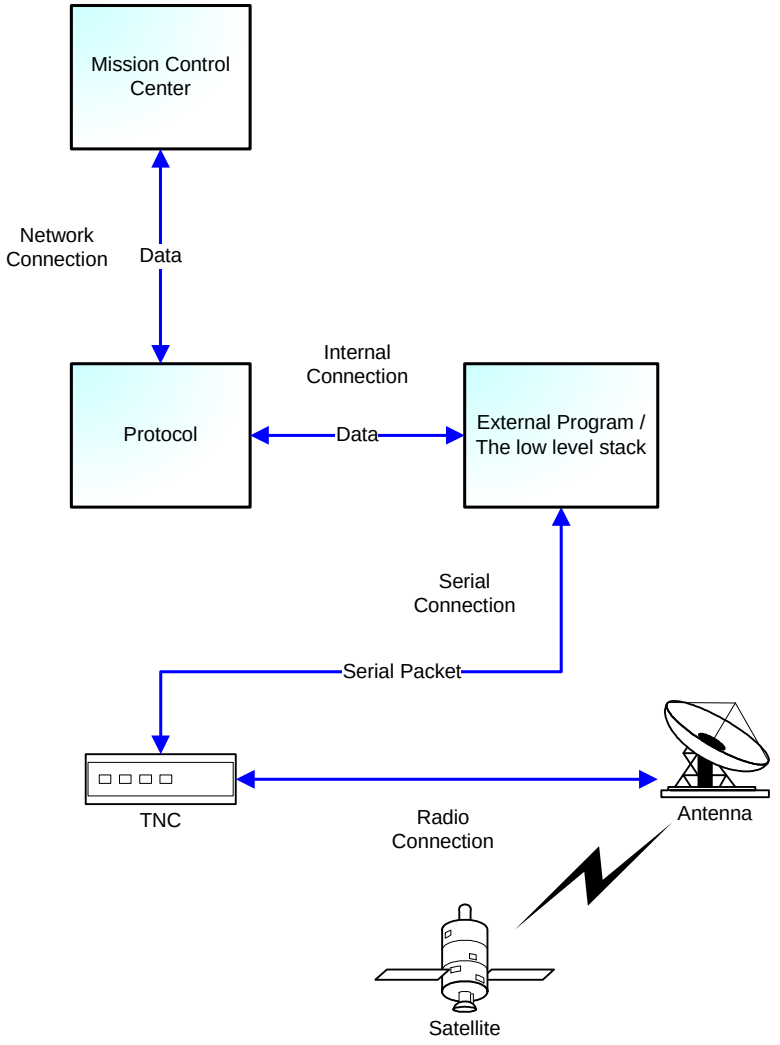

Figure 3.3: Block Diagram of Data Pass

3.4 Logging

In general, the log stores the system events and some data acquired during the communication with the satellites. A wide range of data, mostly telemetry data but also status and error information is stored in the relational database or files. For the ground station control system, we should concentrate to log the system normal events, debug events and execution events information etc. Also, these information can be logged in a file or on a console, probably a relational database in the future work. Different components may choose to log different level messages. For instance, the satellite tracking module may only log messages which have higher priority than "Warn" level if the severity of the satellite tracking module is set to be above "Warn" level. In that case, only messages that have lower priority than "Warn" will not be logged in the control system. Also, the modules can decide to display the log messages to various outputs, such as a file, or the console.

The logging modules can be set to only display the log messages of a given level. Such logging mechanism provides a very flexible means to be used by all the control system components, and can be configured corresponding to different system development stages.

3.5 Ground Station Control

The ground station services have been discussed in previous sections. Satellite tracking, data and protocol handling and logging must be accomplished in the ground station control system. Obviously, in order to make all these services work together and interact with each other, the ground station must have some control mechanisms to cooperate the services of the ground station. Some means to control the services of the ground station will be discussed in following sections.

3.5.1 Session

For each pass schedule, the mission control center(MCC) creates a session object which contains all the information for a satellite contact, and sends it to the ground station. The ground station requests for a contact session which reserves the station hardware over a specific interval of time and commands. A session scheduling service is available to users for reserving the ground station services

and hardware. A session agent should be able to alter the antenna position and the radio frequency by sending the parameters to the satellite tracking system.

3.5.2 Session Scheduling

Since the pass time of satellites is time limited, session scheduling is a very important issue in the ground station control system. A good session scheduling can fully utilize the control system resources to make the system effective. To make a flexible scheduling mechanism of satellite contact sessions, we should construct a sorting algorithm to queue the incoming sessions from the mission control center(MCC), and also verify the session's feasibility. If it is not a feasible session, this session should not be executed by the control system and the system should notify the mission control center the failure reasons of the session. The interaction between the MCC and the session scheduling is shown as Figure 3.4.

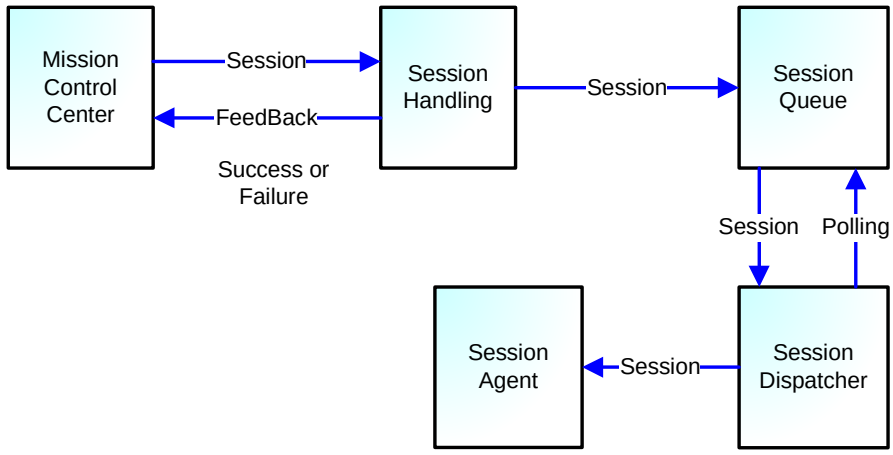

Figure 3.4: Interaction of MCC and Session Scheduling

3.5.3 External Interface

For the integrated control system and testing purpose, some external interface must be provided in this project. We have mentioned that the mission control center(MCC) works closely with ground station.

We have the satellite tracking, data and protocol handling, and logging modules

as discussed in the previous sections. In order to cooperate different modules in the ground station control system, a *ground station manager* component is used to make different modules work collectively and effectively.

Because the control system provides manual control services to manipulate the resources directly, a reasonable manual operation interface must be implemented to allow operators to control the ground station.

Manual Operation

- The manual operation is another flexible method to control the ground station resources. The use case of the manual operation is shown as Figure 3.5.

  The operator can start the satellite tracking system to control the basic ground station hardware such as radio and antenna, start the protocol as a bridge to establish the communication channel with the MCC and low level devices such as the terminal node controller(TNC). The manual operation also provides system monitoring function to evaluate the health of the system in case of errors. If a session is running in the control system currently, the manual operation should be forbidden, but the monitoring function is still available for the operator. The operator is allowed to kill the session in case that the session goes wrong.

Mission Control Center

- The mission control center(MCC) is an important external interface for the automatical execution of the ground station control system. The use case of the MCC is shown as Figure 3.6.

  The mission control center(MCC) should be able to create a session object which contains all information to contact with the satellite like a pass schedule. Obviously, the MCC should be able to send the session to the ground station. If something goes wrong with a session which has already been sent to the ground station, the MCC should be allowed to cancel this session object if it has not been executed yet.

  In addition, the main purpose of the MCC is to communicate with the data and protocol handling in the ground station. The MCC should set up the network connection with the protocol components. The MCC can send the commands to the protocol components, and also receive the telemetry data from them.

Ground Station Manager

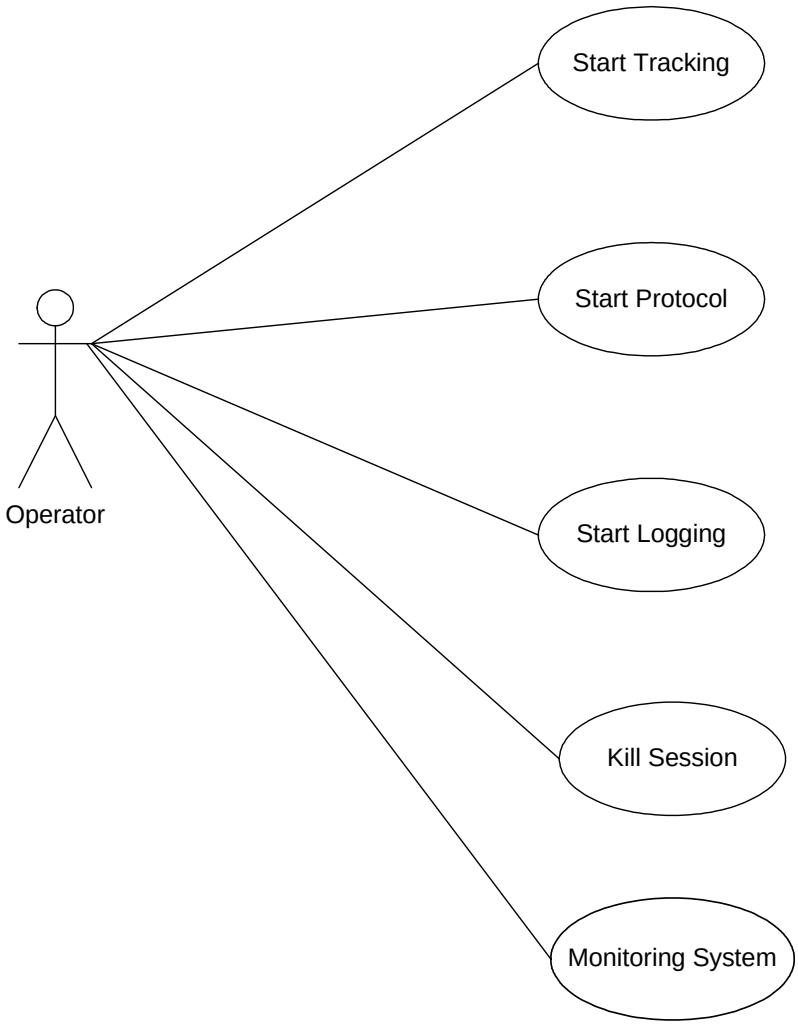

Figure 3.5: Use Case of Manual Operation

- The ground station manager(GSM) plays a vital role in the control system. The GSM is the main gate for the external interface such as the mission control center and the manual operation. All the operations from the external interface should be executed through the GSM in the ground station control system. The block diagram of the ground station manager is shown as Figure 3.7

  one of the ground station manager's purposes is to receive the session

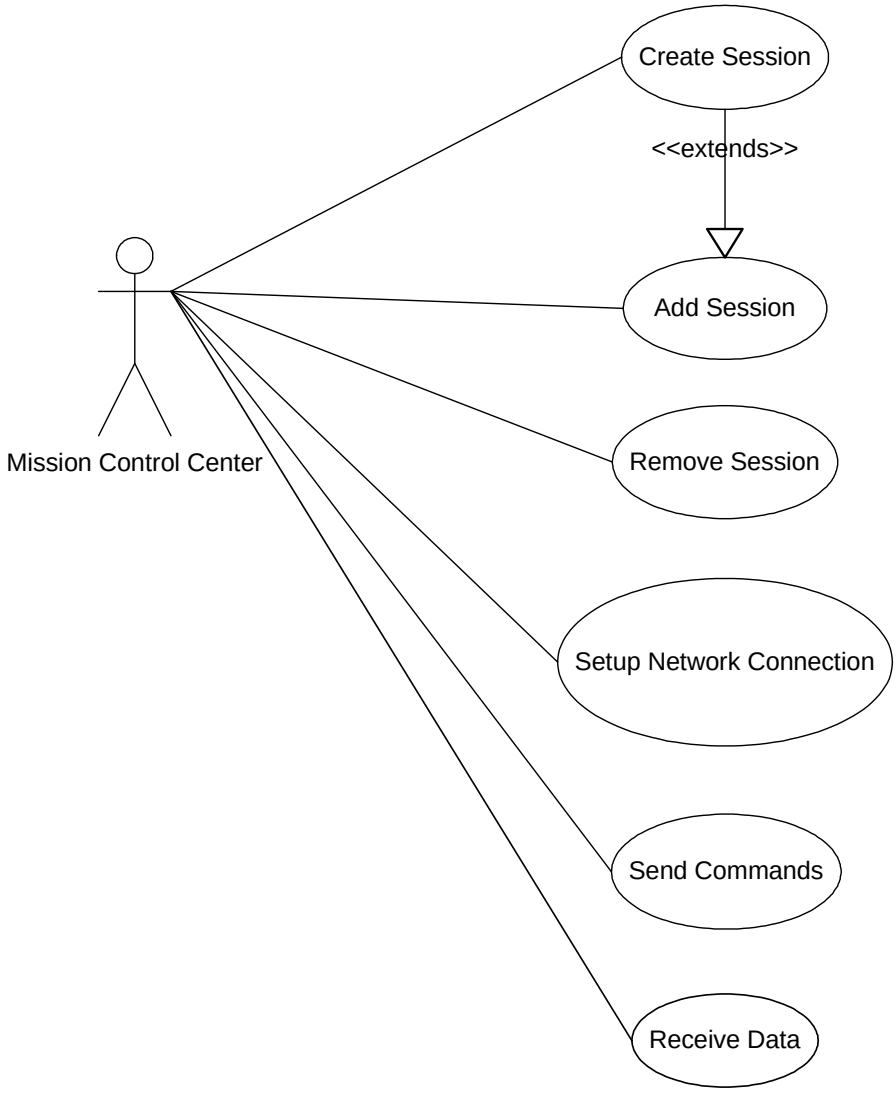

Figure 3.6: Use Case of MCC

object which is previously created by the Mission Control Center (MCC), and verify that the session object has been received correctly and transfer the session object to the session agent component in the control system. The problem of the satellite commanding is the time limitation. Therefore, the session objects should be done and transferred to the session agent before the satellite passes by the ground station. The ground station

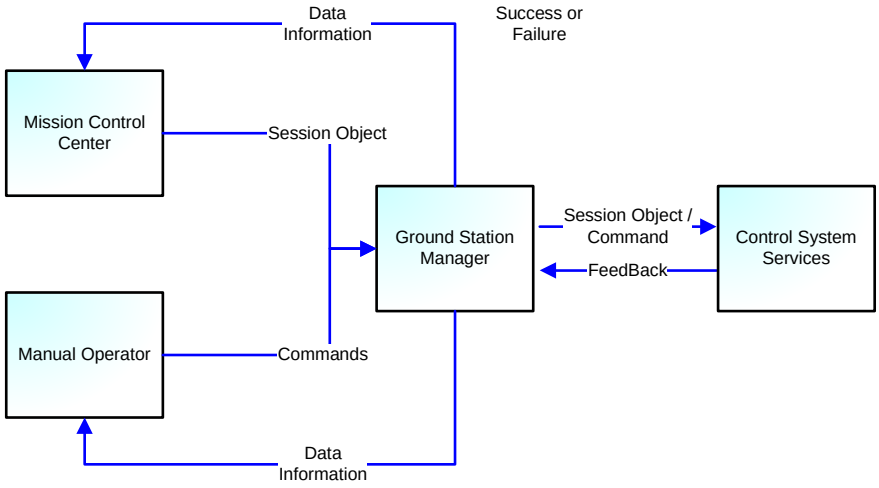

Figure 3.7: Block Diagram of Ground Station Manager

manager should be the entry point of the control system who wants to contact the satellite through the ground station.

On the other hand, the operator also can manually initialize a pass schedule through the ground station manager services, such as preparing the satellite tracking and setting up the network connection with the MCC. All the actions of the operator should be done through the ground station manager to validate the correctness and initialize the control system services. The monitoring functions should also be accessed through the GSM in the control system.

3.6 Architecture of The Control System

From the above analysis of the ground station control system, the five major functionalities of the control system include *Ground Station Manager*, *Session Agent*, *Satellite Tracking*, *Data and Protocol Handling*, *Logging*, and *Session Scheduling*. The software component view of the ground station control system is shown as Figure 3.8. We have discussed the main functional components of the control system in the previous sections. Figure 3.8 depicts the interaction of all the components in the ground station control system.

The external users like the operator and the mission control center(MCC) can

use the control system to manipulate the ground station. All the operations from the external users are executed by the ground station manager(GSM). The GSM can access all the main components of the control system such as the session agent, the protocol and the tracker for the automatic control or manual control purpose. The connection is set up with the MCC, protocols and the external program to transfer the satellite data into the system. The satellite tracking subsystem can be used by the GSM or the session agent. All the components have access to the logging system to record the system events.

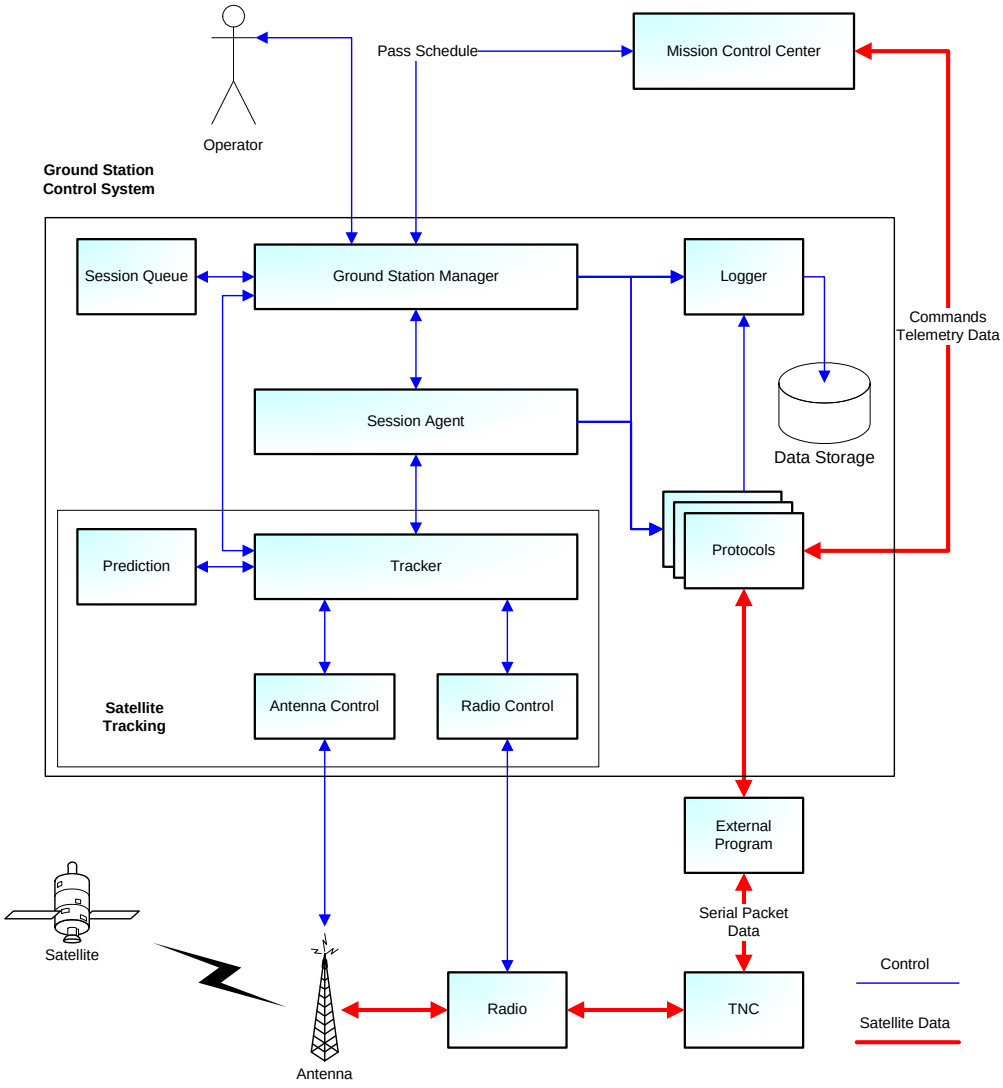

Figure 3.8: Architecture of Control System

Chapter 4

The Implementation

All the functionalities of the ground station control system have been analyzed elaborately in Chapter 3. In this chapter, we will describe the detailed implementation of the control system, such as the interface design, class design, and analyze some interesting technical issues in the development of the system. The implementation of the ground station control system uses the *Java* [22] language running on the *Linux* [23] operating system.

4.1 Interface Design

Interfaces provide more sophisticated ways to organize and control the objects in a system [13]. Interfaces have a feature which classes do not have, that is they can be multiply inherited. A class can implement multiple interfaces, but it only can extend one superclass to inherit the variables and methods. In general, interfaces are frequently used in large project development. The mechanism of interfaces in Java provides easy maintenance, reusable code, plugin compatibility, and compilation efficiency. Since interfaces have a lot of advantages for software development, we decide to design interfaces for major components in the ground station control system's implementation. The overall picture of interface usage is illustrated in Figure 4.1. The interface design of each component will be discussed in the following sections.

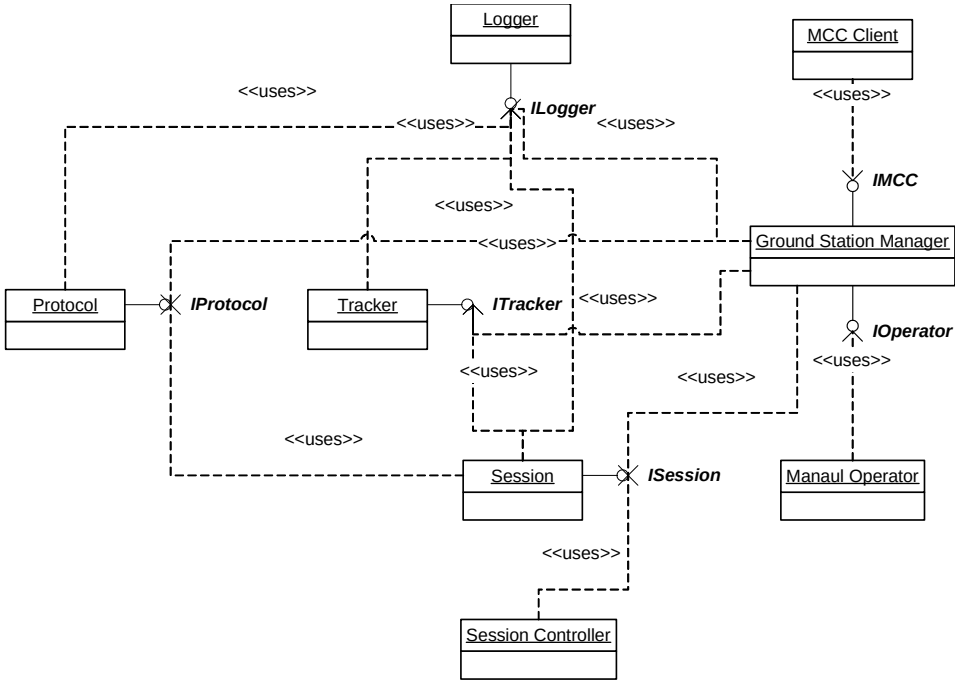

Figure 4.1: Interface Usage of The Control System Components

4.1.1 Protocol

The ground station manager(GSM) and the session agent can start the corresponding protocol thread for a particular satellite. Therefore, the protocol interface should have *startProtocol()* method which can be used by the GSM or the session agent. If the protocol thread suffers from a fatal error in the execution and the thread cannot terminate normally by itself, the GSM or session agent should be able to kill the inaccurate protocol thread via the method *stopProtocol()*. Because the protocol is responsible for setting up the satellite communication channel with the mission control center(MCC), the method *connectTo()* should be used to make a TCP/IP [24] connection with the MCC. In addition, the GSM or the session agent should be allowed to get the protocols running status via the method *getStatus()*. The users of the protocol module only need to access the public methods in the protocol interface, but not the protocol object itself. Such interface design ensures that the users do not have to care about the implementation of protocol. Their concern is the protocol interface. The source code of the protocol interface refers to List B.14. The interface design of the protocol is shown as Figure 4.2.

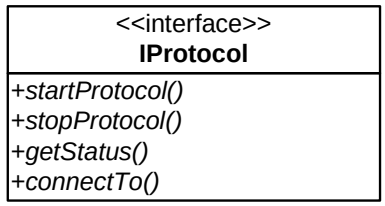

Figure 4.2: Interface Design of The Protocol

4.1.2 Session

The session is a pass schedule which is created by the mission control center(MCC). The implementation of a session can be done in various ways. For example, the session can be the configuration script which can be interpreted by the session agent, or the session can be a object which contains the pass schedule parameters and can be run as a thread by the session agent. Here, we choose the second means, the session object, to be implemented in the control system.

Basically, the session should have the method *startSession()* which allows the user to start a new session thread, and the method *stopSession()* to kill the session in case that a fatal error happens in the session thread. Since the session object is created by the MCC, the session does not know the other components in the control system. But the session has to interact with the others, the method *setEnv()* gets the reference of the other components in the control system, such as the session agent, tracker, protocol, and logger. The session also has to periodically report its running status which includes some parameter and data values to the ground station manager through the method *getStatus()*. The session ID can be known via the method *getSessionID()*. The method *getSessionName()* returns the session name to the users. The start time and the end time of the session can be fetched by the method *getStartTime()* and *getEndTime()*. The method *getProtocolType* returns the protocol type for the communication with the satellite. The method *getSatelliteName()* returns the satellite name. The satellite frequency can be obtained via the method *getFrequency()*. The source code of the session interface refers to List B.26. The interface design of a session is shown as Figure 4.3

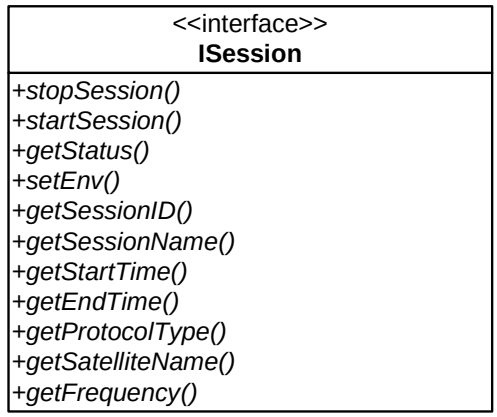

Figure 4.3: Interface Design of The Session

4.1.3 Tracker

The tracker has been done by other students in a 3 weeks special course at Informatics and Mathematical Modelling(IMM) [31]. They have developed the tracker component for the satellite ground station, implemented a radio and motor driver which can communicate with the existing hardware equipment, and modified a Java library for predicting the satellite position. We abstract some method and modify the code based on their project.

Because it takes some time to move the antenna, we should prepare the tracking before the satellite pass using method *initTrack()*. When the time is exactly the start of the satellite pass, we start the tracking of satellite via the method *startTrack()*. A checking routine *checkAntennaPosition()* should be used to validate whether the antenna is in the initial position as predicted before starting the tracking system. If something goes wrong with the tracking thread, eg. it is stuck in some place and cannot finish the tracking task, we should be able to kill the abnormal tracking thread to recover the hardware by the method *stopTrack()*. Before the tracking system starts, we should specify the satellite which we want to track through the method *initSatellite()*. If the ground station manager or session agent wants to know the status of tracking, the tracker should send back the status report using the method *getStatus()*. The method *getExpectedPos()* returns the expected position of the antenna which is calculated by the external prediction program. The source code of the tracker interface refers to List B.34. The interface design of the tracker is shown as Figure 4.4

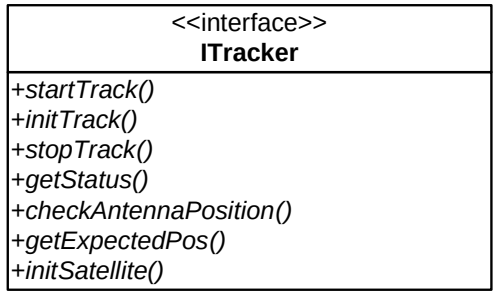

Figure 4.4: Interface Design of The Tracker

4.1.4 Ground Station Manager

The ground station manager(GSM) is in charge of the communication with both the mission control center(MCC) and manual operation. So the GSM has to implement an interface for each of them. The MCC creates the session object which has to be sent to the GSM, and the GSM processes the session after receiving it. The operator manually controls the protocol and tracker, monitors the status of the control system and kills the running session if necessary. All the actions of the operator have to be processed by the GSM.

Interface for The MCC

- The purpose of the MCC is to prepare the session object and send it to the GSM. So the method *addSession()* is necessary for adding a new session. If the MCC wants to withdraw the requested session, it can use the method *removeSession()* to do it. The source code of the MCC interface refers to List B.8. The interface for the MCC is shown as Figure 4.5.

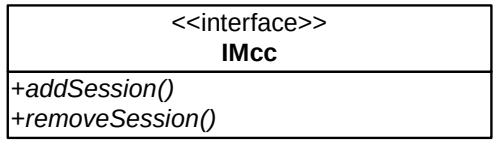

Figure 4.5: Interface Design of The MCC

Interface for Manual Operation

- If the operators want to control the system manually, they have to access

the components of the control system via the GSM. The operator starts a protocol thread by using the method *startProtocol()*, and stop it by using the method *stopProtocol()*. The method *startTracker()* starts a new tracking of the satellite. If the tracker suffers fatal errors while running, the operator can force to terminate the tracking thread through the method *stopTracker()*. Obviously, the operators would like to monitor the whole system status. They can get the protocol status and the tracker status by calling the method *getProtocolStatus()* and *getTrackerStatus()*. If a session is running currently in the satellite ground station control system, the manual operations should be disable and only enable the monitoring functions. If the operators detect that the session is running erroneously, they can kill the running session by the method *killCurrSession()*. The source code of the manual operation interface refers to List B.9. The interface for the MCC is shown as Figure 4.6.

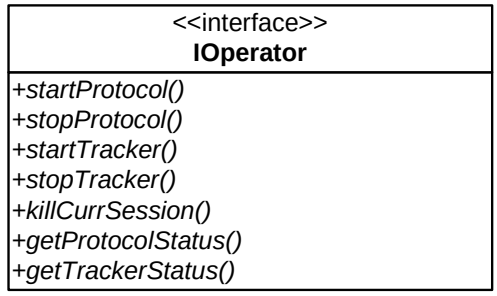

Figure 4.6: Interface Design of The Manual Operation

4.2 Implementation of Ground Station Manager

The ground station manager plays an important role in the control system of making different components communicate with each other. The GSM receives the session object from the mission control center, validates the correctness of the session object, then passes it to the session agent. The operators control the system manually through the GSM module. The GSM is a bridge for the external users like the MCC and operators. The class diagram of the ground station manager is shown as Figure 4.7.

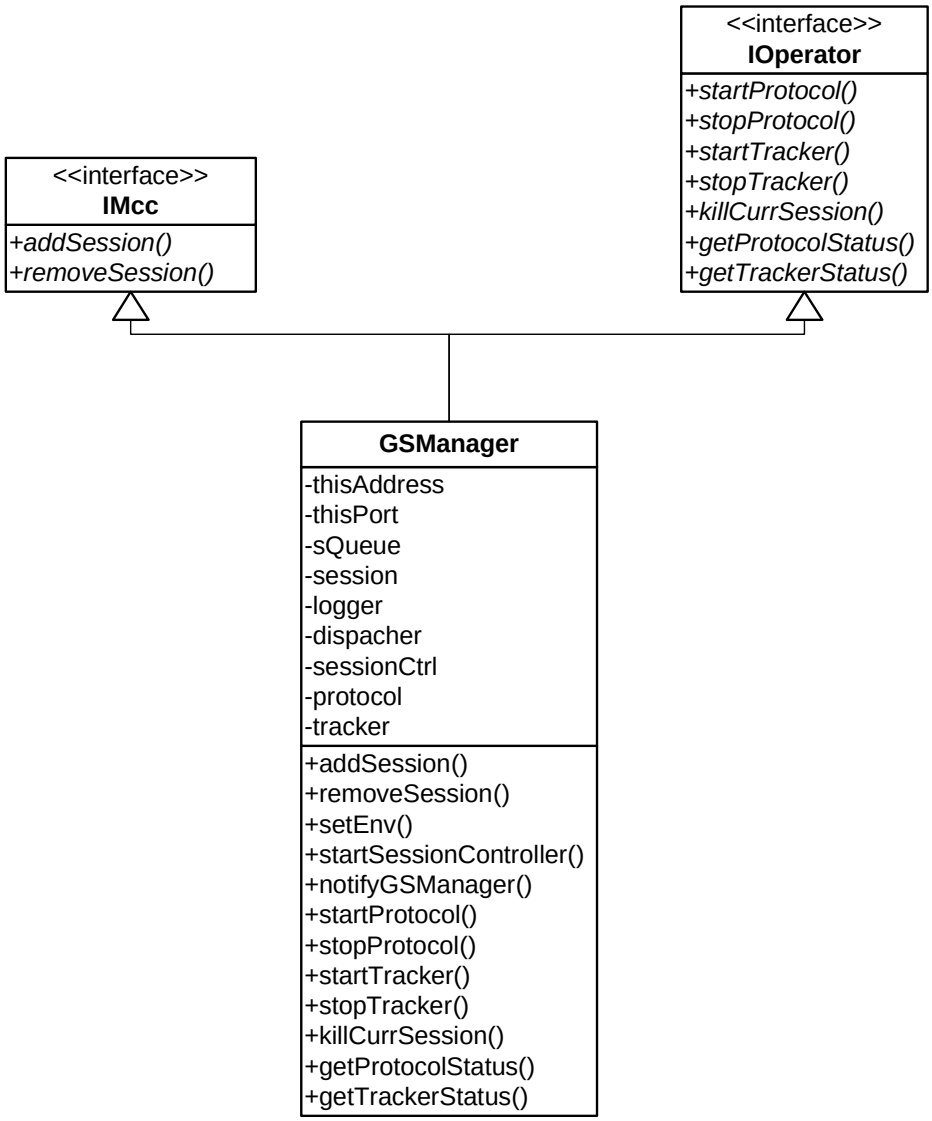

Figure 4.7: Class Diagram of The Ground Station Manager

4.2.1 Session Queue

We discussed the session scheduling in Chapter 3. The session schedule that we call session queue in the control system provides the reservation services of

the ground station resources for the users. The session queue validates the time correctness of the incoming sessions from the mission control center(MCC) and stores the correct sessions into an array list. If the start time of the new session is earlier than current time or the time slice of the new session overlaps with the current sessions in the session queue, the session queue will reject the new session and throw an exception. If the time of the session is correct and does not conflict with others, the session queue should insert new sessions into the proper position of the session array using the method *insertQueue()*. Sometimes, if the MCC wants to delete a session which has been sent to the session queue, the ground station manager can use the method *removeSession()* to delete this session in the session queue. The class diagram of the session queue is shown as Figure 4.8

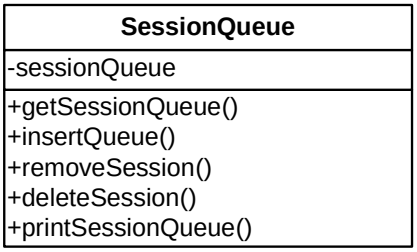

Figure 4.8: Class Diagram of The Session Queue

There is another concept called session dispatcher related to the session queue. The session dispatcher is an independent thread started by the ground station manager for polling the session queue periodically. If the session dispatcher finds that the start time of a session is just few minutes later than the current time, then the session dispatcher should notify the GSM to start this session immediately. After the session finishes the contact with the satellite, the GSM has to do some clean-up work, such as removing this outdated session from the session queue.

4.3 Implementation of Session

The session is a pass schedule constructed by the mission control center(MCC) and sent to the ground station manager(GSM). So the session should have some basic properties related to communication with satellites, such as the start time, end time, satellite name. The method *startSession()* provides the service to start the session thread, and *stopSession()* to kill the active session in case that the session has something running errors. In order to provide remote access,

the mission control center should send the session object via the Java Remote Method Invocation(RMI) [21], so the session should implement the Serializable interface. Because the session should run as a thread in the ground station control system, the session also has to implement the Runnable interface. The class diagram of the session is shown as Figure 4.9

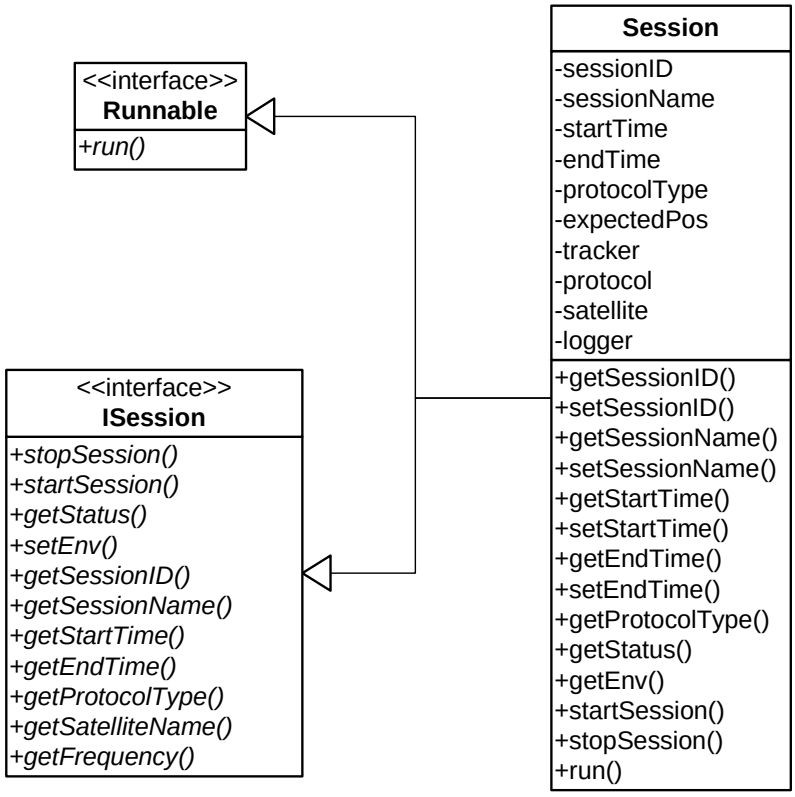

Figure 4.9: Class Diagram of The Session

4.3.1 Session Controller

How does the Ground Station Manager(GSM) run sessions? To decouple the relation between the Ground Station Manager and the Session, the ground station control system uses the factory pattern to run sessions. The Session Controller should completely abstract the initialization and execution of the session from the Ground Station Manager. Figure 4.10 shows the basic block diagram of the GSM, the session and the session controller.

Figure 4.10: Block Diagram of The GSM, Session and Session Controller

When a session is ready to run, the GSM starts a session controller thread to run this session. Then, the session controller thread calculates the duration of the session based on the start time and the end time of the session. When all the preparation work have been done, the session controller starts up the session thread. The session thread should run the corresponding protocol to set up the communication link, and start the tracker to keep contact with the satellite.

4.3.2 Session Scenario Analysis

The execution of a session could be running in the session controller thread(two threads scenario) or running as an independent thread (three threads scenario). The GSM and the session controller threads are running in the first scenario. The second scenario hosts three threads including the GSM, session controller and session threads. In the following, we will analyze the two scenarios and compare them.

Two Threads Scenario

When the start time of a session is approaching the current time, the session dispatcher tell the GSM to start this session immediately. Then, the GSM initializes and runs a new session controller thread to do the rest of control work for the session. The session controller starts a timer to monitor the session time slice, and call the run method in the session object(not starting a session thread). If the session does not finish its tasks within its duration and the timer times out, the timer notifies the session controller that the time is up. The session controller should then terminate the session, and also stop the protocol and the tracker. Figure 4.11 shows the sequence diagram of two threads scenario.

Three Threads Scenario

When the start time of a session is approaching the current time, the session dispatcher tells the GSM to start this session immediately. Having the same beginning as the two thread scenario, the GSM starts a session controller thread. The difference is that the session controller thread starts the session as a thread

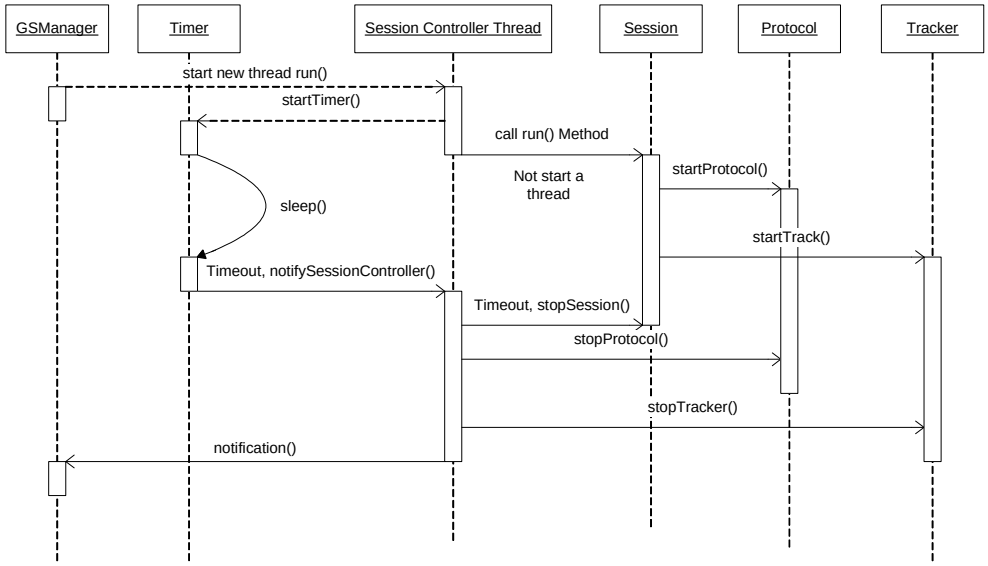

Figure 4.11: Sequence Diagram of Two Threads Scenario(Time Out)

and then sleeps for the running duration of the session after starting a session thread.

If the session finishes its tasks in the time slice, then the session wakes up the sleeping session controller thread. The session controller stops the protocol and tracker, notifies the GSM that the session has finished. Then the GSM can remove this session from the session queue. The sequence diagram of three thread scenario(normal) is shown as Figure 4.12.

If the session cannot finish its tasks within the duration, the time is up and the session controller wakes up by itself. The session controller forces the termination of the session thread. After the session has been terminated by the session controller, the session controller stops the protocol and the tracker. When all the clean-up work have been done, the session controller tells the GSM that the session has done. Figure 4.13 shows the sequence diagram of three threads scenario(time out).

Considering the two scenarios, the three threads solution provides clearer and wiser way to achieve our goal. Because we have the GSM, session controller and session three threads running in the system, they have their own responsibilities and tasks. GSM is only in charge of starting the session controller thread, and leaves the session management work to the session controller. The purpose of

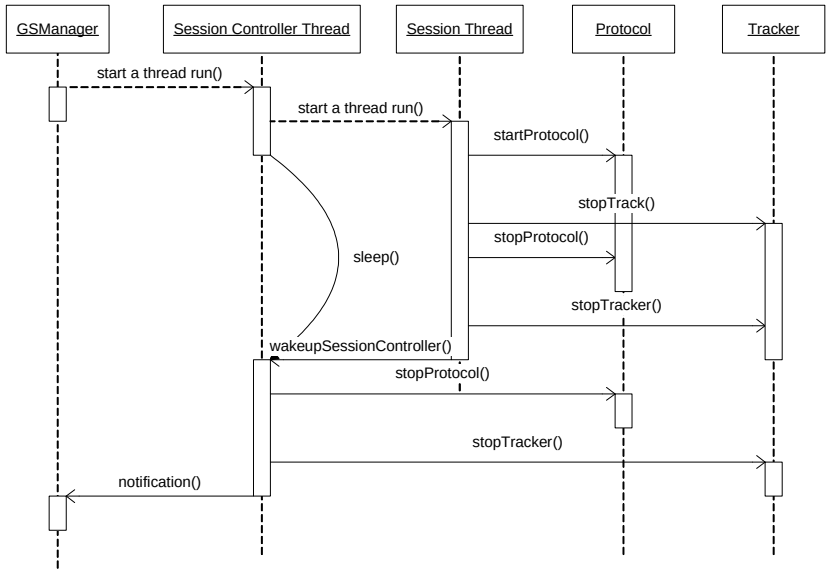

Figure 4.12: Sequence Diagram of Three Threads Scenario(Normal)

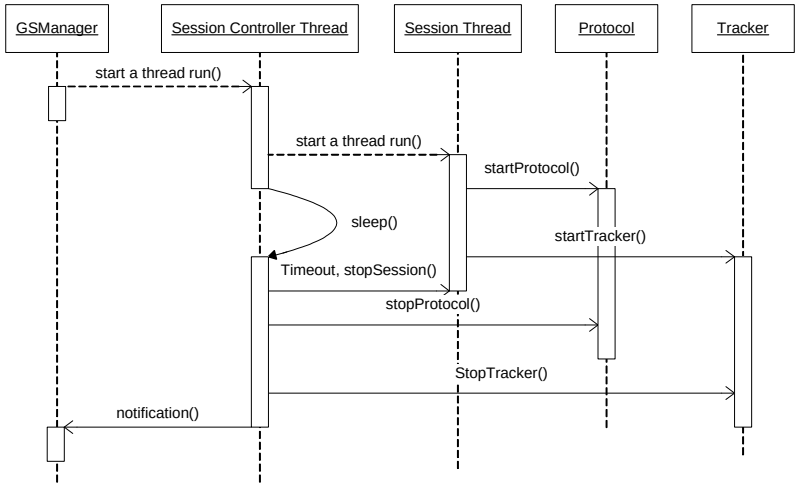

Figure 4.13: Sequence Diagram of Three Threads Scenario(Time Out)

the session controller is to start up the session thread, terminate the session thread in case the session cannot finish its tasks within the certain duration and stop the protocol and the tracker. In this solution, we separate the session control work from GSM and let the session controller do it. Therefore, the GSM

is more flexible to complete the other assignments.

4.4 Implementation of Protocol

At the scope of the project, we describe that the control system should be able to accommodate to the different communication protocols used by different satellites including AX-25 based ones. We spent some time to do the research of the AX-25 stack in the beginning stage, but we could not find a practical solution for the AX-25 protocol because of the time limit. Still, the idea of the protocol implementation has been delineated.

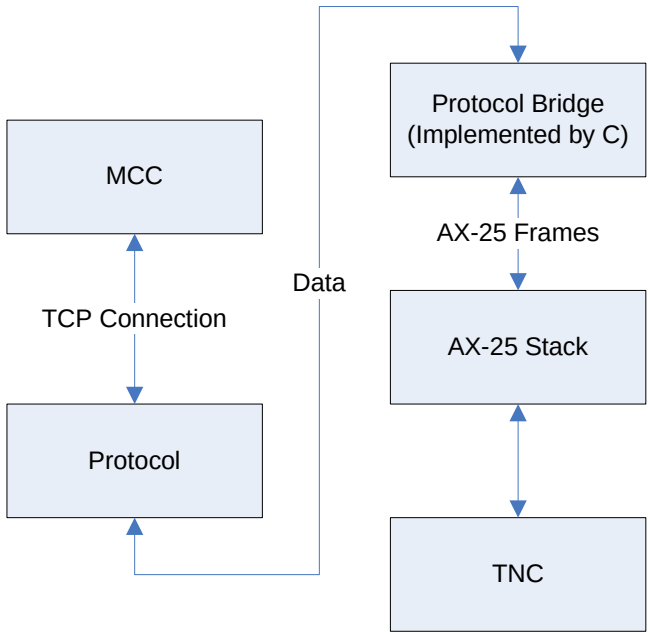

Figure 4.14: Block Diagram of The AX-25 Protocol

The implementation of the protocol should be like Figure 4.14. The purpose of the protocol component is to set up the TCP-connection with the mission control center(MCC) and also to talk with the protocol bridge. The protocol bridge is a class which only contains the native interfaces implemented by C language. The protocol bridge is used to construct the AX-25 frames and send them to the AX-25 stack which is connected with terminal node controller(TNC) via a serial line.

The session or the ground station manager(GSM) can use the method *startProtocol()* to start a protocol service. The method *stopProtocol()* provides the means to kill the protocol thread if fatal errors occur in the protocol thread. The status of the protocol can be fetched via the method *getStatus()* if the GSM or session likes to know the current status of the protocol thread. The protocol bridge has been implemented in a prototype version to simulate the scenario for demonstration purpose. It implements two methods which can read and write files in the local file system. The class diagram of the protocol is shown as Figure 4.15.

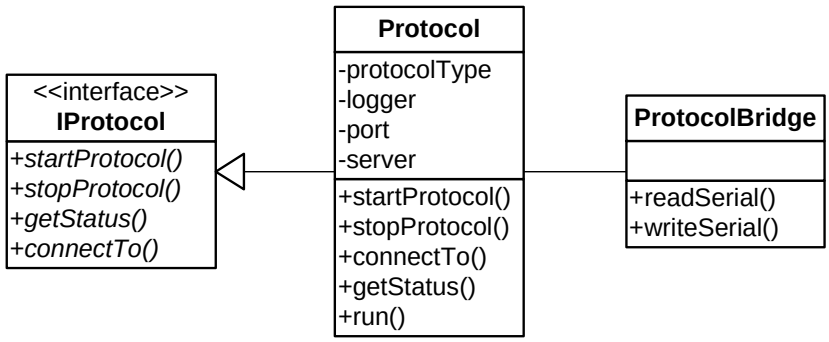

Figure 4.15: Class Diagram of The Protocol

4.5 Implementation of Tracker

The tracking system has been done in a three weeks project by three students. A radio driver for YAESU FT-847 and a motor driver for YAESU GS-232A which are used to control the current ground station hardware. They use the Sputnik [25] Java library to predict the satellite positions. Figure 4.16 shows the full picture of the tracking system.

When the ground station manager(GSM) or the session calls the method *initTrack()* to prepare the tracker for a satellite pass, the tracker fetches the initial position values by querying the Predict object and then pass on the values to the radio and motor drivers. After making sure that the antenna is pointed to the first visible position of the satellite, the method *startTrack()* can be used to start tracking of the satellite and put the drivers in tracking mode. The tracker repeatedly inquires the predict object to obtain the current position and velocity of the satellite, and adjust the corresponding antenna and radio parameter values by the motor and radio drivers. The class diagram of the tracker is shown as Figure 4.17.

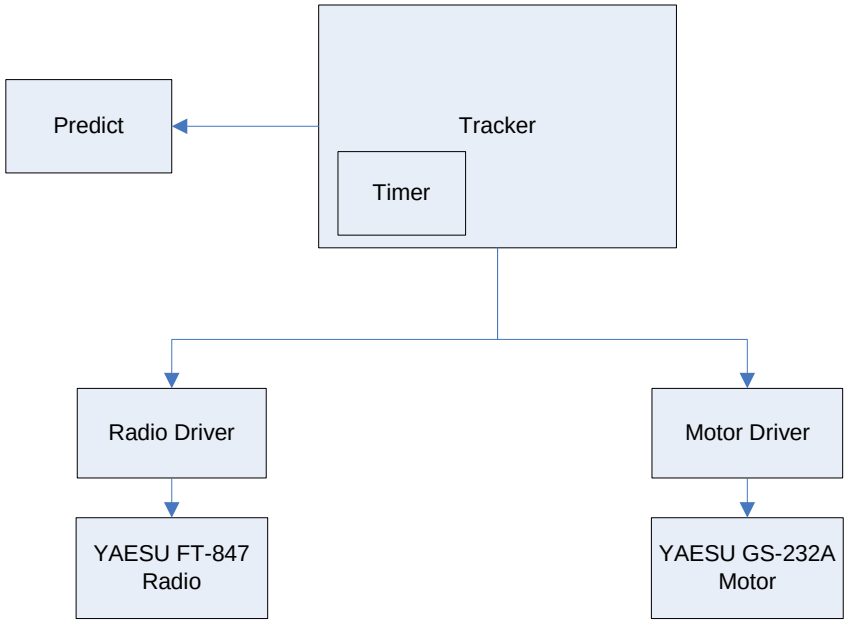

Figure 4.16: Overview picture of The Tracking System

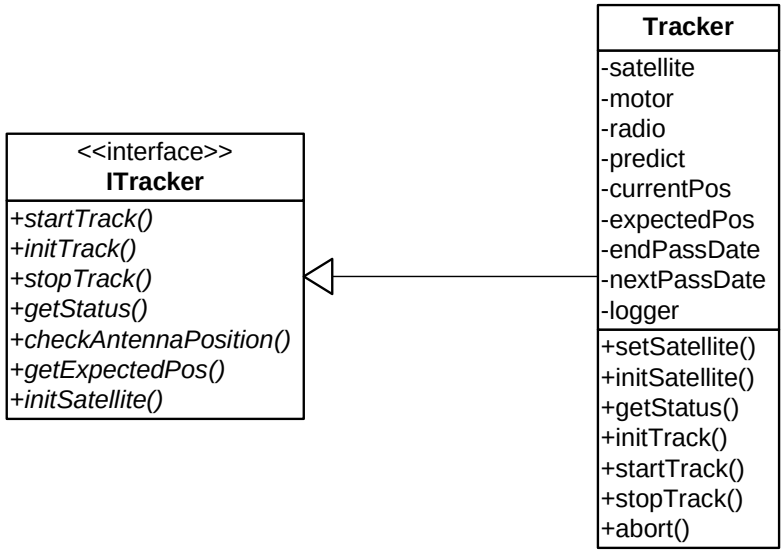

Figure 4.17: Class Diagram of The Tracker

4.6 Implementation of Common Function

The logger and the status objects are used in every component of the ground station control system. They are common functions to provide the logging and status construction for every module in the system.

4.6.1 The Logger

As we mention in Chapter 3, the logger defines 5 levels of importance including FATAL, ERROR, WARN, INFO, DEBUG represented by 5 integers. The logger can be set to only report messages of a given level in a given module via the method *setLevel()*. The implementation of logger consists of a abstract class *ILog*, and the other subclasses which extend the super class *ILog* and present to log the messages into the different storages such as console or file. The source code for the logger interface refers to List B.12. The class diagram of the logger is shown as Figure 4.18.

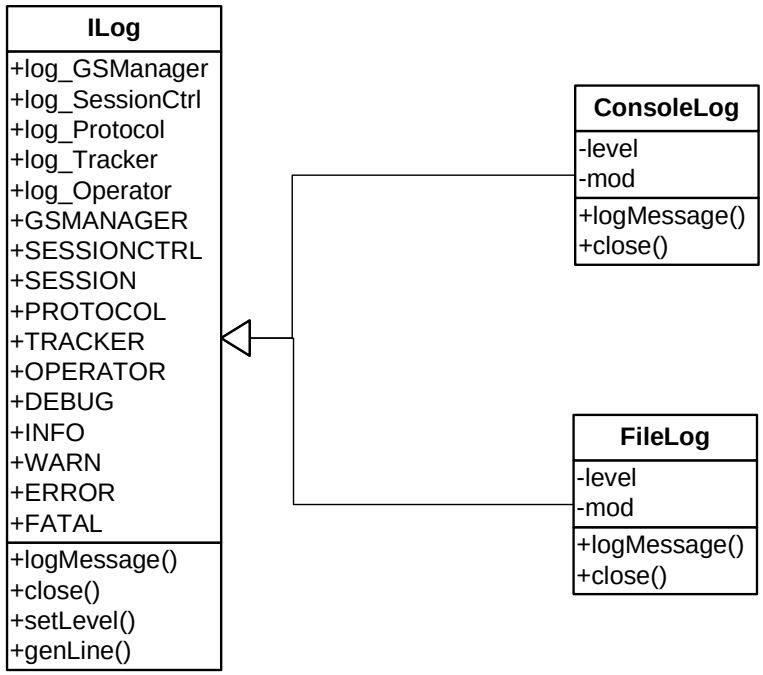

Figure 4.18: Class Diagram of The Logger

4.6.2 The Status Object

The status object is designed to contain all the status information of the modules, and can be reused by different modules in the control system. So a flexible design of the status object can increase the modularity of the system.

The Path

The purpose of path class is to provide the explicit path for the parameter values of the status object. For instance, if the tracker specifies the path as "Freq" for the frequency value in the tracker and transfer the status to the ground station manager(GSM), the GSM add a prefix "tracker" in the path of the status object and pass this object to the manual operattion. After the manual operation checks the path of status object as "Tracker.Freq", it knows this parameter value is from the frequency of the tracker module. Such path class is mainly used in passing the parameter values of status object in the different modules of the system. Figure 4.19 illustrates the example.

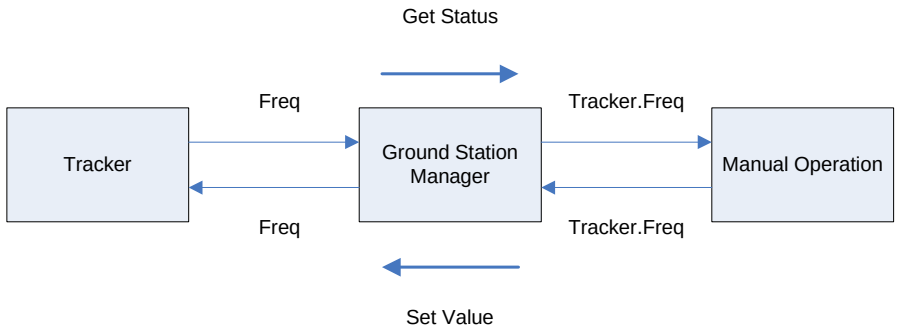

Figure 4.19: An Example of The Path Usage

The path class has three main methods: *addPrefix()* to add a prefix to indicate the source path of the values, *getPrefix()* to obtain the prefix of the values, and *deletePrefix()* to delete the prefix after passing the request to the right module in the system. Figure 4.20 shows the class diagram of path.

The Parameter Value

The parameter value is a super class which is inherited by four subclasses: integer parameter value(IntParaVal), numerical parameter value(NumParaVal), string parameter value(StrParaVal) and boolean parameter value(BoolParaVal). The parameter value class is used to store the status values of the modules. Be-

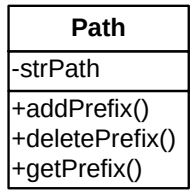

Figure 4.20: Class Diagram of The Path

cause the values of status could be various data types, we provide four different subclasses to match the data types. Indeed, the status object includes a list of the parameter value classes which contain the values of the different parameters in the modules. Figure 4.21 shows the class diagram of the parameter value.

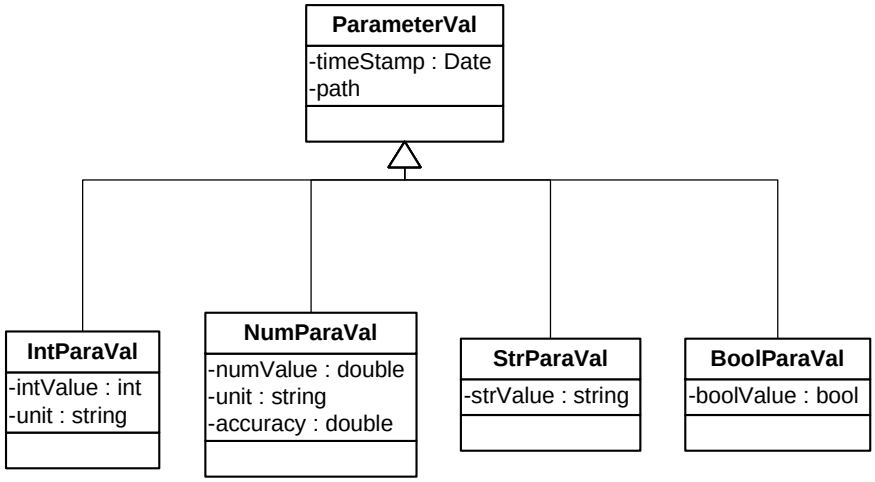

Figure 4.21: Class Diagram of Parameter Value

4.7 Implementation of Clients

The implementation of clients includes the mission control center(MCC) and manual operation. Since we have decided to provide the remote access for the clients, the ground station control system has to implement the remote services for the clients. There are several technologies which can be used in the control system such as TCP, Web services [32] and Java RMI [21]. The control system can run as a TCP server and receives requests from clients via the socket. The

web services define a open standard (XML, SOAP, etc.) based Web applications that interact with other web applications for the purpose of exchanging data. Because the web services provide the remote access over the web, the control system also can use the web services to reach our goal. Since the implementation of the control system should be done in Java, naturally, we decide to adopt Java Remote Method Invocation(RMI) to provide the remote access of the control system. All the operation of clients can be done remotely via the Java Remote Method Invocation(RMI) [21] technology.

4.7.1 RMI and RMI Security Manager

Java RMI is a mechanism which supports one object to invoke a method on another object located remotely, sharing resources and process load across systems. We choose RMI to implement the remote access because the RMI provides a easy solution which can dynamically load the class in a different address space. Figure 4.22 shows the RMI architecture layers [26].

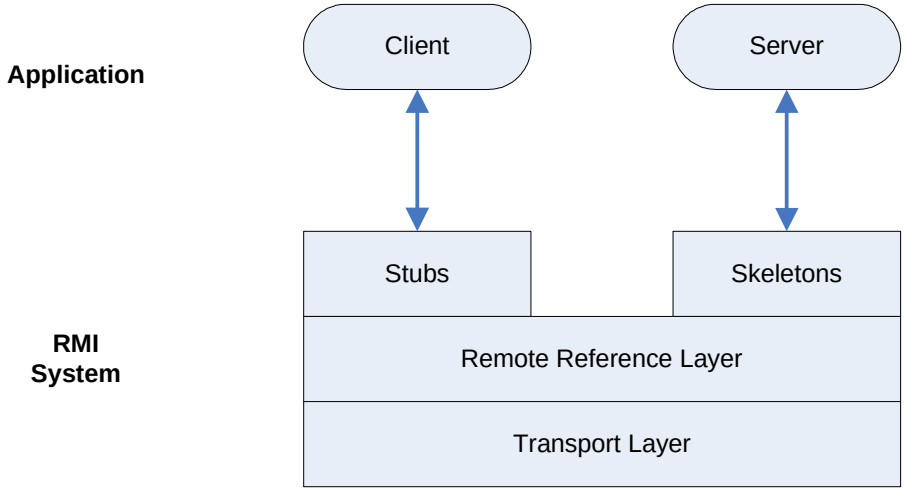

Figure 4.22: Architecture RMI Layers

Since the ground station control system provides the remote access for the clients, the security issues should be considered thoroughly in the implementation. When developing the control system with RMI, some actions may be restricted in order not to be performed by remotely loaded classes. Otherwise, the potential unsecured code could access the control system resources. The *java.rmi* package provides a default security manager implementation that we can install with the following code:

if(System.getSecurityManager() == null) {
System.setSecurityManager(new RMISecurityManager()); }

We also can create our own security manager implementation to enforce custom security policies. In Java 2, the RMI security manager class requires that we have to specify a security police file at runtime by defining a value for the *java.security.policy* property:

java -Djava.security.policy=policyfilename

If we do not specify the security policy file, in the default, Java will search for the system wide policy file in the JAVA_HOME directory where the java has been installed, or look for the user defined policy file in the user home directory. In the implementation of the ground station control system, in order to deduct the workload of the implementation, we assume the ground station trusts all the connections from the mission control center(MCC) and the manual operation. Therefore, the following simple policy file has been made to grant full access permission to everyone.

grant { permission java.security.AllPermission; };

Alternatively, an elaborate RMI security manager can be make by defining the security policy file. The article [30] explains the policy implementation and policy file syntax if you want to know more about the policy file. Here is an example which only allows the connection from the client with IP "192.38.79.147" and the port "4545".

grant{
permission java.net.SocketPermission "192.38.79.147:4545", "connect, resolve";
permission java.net.SocketPermission "192.38.79.147:4545", "accept, resolve";
};

4.7.2 Mission Control Center

Basically, the MCC creates the session object and sends it to the ground station control system. We discuss the interface design of the MCC which is implemented by the ground station manager(GSM). All the interface implementations are done to support the remote access by Java RMI. The GSM is running as a

RMI server (to register its services in an RMI registry), and the MCC invokes the methods in the remote GSM object. In order to be copied from one space address and another, all the classes must be Serializable classes.

4.7.3 Manual Operation

The Mission Control Center constructs the session objects containing the pass schedules and sends them to the control system which can automatically run the requested sessions to accomplish the tasks. However, manual control is necessary for some circumstances when the operators wants to start the tracking system and protocol, to monitor the system status manually. Of course, the operator should be able to access the tracker and the protocol modules directly in theory. But we choose the solution for the manual operation which is to access all the ground station services via the ground station manager(GSM). So the GSM is the key component for the manual control. Figure 4.23 shows the normal system sequence diagram for manual control and how the Ground Station Manager controls the other components work together.

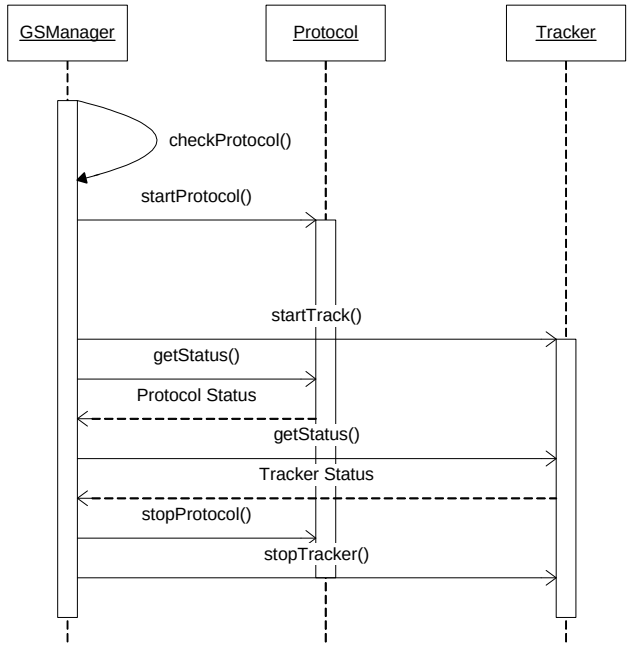

Figure 4.23: Sequence Diagram of Manual Operation

We have implemented a status monitor class to poll the status of the protocol

and tracker periodically to obtain the up to date status of the control system. The status monitor is running as an independent thread to ask the manual operation to refresh the status every 10 seconds. When the manual operation receives the refresh status request from the status monitor, the manual operation sends the query to the ground station manager(GSM). The GSM responses the request to return the tracker and protocol's current status back to the manual operation. The block diagram of the status monitor is shown as Figure 4.24. The manual operation also uses the Java RMI to set up the communication with the remote control system.

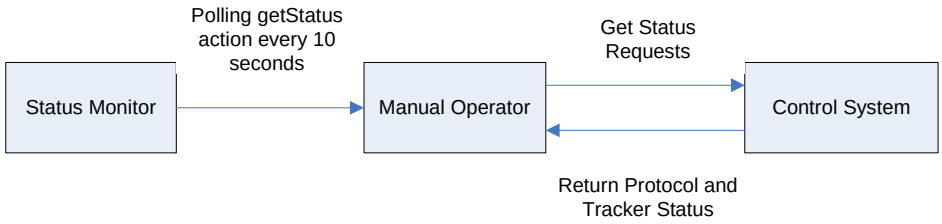

Figure 4.24: Block Diagram of The Status Monitor

4.8 Concurrent Issues

Since the control system is a multi-thread system, the concurrent issues must be thoroughly considered to avoid the thread conflict. Java uses the *synchronized* keyword to indicate that only one thread at a time can be executing in this or any other synchronized method of the object representing the monitor [34].

The session queue is a shared resource for the ground station manager(GSM) and the mission control center(MCC). In order to avoid the conflicting access to the session queue, we use the *synchronized* keyword for the publicly accessible methods to protect the session queue such as *insertSession()* and *removeSession()*.

Because we only allow one session running in the control system at the same time, there will be three threads: the GSM, Session controller and session threads in the system. The session controller thread is started up by the GSM thread. The session controller thread starts up a session thread to do the satellite tracking. Afterwards the session controller calls *wait()* to block and leave the monitor until a *notify()* from the session thread or wakes up by itself when time is out. After all tasks have been done, both session controller and session threads will finish.

For the concurrent issues in the tracker, because the tracker has the Timer threads which run the tasks to control the antenna and radio in the ground station, we use the *synchronized* keyword for the tracker object to protect the threads in the tracker.

Chapter 5

Testing

Most software produced today is modular. System testing is a necessary phase of the software testing in which software engineers see if there are any communications flaws–either not passing information, or passing incorrect information–between modules in the system [27]. In general, software testing and evaluation make sure that the system runs as expected without failures.

5.1 Unit Testing

Unit testing validates that a particular module of the system conforms to its specification and correctly performs all its required functions. It produces tests for the behavior of components of the system to ensure their correct behavior prior to the system integration. In this section, we will test the individual module in the system and present the test strategy for these modules. A practical way to do unit testing is to design test cases (Appendix C) for all functions and methods of the modules.

5.1.1 Session Queue

The unit testing of the session queue should focus on the two main methods *insertQueue(Session)* and *removeSession(SessionName)*.

insertQueue(Session)

- There may be three reasons why a new session cannot be successfully inserted into the session queue. The first reason is that the start time of new session is conflicting with the current time. The second reason is that the time slice of new session overlaps with some sessions in the session queue. The third reason is that the new session name is a duplicate of all in the session queue because we assume that the session name is a unique identification of the sessions.

  Considering the possibilities of the conflict with the current time, we design two test cases for methods *insertQueue(Session)*: the start time of the new session is earlier than the current time and the start time of the new session is slightly later than the current time(less than 5 minutes). These two test cases fail to insert the new session into the session queue as expected in the testing.

  For the overlap problems, we design several time slices for the new session which overlap with the existing sessions in the session queue. Figure 5.1 shows four different test cases for the overlap problems. The length of the box indicates the time length of the session. All the test cases fail to insert the new session into the session queue.

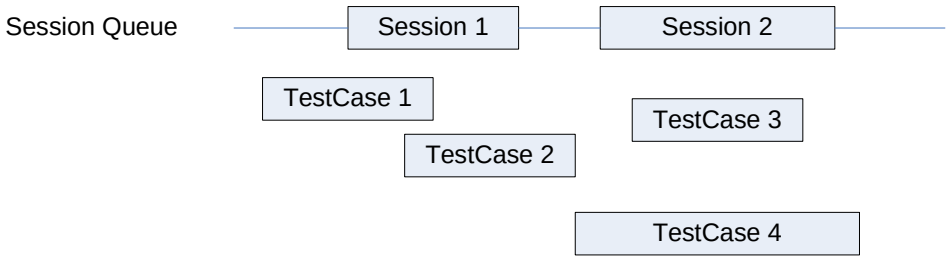

Figure 5.1: Test Cases for Overlap Problem in the Session Queue

For the duplicate name problem, the testing is fairly easy to achieve. We just simply send a new session which has the same session name as an existing session in the session queue. The result of the testing trivially fails to send the new session to the session queue as expected and throws a session queue exception to warn the users of this failure.

removeSession(SessionName)

- The unit testing of the method *removeSession(SessionName)* is much simpler than the *insertQueue(Session)* since we only have two possibilities. If the session name exists in the session queue, then the removing operation should be successful. If the session name is not in the session queue, the result of the testing should throw a session queue exception failure message. The real testing results are exactly matching the expectation as we analyzed above.

The other methods like *printSessionQueue()* are obviously functional since the codes in the methods are fairly easy to understand. We do not have to design particular test cases for these methods.

5.1.2 Ground Station Manager

The unit testing of ground station manager(GSM) is rather complicated compared to the unit testing of the session queue because the GSM is the main entry of the ground station control system. Some methods of GSM are strongly related to other modules which means that these methods intend to be tested in the integration testing phase. Therefore, we are going to go through the main functions like *registry of GSM services*, *start session dispatcher*, *add session*, *remove session*, and *start session controller* in the unit testing of the GSM.

Registry of GSM Services

- Because the GSM provides some services which can be accessed remotely by the mission control center(MCC) and the manual operation, GSM should run as an RMI server and register its services in the RMI registry. According the test cases, the test results show that the GSM successfully binds with a particular port in the local address and register "GSManager" in the name space of RMI registry. These functions have been fulfilled in the GSM.

Start Session Dispatcher

- The purpose of the session dispatcher is to poll the session queue to check whether the first session is approaching the start time. The session dispatcher should be initialized by the GSM and run as an independent thread

to poll the session queue periodically(30 seconds). The test results show that the GSM successfully starts up the session dispatcher. These functions exactly match our design.

Add Session

- The method *addSession(Session)* is defined in the mission control center(MCC) interface and the GSM has implemented it. The MCC can invoke this method to remotely insert a new prepared session into the session queue. The test results show that the new session can be added successfully via the method *addSession(Session)* if this session is correct and does not conflict with other sessions in the session queue.

Remove Session

- The method *removeSession(SessionName)* is also defined in the MCC interface which has been implemented by GSM. We designed two test cases for the method *removeSession(SessionName)*. One of the test cases is to construct a session name which does not exist in the session queue. This test case should fail to remove the session since this session is not in the queue. Another test case is to set an existing session name as the *removeSession()* parameter. This test case should remove the session successfully from the session queue. The test results show that the method *removeSession(SessionName)* works in both test cases.

Start Session Controller

- In the implementation of the GSM, the GSM is responsible for running the session controller which controls the session object. In the unit testing of the GSM, the GSM successfully initializes the session controller object and starts it up as a thread to manage the session. This function is fairly easy to test and we are going to test the interaction between the modules in Section 5.2.

5.1.3 Protocol

Because we have not implemented a real protocol like an AX-25 based protocol in the control system, the testing of starting and stopping the protocol thread is

simple. Therefore, We do not have to design the test cases for these functions. The unit testing of the protocol focuses on the three part of functions: *TCP connection with MCC*, *get protocol status*, and *external program*.

TCP Connection with MCC

- Since the protocol thread has to set up the TCP communication connection with the MCC, the testing of this function should make sure that the protocol thread can connect with the MCC and send some messages to the MCC. The testing results show that the protocol can connect with the MCC and also that the MCC can receive the messages from the protocol. These functions has been approved as expected in the unit testing.

Get Protocol Status

- The method *getStatus()* returns a list of protocol parameter values. The unit testing should check this function returns the array list which contains the correct values of the protocol parameters. The testing results indicate that the method can return the right values of the parameter list.

External Program

- The external program of the protocol is implemented using the C language. The interface has been defined in the protocol bridge class which provides the two methods to read and write files. The test result demonstrates the external program can read the content of a file and also write some messages back to a file.

5.1.4 Session Controller

The unit testing of the session controller mainly has two part of testing. The session controller starts up the session thread and terminates the session thread. The results of the test cases show that the two functions run as expected.

5.1.5 Session

Because the session object is a pass schedule which is received from the mission control center, the testing of the session class should concentrate the session in-

terface. We have defined some methods in the session interface like the methods *stopSession()* and *startSession()*. Through the test cases, the functions can be successfully accomplished in the session.

5.1.6 Tracker

The unit testing of the tracker is fairly simple because the tracker has been done by other students and they already made thorough testing of the tracker. We made some interfaces for the tracker and only have to test these interfaces. The method *initSatellite()*, *initTrack()*, *checkAntennaPosition()* and *startTrack()* can be functionally used by the GSM or the session and start a tracking system by them. The method *stopTrack()* also can terminate the tracking system in the test cases. Through the testing, the method *getStatus()* can successfully return the tracker status to the GSM.

5.1.7 Mission Control Center

The main purpose of the mission control center(MCC) is to test the whole ground station control system. The test cases have to set up the TCP server, receive the connection from clients, lookup the RMI registry to find the registered services of the GSM and send the new session to the GSM. All these functions have been tested thoroughly by using the test cases.

5.1.8 Manual Operation

The test cases of the manual operation specially design for the services provided by the GSM. Some test cases should start and stop the protocol manually. Some test cases should start and stop the tracker. The operator gets the status of the protocol and the tracker using the test cases. Other test cases are to kill the current running session thread. The testing results show that all the functions can be done successfully in the manual operation.

5.2 Integration Testing

The integration testing is a phase in which individual system modules are combined together and tested as a group. The purpose of integration testing is to

verify that the functionalities, reliabilities, and performance of the system are as expected from the requirement specification of the system. The integration testing is also through test cases using black box testing.

5.2.1 Automation Session Testing

The automation session testing of the control system is to generate a session object which can be automatically executed by the control system. A mission control center(MCC) must be made to simulate the real MCC system, but only provides some simple tasks for testing ground station control system. Figure 5.2 shows the possible states of a session object. In generally, the life circle of a session has four states: generated by the MCC, waiting in the session queue, running as a thread and terminated normally or by other modules. In the "Generated" state, the session may turn into "Waiting" if added into the session queue, or shift to the final state if cancelled. For the "Waiting" status, the session could have two possible changes: turn to "Running" state if the start time is close to the current time and to the final state if removed by the requests from the MCC. If the session is in the "Running" state, the session may finish all its tasks to quit to the "Terminated" state, or be killed by the operator and turn to the "Terminated" state. In the end, the "Terminated" state of the session always finishes and turn to the final state.

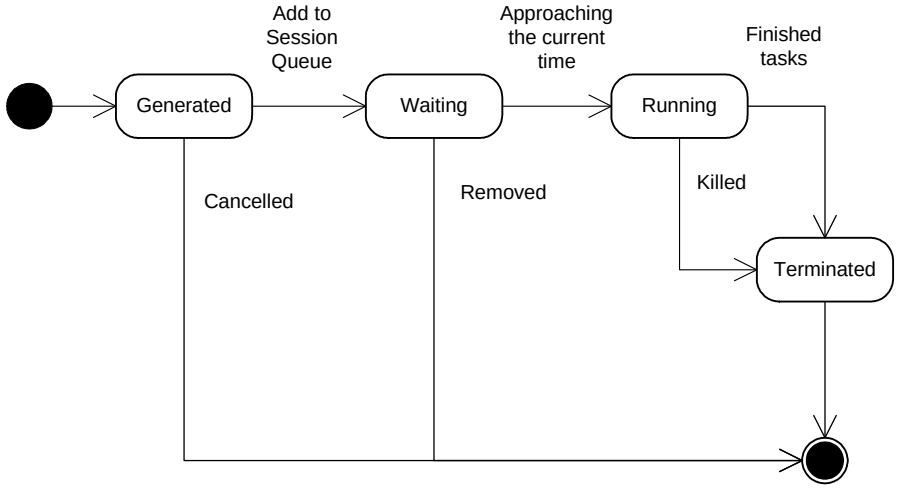

Figure 5.2: State Diagram of The Session

From Figure 5.2, the test cases should go through all the possible routes of the session from the initial state to the final state.

- The MCC generates a new session as a pass schedule. The session object might be cancelled by some accidents like a system crash and turn into the final state.

- If the session has been added into the session queue as the state "Wait" after generated, the MCC requests the GSM to remove this session from the session queue. The session state switches to the final state after being removed from the session queue. To test this possibility, the MCC generates a session and sends it to the GSM to add to the session queue. Before the session runs, the MCC invokes the remove session method to request the GSM to delete this session from the session queue.

- If the session is still in the queue and the start time is slightly later than the current time, the session dispatcher tells the GSM that the session is ready to run. The GSM starts up a session controller thread to manage the session. The session controller thread starts up the session thread to run the pass schedule. If the session cannot finish its tasks within its duration, the session controller thread wakes up and terminates the session thread to the final state. If the session finishes its tasks within its duration, the session wakes up the session controller thread and turns into the final state. If the session is running and the GSM receives a kill session command from the operator, the session is terminated to the final state. All these possibilities must go through the testing in the test cases.

5.2.2 Manual Control Testing

The integration testing of the manual control should concentrate on testing all the functionalities which are provided by the manual operation GUI interface. The table 5.1 shows all possible test cases which have to be tested in the manual control.

If the session is currently running in the control system, the manual control of the protocol and the tracker should be forbidden and throw some error messages. But the monitor functions of the manual operation should be allowed to monitor the status of the protocol and the tracker. Also, the kill session service should be functional when the session is running.

On the other hand, the operator can manually start up the control system services like the protocol and the tracker, and also be able to stop these services if they are running in the control system. Obviously, the monitor functions are

	If session is not running	If session is running
Start Protocol	Allow	Do Not Allow
Stop Protocol	Allow	Do Not Allow
Start Tracker	Allow	Do Not Allow
Stop Tracker	Allow	Do Not Allow
Get Protocol Status	Allow	Allow
Get Tracker Status	Allow	Allow
Kill Session	Throw Exception	Allow

Table 5.1: Manual Control Testing

allowed in this scenario. Because the control system is in the manual control mode, the kill session function should be disabled.

5.3 On Site Testing

The on site testing has taken place in the DTUsat ground station in Building 348. There are the antenna and radio which can communicate with the satellites when they pass by the ground station. The present hardware in the DTUsat ground station laboratory are a YAESU GS232a serial controller hooked up to a YAESU G-5500 motor controller and a YAESU FT-847 radio. The ground station control system has been tested to track some satellites like CUTE-1(Cubical TITech Engineering Satellite) [28].

5.3.1 Tracking CUTE-1

The start time and end time of session can be known by using a satellite tracking software called NOVA [29]. The mission control center(MCC) set up the start time, end time, satellite name and frequency for the new session. Then MCC sends the new session to the GSM and add it into the session queue to wait for the execution. When the pass time is approaching, the session automatically run to track the satellite pass by the control system. On the ground station site, The antenna is moving according the track of the satellite after the session starts up the tracker. The radio YAESU FT-847 can do the Doppler correction to alter the frequency of the radio in the CW mode. The Morse signals can be clearly heard from the radio device.

Chapter 6

Conclusion

This chapter concludes the thesis work related to the project of *Satellite Ground Station Control System*. In the conclusion, we first evaluate to which extent the control system has fulfilled the system specification defined in Chapter 1 and summarize the main contribution of the project. Then we describe some interesting problems which we encountered but did not have time to implement during the project.

6.1 Evaluation of Control System

Through the unit testing, integration testing and on site testing in Chapter 5, all the functionalities of the ground station control system have been thoroughly tested. The automation of control system executes successfully as we can see from the test cases. The manual control of the ground station also achieves our goal which has been defined in the previous chapters.

The automation of the ground station control system provides a simple and flexible solution to be used for small student satellites. The major services of the ground station like *satellite tracking*, *data and protocol handling* and *logging* have been accomplished in the implementation phase of the project. The ground station control system is able to control the antenna and radio to

track the satellite during the pass. The system also implements a prototype of the protocols which are used to set up the communication link with the satellite. A rudimentary external satellite mission control center has been implemented to set up the session which can automatically control the satellite passes in the control system.

Except the automation of the control system, a flexible alternative *manual operation* has been accomplished to allow for manual control of the system. The operator can control the antenna and radio via the tracker, monitor all the component status of the system, and be able to recover the system in case of the errors. The remote access and automation of the control system frees the operators from physically being at the ground station and give them more flexible work space.

6.2 Further Work

Protocol

In the further work of the ground station control system, the more elaborate work should be done with the protocol component of the system. The protocol component should be able to accommodate the different communication protocols used by satellites including AX-25 based ones in the system. The control system should start the corresponding protocol according the protocol type which has been defined in the session object. The protocol can set up the network connection with the mission control center for transferring the packages, and invoke an external program to access the low level protocol stack via serial connection with the terminal node controller(TNC). The external program used to construct the protocol frames will be the primary task in the further work.

RMI Security Policy

For the remote access, the more sophisticated RMI security policy file has to be defined to deny the accesses from untrusted users and computers in the control system.

Web Remote Access

The remote control over the web could be the part of the further work for the control system. Using the web access, the operator can access the control system and monitor the system running status almost everywhere with the internet connection. The security issues become more important if the control system

provides the web access. The user authentication and encrypted data exchange channel have to be thoroughly considered in the system.

Appendix A

Class Diagram

A.1 Common Function

A.1.1 Parameter Value

Refer Figure 4.21 in Chapter 4.

A.1.2 Path

Refer Figure 4.20 in Chapter 4.

A.2 Ground Station

A.2.1 Ground Station Manager

Refer Figure 4.7 in Chapter 4.

A.2.2 Log

Refer Figure 4.18 in Chapter 4.

A.2.3 Protocol

Refer Figure 4.15 in Chapter 4.

A.2.4 Satellite

Position

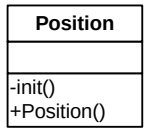

Figure A.1: Class Diagram of Position

Position Velocity

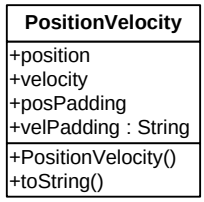

Figure A.2: Class Diagram of Position Velocity

Predict

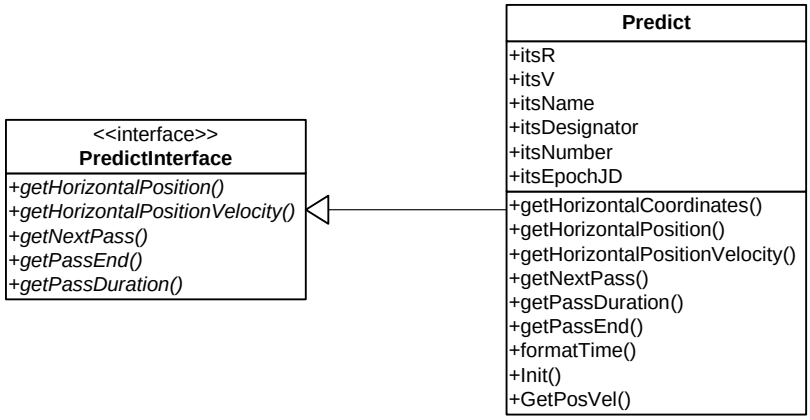

Figure A.3: Class Diagram of Predict

Predict Simulator

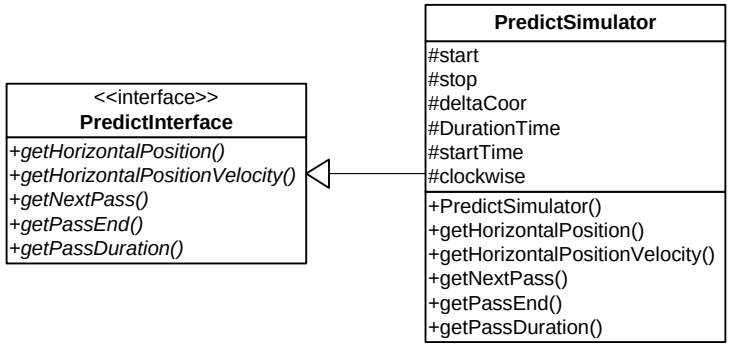

Figure A.4: Class Diagram of Predict Simulator

Satellite

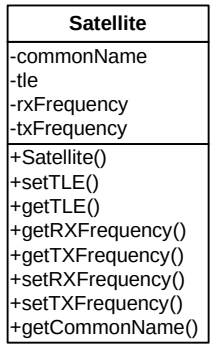

Figure A.5: Class Diagram of Satellite

Satellite Reader

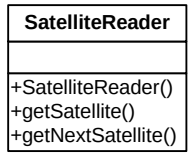

Figure A.6: Class Diagram of Satellite Reader

Station

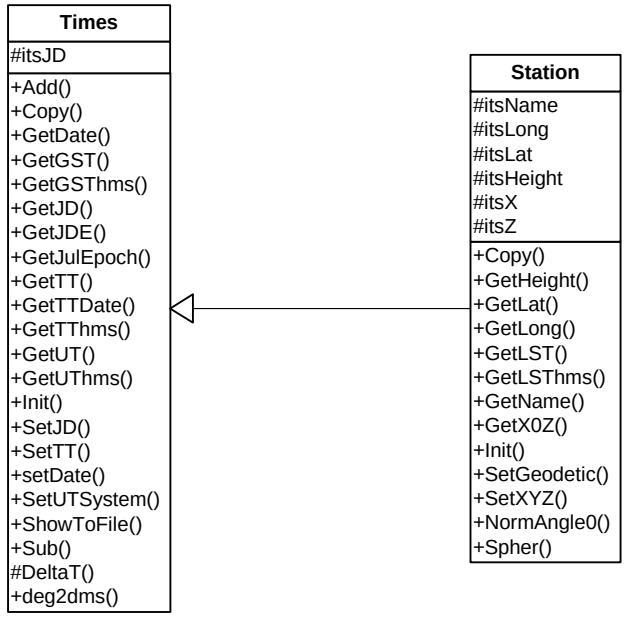

Figure A.7: Class Diagram of Station

Velocity

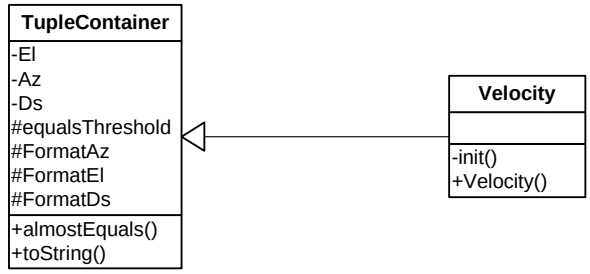

Figure A.8: Class Diagram of Velocity

TLE

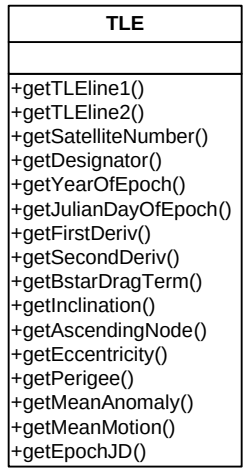

Figure A.9: Class Diagram of TLE

A.2.5 Session

Session

Refer Figure 4.9 in Chapter 4.

Session Controller

Figure A.10: Class Diagram of Session Controller

Session Exception

Figure A.11: Class Diagram of Session Exception

Session Queue

Refer Figure 4.8 in Chapter 4.

A.2.6 Tracker

Tracker

Refer Figure 4.17 in Chapter 4.

Motor Driver GS232a

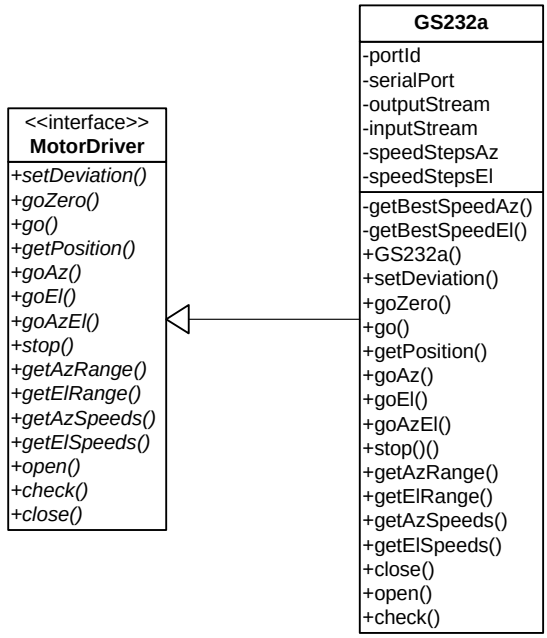

Figure A.12: Class Diagram of Motor Driver GS232a

Motor Driver GS232a Position Transformer

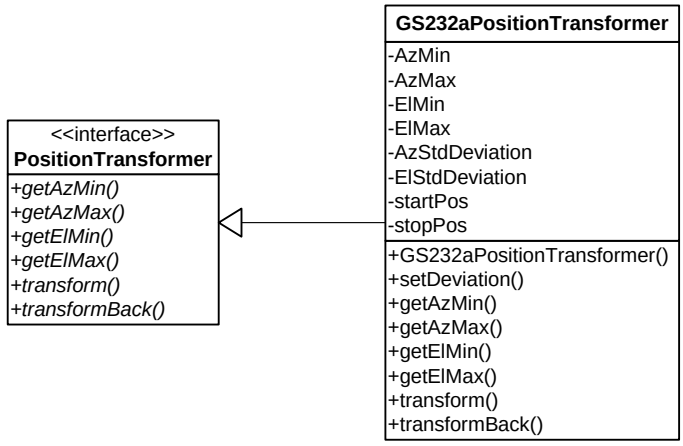

Figure A.13: Class Diagram of Motor Driver GS232a Position Transformer

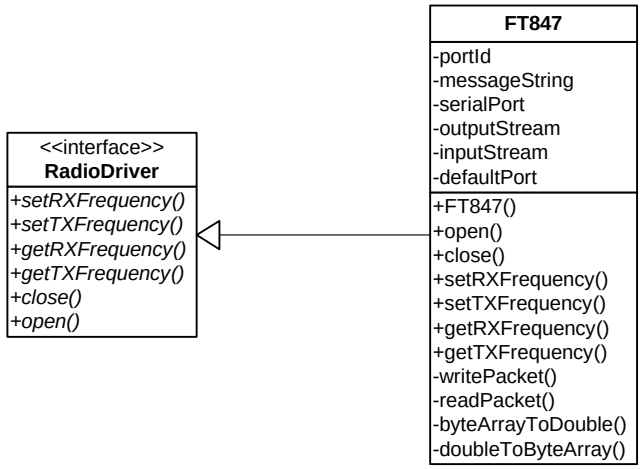

Figure A.14: Class Diagram of Radio Driver FT847

Appendix B

Source Code

B.1 Common Function

B.1.1 Parameter Value

Listing B.1: Bool Parameter Value (BoolParaVal.java)

```
package common.parameterVal;

/*
 * Created on Aug 15, 2005 @author Yu
 */

public class BoolParaVal extends ParameterVal {

        public boolean boolValue;

}
```

Listing B.2: Integer Parameter Value (IntParaVal.java)

```
package common.parameterVal;

/*
 * Created on Aug 15, 2005 @author Yu
 */
public class IntParaVal extends ParameterVal {
```

```
        public int intValue;

        public String unit;

}
```

Listing B.3: Numerical Parameter Value (NumParaVal.java)

```
package common.parameterVal;

/*
 * Created on Aug 15, 2005 @author Yu
 */

public class NumParaVal extends ParameterVal {

        public double numValue;

        public String unit;

        public double accuracy;

}
```

Listing B.4: String Parameter Value (StrParaVal.java)

```
package common.parameterVal;

/*
 * Created on Aug 15, 2005 @author Yu
 */

public class StrParaVal extends ParameterVal {

        public String strValue;

}
```

Listing B.5: Parameter Value (ParameterVal.java)

```
/*
 * Created on Aug 15, 2005 @author Yu
 */
package common.parameterVal;

import java.util.*;
import java.io.*;

import common.path.Path;

public class ParameterVal implements Serializable {

        public Date timeStamp;

        public Path path;

}
```

B.1.2 Path

Listing B.6: Path (Path.java)

```
/*
 * Created on Aug 11, 2005 @author Yu
 */

package common.path;

import java.io.*;

public class Path implements Serializable {

        public String strPath;

        public Path(String strPath) {
                this.strPath = strPath;
        }

        public void addPrefix(String prefix) {

                if (!(strPath.equals(""))) {
                        strPath = prefix + "." + strPath;
                }
        }

        public void deletePrefix() {

                int index;
                String strTmp;

                index = strPath.indexOf(".");
                strTmp = strPath.substring(index + 1);
                strPath = strTmp;
        }

        public String getPrefix() {

                int index;
                String strTmp;

                index = strPath.indexOf(".");
                strTmp = strPath.substring(0, index);
                return strTmp;
        }

}
```

B.2 Ground Station

B.2.1 Ground Station Manager

Listing B.7: Ground Station Manager Interface (IGSManager.java)

```
package groundStation.gsManager;
```

```
/*
 * Created on Jun 5, 2005 @author Yu
 */

public interface IGSManager {

        String alarm();

}
```

Listing B.8: Mission Control Center Interface (IMcc.java)

```
/*
 * Created on Jul 26, 2005 @author Yu
 */

package groundStation.gsManager;

import groundStation.session.Session;

import java.rmi.*;

public interface IMcc extends Remote {

        public void addSession(Session session) throws RemoteException;

        public void removeSession(String sessionName)
                                                    throws RemoteException;

}
```

Listing B.9: Manual Operation Interface (IOperator.java)

```
/*
 * Created on Aug 15, 2005 @author Yu
 */

package groundStation.gsManager;

import java.rmi.*;
import java.util.ArrayList;

public interface IOperator extends Remote {

        public void startProtocol() throws RemoteException;

        public void stopProtocol() throws RemoteException;

        public void startTracker() throws RemoteException;

        public void stopTracker() throws RemoteException;

        public void killCurrSession()
                        throws RemoteException, GSManagerException;

        public ArrayList getProtocolStatus() throws RemoteException;

        public ArrayList getTrackerStatus() throws RemoteException;

}
```

Listing B.10: Ground Station Exception (GSManagerException.java)

```
package groundStation.gsManager;

import groundStation.session.ChainedException;

/*
 * Created on Aug 1, 2005 @author Yu
 */

public class GSManagerException extends ChainedException {

        public GSManagerException(String message) {
                super(message);
        }

        public GSManagerException(String message, Throwable cause) {
                super(message, cause);
        }

}
```

Listing B.11: Ground Station Manager (GSManager.java)

```
/*
 * Created on Jun 5, 2005 @author Yu
 */

package groundStation.gsManager;

import groundStation.log.ConsoleLog;
import groundStation.log.ILog;
import groundStation.protocol.IProtocol;
import groundStation.protocol.Protocol;
import groundStation.session.Session;
import groundStation.session.SessionController;
import groundStation.session.SessionQueue;
import groundStation.session.SessionQueueDispatcher;
import groundStation.session.SessionQueueException;
import groundStation.tracker.ITracker;
import groundStation.tracker.Tracker;

import java.io.*;
import java.util.ArrayList;
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.UnicastRemoteObject;
import java.net.*;

public class GSManager extends UnicastRemoteObject
                implements IGSManager, IMcc, IOperator {

        private static final long serialVersionUID = 1L;

        private int mod;

        private int level;

        private String thisAddress;

        private int thisPort;

        private Registry registry;
```

```
        private SessionQueue sQueue;

        private Session session;

        private String logPattern;

        private ILog logger;

        private SessionQueueDispatcher dispatcher;

        private SessionController sessionCtrl;

        private Protocol protocol;

        private ITracker tracker;

        public boolean sessionFlag = false;

        public GSManager() throws RemoteException, GSManagerException {

                //              Create the logger
                logPattern = "Console";
                logger = (ILog) new ConsoleLog(mod, level);
                mod = logger.GSMANAGER;
                level = logger.DEBUG;

                try {
                        thisAddress =
                                (InetAddress.getLocalHost()).toString();
                } catch (Exception e) {
                        throw new
                        RemoteException("Can not get inet address");
                }

                thisPort = 4545; // registry port
                logger.logMessage(mod, level, "this address = "
                                + thisAddress
                                + ", this port = " + thisPort);

                try {
                        registry =
                        LocateRegistry.createRegistry(thisPort);
                        registry.rebind("GSManager", this);
                } catch (RemoteException e) {
                        throw e;
                }

                sQueue = new SessionQueue();

                //start the pusher to monitor session queue
                dispatcher = new
                        SessionQueueDispatcher(sQueue, this, 30);
                logger.logMessage(mod, level, "Dispatcher is running.");
                dispatcher.start();
        }

        public void addSession(Session session) throws RemoteException {

                String name;

                try {
                        name = session.getSessionName();
                        logger.logMessage(mod, level,
                                "New Session id is " + name);
                        sQueue.insertQueue(session);
```

```
                sQueue.printSessionQueue();
        } catch (SessionQueueException e) {
                logger.logMessage(mod, level, e.toString());
        }
}

public void removeSession(String sessionName)
                        throws RemoteException {

        try {
                sQueue.removeSession(sessionName);
                sQueue.printSessionQueue();
        } catch (SessionQueueException e) {
                logger.logMessage(mod, level, e.toString());
        }
}

public String alarm() {
        String result;

        result = "something wrong with GS Manager";
        return (result);
}

public void setEnv() {

        try {
                //              Create Tracker

                tracker = (ITracker) new Tracker(logger);
                protocol = new
                Protocol(session.getProtocoltype(), logger);

        } catch (Exception e) {
                e.printStackTrace();
        }

}

public void startSessionController(Session session)
                                        throws Exception {

        int level = 0;
        int mod = 0;

        this.session = session;
        setEnv();

        //set flag
        sessionFlag = true;

        //Create Session Controller
        sessionCtrl = new SessionController(session, tracker,
                        logger, this, protocol);

        session.getEnv(sessionCtrl, tracker, protocol, logger);

        sessionCtrl.start();
}

public void notifyGSManager() {
        sessionFlag = false;
        try {
                sQueue.deleteSession();
        } catch (SessionQueueException e) {
```

```
                logger.logMessage(mod, level, e.toString());
            }
        }

        public void startProtocol() {
            protocol.startProtocol();
        }

        public void stopProtocol() {
            protocol.stopProtocol();
        }

        public void startTracker() {
            tracker.startTrack();
        }

        public void stopTracker() {
            tracker.stopTrack();
        }

        public void killCurrSession() throws GSManagerException {
            if (sessionFlag) {
                sessionCtrl.killSession();
            } else {
                throw new
                GSManagerException("No session found!");
            }
        }

        public ArrayList getProtocolStatus() {
            ArrayList tmpList;

            tmpList = protocol.getStatus();
            return (tmpList);
        }

        public ArrayList getTrackerStatus() {
            ArrayList tmpList;

            tmpList = tracker.getStatus();
            return (tmpList);

        }

        static public void main(String args[]) {
            try {

                System.setSecurityManager
                        (new RMISecurityManager());
                System.out.println("GSManager is running!");
                GSManager GSMgr = new GSManager();
            } catch (Exception e) {
                e.printStackTrace();
                System.exit(1);
            }
        }

}
```

B.2.2 Log

Listing B.12: Logger Interface (ILog.java)

```
/*
 * Created on Jun 13, 2005 @author Yu
 */

package groundStation.log;

import java.util.Date;

public abstract class ILog {

        public static int log_GSManager = 4;

        public static int log_SessionCtrl = 4;

        public static int log_Session = 4;

        public static int log_Protocol = 4;

        public static int log_Tracker = 4;

        public static int log_Antenna = 4;

        public static int log_Radio = 4;

        public static int log_Timer = 4;

        public static int log_Operator = 4;

        public static final int GSMANAGER = 0;

        public static final int SESSIONCTRL = 1;

        public static final int SESSION = 2;

        public static final int PROTOCOL = 3;

        public static final int TRACKER = 4;

        public static final int ANTENNA = 5;

        public static final int RADIO = 6;

        public static final int TIMER = 7;

        public static final int OPERATOR = 8;

        public static final int DEBUG = 0;

        public static final int INFO = 1;

        public static final int WARN = 2;

        public static final int ERROR = 3;

        public static final int FATAL = 4;

        public void logMessage(int mod, int level, String msg) {
        }

        public void close() {
        }

        public void setLevel(int mod, int logLevel) {
```

```
                if (logLevel > -1 && logLevel < 5) {
                        switch (mod) {
                        case GSMANAGER:
                                log_GSManager = logLevel;
                                break;
                        case SESSIONCTRL:
                                log_SessionCtrl = logLevel;
                                break;
                        case SESSION:
                                log_Session = logLevel;
                                break;
                        case PROTOCOL:
                                log_Protocol = logLevel;
                                break;
                        case TRACKER:
                                log_Tracker = logLevel;
                                break;
                        case ANTENNA:
                                log_Antenna = logLevel;
                                break;
                        case RADIO:
                                log_Radio = logLevel;
                                break;
                        case TIMER:
                                log_Timer = logLevel;
                                break;
                        case OPERATOR:
                                log_Operator = logLevel;
                                break;
                        default:
                                System.out
                                .println
                        ("Ilog.setLevel: No such module in setLevel!");
                                break;
                        }
                } else {
                        System.out
                        .println("Ilog.setLevel: No such log level!");
                }
        }

        public String genLine(String msg) {

                String line = new
                        Date(System.currentTimeMillis()).toString()
                        + " : " + msg + " \n";
                return line;

        }
}
```

Listing B.13: Console Logger (ConsoleLog.java)

```
package groundStation.log;

/*
 * Created on Jun 13, 2005 @author Yu
 */

public class ConsoleLog extends ILog {

        private int level;
```

```
        private int mod;

        public ConsoleLog(int mod, int logLevel) {
                this.level = logLevel;
                this.mod = mod;
                setLevel(mod, level);
        }

        public void logMessage(int mod, int level, String msg) {

                switch (mod) {
                case GSMANAGER:
                        if (log_GSManager >= level)
                                System.out.println(genLine(msg));
                        break;
                case SESSIONCTRL:
                        if (log_SessionCtrl >= level)
                                System.out.println(genLine(msg));
                        break;
                case SESSION:
                        if (log_Session >= level)
                                System.out.println(genLine(msg));
                        break;
                case PROTOCOL:
                        if (log_Protocol >= level)
                                System.out.println(genLine(msg));
                        break;
                case TRACKER:
                        if (log_Tracker >= level)
                                System.out.println(genLine(msg));
                        break;
                case ANTENNA:
                        if (log_Antenna >= level)
                                System.out.println(genLine(msg));
                        break;
                case RADIO:
                        if (log_Radio >= level)
                                System.out.println(genLine(msg));
                        break;
                case TIMER:
                        if (log_Timer >= level)
                                System.out.println(genLine(msg));
                        break;
                case OPERATOR:
                        if (log_Operator >= level)
                                System.out.println(genLine(msg));
                        break;
                default:
                        System.out
                .println(genLine(" No such module in ConsoleLog"));
                }

        }

        public void close() {

        }

}
```

B.2.3 Protocol

Listing B.14: Protocol Interface (IProtocol.java)

```
/*
 * Created on Jun 5, 2005 @author Yu
 */

package groundStation.protocol;

import java.util.ArrayList;

public interface IProtocol {

        public boolean startProtocol();

        public boolean stopProtocol();

        public ArrayList getStatus();

        public void connectTo();

}
```

Listing B.15: Protocol Bridge (ProtocolBridge.java)

```
package groundStation.protocol;

public class ProtocolBridge {

        public native byte[] readSerial(String fileName);

        public native void writeSerial(String fileName, String msg);

        static {
                System.loadLibrary("Protocol");
        }
}
```

Listing B.16: Protocol (Protocol.java)

```
/*
 * Created on Jun 5, 2005 @author Yu
 */

package groundStation.protocol;

import java.util.*;
import java.net.*;
import java.io.*;

import common.parameterVal.StrParaVal;
import common.path.Path;

import groundStation.log.ILog;

public class Protocol extends Thread implements IProtocol {

        private String protocolType;

        private String statusMsg;
```

```
private ILog logger;

private int port;

private String server;

private Thread protocolThread;

public Protocol(String protocolType, ILog logger) {
        this.logger = logger;
        this.protocolType = protocolType;
        port = 1500;
        server = "192.38.79.147";
}

public void connectTo() {

}

public boolean startProtocol() {

        System.out.println("Protocol from server.");
        protocolType = "AX25";
        this.start();
        return true;
}

public boolean stopProtocol() {

        return true;
}

public ArrayList getStatus() {
        StrParaVal val1;
        StrParaVal val2;
        ArrayList tmpList;

        tmpList = new ArrayList();

        val1 = null;
        val2 = null;

        System.out
        .println("request protocol status from client.");

        try {
                val1 = new StrParaVal();
                val1.timeStamp =
                        new Date(System.currentTimeMillis());
                val1.path = new Path("Protocol.Type");
                val1.strValue = protocolType;
        } catch (Exception e) {
                e.printStackTrace();
        }

        try {
                val2 = new StrParaVal();
                val2.timeStamp =
                        new Date(System.currentTimeMillis());
                val2.path = new Path("Protocol.StatusMsg");
                val2.strValue = statusMsg;
        } catch (Exception e) {
                e.printStackTrace();
        }
```

```
            tmpList.add(val1);
            tmpList.add(val2);

            return tmpList;
    }

    public synchronized void run() {

            Socket socket = null;
            String lineToBeSent;
            BufferedReader input;
            PrintWriter output;
            int ERROR = 1;

            try {

                    // setup a tcp connection with MCC
                    try {
                            socket = new Socket(server, port);
                            System.out
                            .println("Connected with server "
                            + socket.getInetAddress() + ":"
                            + socket.getPort());
                    } catch (UnknownHostException e) {
                            System.out.println(e.toString());
                    } catch (IOException e) {
                            System.out.println(e.toString());
                    }

                    ProtocolBridge bridge = new ProtocolBridge();

                    byte[] readMsg;

                    String fileName =
                    "/course/satellite/src/protocolData/ax25.write";
                    bridge.writeSerial(fileName,
                            "It is from protocol message");
                    readMsg = bridge.readSerial
                    ("/course/satellite/src/protocolData/ax25.read");

                    try {
                            input = new BufferedReader
                            (new InputStreamReader(System.in));
                            output = new PrintWriter
                            (socket.getOutputStream(), true);

                            String msg = new String(readMsg);
                            output.println(msg);
                    } catch (IOException e) {
                            System.out.println(e);
                    }

                    try {
                            socket.close();
                    } catch (IOException e) {
                            System.out.println(e);
                    }

                    Thread.sleep(4000);
                    statusMsg =
                    "The protocol thread is in the first step!";

                    Thread.sleep(30000);
                    statusMsg =
                    "The protocol thread is in the second step!";
```

```
                } catch (Exception e) {

                }
        }
}
```

B.2.4 Satellite

Listing B.17: Position (Position.java)

```
/*
 * Position
 * Copyright (C) 2005 Allan Nielsen, Martin Seebach, Thilo Bangert
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA  02110-1301, USA
 */
package groundStation.satellite;

//import dk.dtu.sat.TupleContainer;

/**
 * This contains position parameters for a celestial object.
 *
 * @version 1.0
 * @author awn
 *
 * <pre>
 *
 *  $Id: Position.java 450 2005-06-23 18:19:39Z martin $
 *
 * </pre>
 */
public class Position extends TupleContainer {

        /**
         * Initialisation method. Sets threshold and formatting
         * strings in the parent class.
         */
        private void init() {
                this.equalsThreshold = 2.5;
                this.FormatAz = "000.00\u00B0Az";
                this.FormatEl = "00.00\u00B0El;-00.00\u00B0El";
                this.FormatDs = "00000.0km";
        }

        /**
```

```
         * Constructs a Position object with only Azimuth and
         * Elevation parameters.
         *
         * @param Az
         * @param El
         */
        public Position(double Az, double El) {
                this.El = El;
                this.Az = Az;
                this.Ds = 0;
                this.init();
        }

        /**
         * Constructs a Position object with all parameters.
         *
         * @param Az
         * @param El
         * @param Ds
         */
        public Position(double Az, double El, double Ds) {
                this.El = El;
                this.Az = Az;
                this.Ds = Ds;
                this.init();
        }
}
```

Listing B.18: Position Velocity (PositionVelocity.java)

```
/*
 * PositionVelocity
 * Copyright (C) 2005 Allan Nielsen, Martin Seebach, Thilo Bangert
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA  02110-1301, USA
 */
package groundStation.satellite;

//import dk.dtu.sat.Velocity;

/**
 * Container object to hold the position and velocity of
 * a celestial object.
 *
 * @version 1.0
 * @author seebach, bangert, allan
 *
 * <pre>
 *
 *  $Id: PositionVelocity.java 450 2005-06-23 18:19:39Z martin $
 *
```

```
 * </pre>
 */
public class PositionVelocity {

        /**
         * Position object
         */
        public Position position;

        /**
         * Velocity object
         */
        public Velocity velocity;

        /**
         * This string is inserted before the position-line
         * in toString()
         *
         * @see toString
         */
        public String posPadding = "";

        /**
         * This string is inserted before the velocity-line
         * in toString()
         *
         * @see toString
         */
        public String velPadding = "";

        /**
         * Constructs a PostionVelocity object by holding the
         * given objects of the samen type. The formatting strings
         * are are set to accomodate prettier stringprinting
         *
         * @param pos
         *              Position object
         * @param vel
         *              Velocity object
         */
        public PositionVelocity(Position pos, Velocity vel) {
                this.position = pos;
                pos.FormatAz = "000.00\u00B0  Az";
                pos.FormatEl = " 00.00\u00B0  El;-00.00\u00B0  El";
                pos.FormatDs = "00000.0    km";
                this.velocity = vel;
                vel.FormatAz = " +0.0000\u00B0Az; -0.0000\u00B0Az";
                vel.FormatEl = " +0.0000\u00B0El; -0.0000\u00B0El";
                vel.FormatDs = "   +0.0000 km ;   -0.0000 km ";
        }

        /**
         * Formats the contained objects on two lines,
         * prepended with respectively vel- and posPadding
         *
         * @see velPadding
         * @see posPadding
         */
        public String toString() {
                return posPadding + "Pos: " + position.toString()
                                + "\n" + velPadding
                                + "Vel: " + velocity.toString();
        }
}
```

Listing B.19: Predict Interface (PredictInterface.java)

```
/*
 * PredictInterface
 * Copyright (C) 2005 Allan Nielsen, Martin Seebach, Thilo Bangert
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA  02110-1301, USA
 */
package groundStation.satellite;

import java.util.Calendar;

/**
 * Defines how the pass of a celestial object is communicated.
 * Two main areas of function are expected: - The availability
 * of horizontal coordinates to any given time - The prediction
 * of a (to a given time) future or current pass of an object.
 *
 * @version 1.0
 * @author seebach, allan, bangert
 *
 * <pre>
 *
 *  $Id: PredictInterface.java 450 2005-06-23 18:19:39Z martin $
 *
 * </pre>
 */
public interface PredictInterface {

        /**
         * Get Position to the given time.
         *
         * @param t
         *            Any point in time
         * @return Position of object to the given time
         */
        public Position getHorizontalPosition(Calendar t);

        /**
         * Get Position and Velocity to the given time. Galculate
         * Velocity over deltaT milliseconds
         *
         * @param t
         *            Any point in time
         * @param deltaT
         *            Timespan over which to calculate velocity
         * @return Position and Velocity of object to the given time
         */
        public PositionVelocity getHorizontalPositionVelocity
                                        (Calendar t, int deltaT);

        /**
```

```
         * Find when the first pass after the given time will begin.
         *
         * @param t
         *            Any point in time
         * @return The time of the beginning of the next pass
         */
        public Calendar getNextPass(Calendar t);

        /**
         * Find when the first pass after the given time will end.
         *
         * @param t
         *            Any point in time
         * @return The time of the end of the next or current pass
         */
        public Calendar getPassEnd(Calendar t);

        /**
         * Find the duration in seconds of the pass beginning on
         * the given time.
         * @param t
         *            Output of getNextPass()
         * @return The duration of the pass in seconds
         */
        public long getPassDuration(Calendar t);

}
```

Listing B.20: Predict Simulator (PredictSimulator.java)

```
/*
 * Created on Jun 14, 2005
 *
 * $Id: PredictSimulator.java 435 2005-06-22 15:57:13Z allan $
 */
package groundStation.satellite;

import java.util.Calendar;

/**
 * @author awn
 *
 */
public class PredictSimulator implements PredictInterface {

        Position start;

        Position stop;

        Position deltaCoor;

        long DurationTime;

        Calendar startTime;

        boolean clockwise;

        public PredictSimulator(TLE t) {
                start = new Position(350, 0);
                stop = new Position(10, 180);
                clockwise = true;
                DurationTime = 100000;
                startTime = Calendar.getInstance();
                startTime.add(Calendar.SECOND, 30);
```

```
                deltaCoor = new
                Position(stop.Az - start.Az, stop.El - start.El);

                if (clockwise) {

                } else {

                }

        }

        public Position getHorizontalPosition(Calendar t) {
                long deltaTime =
                t.getTimeInMillis() - startTime.getTimeInMillis();

                if (deltaTime <= 0)
                        return start;
                else if (deltaTime > DurationTime)
                        return stop;

                return new Position(start.Az
                                + (deltaCoor.Az / (double) DurationTime)
                                * deltaTime, start.El
                                + (deltaCoor.El / (double) DurationTime)
                                * deltaTime);
        }

        public PositionVelocity getHorizontalPositionVelocity
                                        (Calendar t, int i) {
                long deltaTime =
                t.getTimeInMillis() - startTime.getTimeInMillis();
                if (deltaTime <= 0)
                        return new
                        PositionVelocity(start, new Velocity(0, 0));
                return new PositionVelocity(new Position(start.Az
                                + (deltaCoor.Az / (double) DurationTime)
                                * deltaTime, start.El
                                + (deltaCoor.El / (double) DurationTime)
                                * deltaTime),
                                new Velocity(0, 0));
        }

        public Calendar getNextPass(Calendar t) {
                return startTime;
        }

        public Calendar getPassEnd(Calendar t) {
                Calendar cal = Calendar.getInstance();
                cal.setTime(t.getTime());
                cal.add(Calendar.MILLISECOND, (int) DurationTime);
                return cal;
        }

        public long getPassDuration(Calendar t) {
                return DurationTime;
        }
}
```

Listing B.21: Satellite (Satellite.java)

```
/*
 * Sattelite
 * Copyright (C) 2005 Allan Nielsen, Martin Seebach, Thilo Bangert
 *
```

```
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA  02110-1301, USA
 */
package groundStation.satellite;

//import dk.dtu.sat.TLE;

/**
 *
 * A Satellite. all properties concerning a satellite.
 *
 * <pre>
 *
 *
 *   TODO
 *   ====
 *   - The radio properties have not been well defined (largely
 *  due to missing experience in this area). (This will affect
 *  many other areas within the project - RadioInterface, Tracker ao.)
 *
 *
 * </pre>
 *
 * @author bangert, martin, allan
 * @version 1.0
 *
 * <pre>
 *
 *  $Id: Satellite.java 464 2005-06-24 12:40:12Z bangert $
 *
 * </pre>
 */
public class Satellite {

        //the satellites name
        private String commonName;

        //the satellites TLE
        private TLE tle;

        //the satellites receive frequency
        private int rxFrequency;

        //the satellites transmit frequency
        private int txFrequency;

        /**
         * Create a Satellite where only the name is set.
         *
         * @param name
         *            the name of the satellite.
         */
        public Satellite(String name) {
```

```
        commonName = name;
    }

    /**
     * Create a Satellite from a name and its TLE.
     *
     * @param name
     *            the name of the satellite.
     * @param t
     *            the satellites TLE object.
     */
    public Satellite(String name, TLE t) {
        commonName = name;
        tle = t;
    }

    /**
     * Set the satellites TLE object.
     *
     * @param t -
     *            the TLE object.
     */
    public void setTLE(TLE t) {
        tle = t;
    }

    /**
     * Get the satellites TLE object.
     *
     * @return the TLE object.
     */
    public TLE getTLE() {
        return tle;
    }

    /**
     * Get the receive frequency.
     *
     * @return the current receive frequency.
     */
    public int getRXFrequency() {
        return rxFrequency;
    }

    /**
     * Get the transmit frequency.
     *
     * @return the current transmit frequency.
     */
    public int getTXFrequency() {
        return txFrequency;
    }

    /**
     * Set this Satellites receive frequency.
     *
     * @param freq
     *            the receive frequency.
     */
    public void setRXFrequency(int freq) {
        rxFrequency = freq;
    }

    /**
     * Sets this Satellites transmit frequency.
```

```
         *
         * @param freq
         *            the transmit frequency.
         */
        public void setTXFrequency(int freq) {
                txFrequency = freq;
        }

        /**
         * Get the Satellites name.
         *
         * @return the name of the satellite.
         */
        public String getCommonName() {
                return commonName;
        }
}
```

Listing B.22: Satellite Reader (SatelliteReader.java)

```
/*
 * SatteliteReader
 * Copyright (C) 2005 Allan Nielsen, Martin Seebach, Thilo Bangert
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA  02110-1301, USA
 */
package groundStation.satellite;

import java.io.Reader;
import java.io.BufferedReader;
import java.io.IOException;

//import dk.dtu.sat.TLE;

/**
 * A SatelliteReader. Helper functions to easily create Satellite
 * object from TLE files. Usually one would download a TLE file
 * from http://www.celestrak.com/, use a FileReader to open it
 * and a SatelliteReader to get the data out of it.
 *
 * @author bangert, martin, allan
 * @version 1.0
 *
 * <pre>
 *
 *  $Id: SatelliteReader.java 464 2005-06-24 12:40:12Z bangert $
 *
 * </pre>
 */
public class SatelliteReader extends BufferedReader {
```

```
        /**
         * Takes a Reader as argument - typically a FileReader.
         *
         * @param in -
         *             the Reader providing data to the SatelliteReader.
         */
        public SatelliteReader(Reader in) {
                super(in);
        }

        /**
         * This function will search through the given Reader for a
         * line that matches the given Satellite name.
         *
         * @param common_name -
         *             name of the wanted satellite.
         * @return the wanted Satellite. Is null if the satellite
         *         could not be found.
         * @throws IOException
         */
        public Satellite getSatellite(String common_name)
                                        throws IOException {
                Satellite sat = null;
                TLE tle = null;

                String line;
                while (ready()) {
                        line = readLine();
                        //System.out.println(line.trim());
                        if (line.trim().equals(common_name)) {
                                sat = new Satellite(line.trim(),
                                        new TLE(readLine(), readLine()));

                                break;
                        } else {
                                //skip two lines
                                line = readLine();
                                line = readLine();
                        }
                }
                return sat;
        }

        /**
         * A functiont that returns the next Satellite object in a
         * TLE file. Usefull if you want to create satellite objects
         * of all satellites in a file.
         *
         * @return the read Satellite object.
         * @throws IOException
         */
        public Satellite getNextSatellite() throws IOException {
                Satellite sat = null;

                String name, line1, line2;
                name = readLine();
                line1 = readLine();
                line2 = readLine();
                if (name != null && line1 != null && line2 != null)
                        sat = new Satellite(name, new TLE(line1, line2));

                return sat;
        }
}
```

Listing B.23: Tuple Container (TupleContainer.java)

```
/*
 * TupleContainer
 * Copyright (C) 2005 Allan Nielsen, Martin Seebach, Thilo Bangert
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA  02110-1301, USA
 */
package groundStation.satellite;

import java.text.DecimalFormat;

//import dk.dtu.sat.Velocity;

/**
 *
 * TupleContainer is a container for the Position and Velocity classes.
 * Provides the toString and almostEquals methods which are very alike.
 *
 * @see Velocity
 * @see Position
 * @author bangert, martin, allan
 * @version 1.0
 *
 * <pre>
 *
 *  $Id: TupleContainer.java 450 2005-06-23 18:19:39Z martin $
 *
 * </pre>
 */
public abstract class TupleContainer {
        // Elevation, Azimuth, Distance
        /**
         * Elevation parameter, expected unit is degrees or
         * degrees/second.
         */
        public double El;

        /**
         * Azimuth parameter, expected unit is degrees or
         * degrees/second.
         */
        public double Az;

        /**
         * Distance parameter, expected unit is kilometers or
         * kilometers/second.
         */
        public double Ds;

        // the difference which is small enough to consider
        // two tuples equal
```

```
        protected double equalsThreshold = 0;

        // Formatstrings for DecimalFormat
        protected String FormatAz;

        protected String FormatEl;

        protected String FormatDs;

        /**
         * Nicely format the contents of the object.
         *
         * @return Formatted string
         */
        public String toString() {

                DecimalFormat f_az = new DecimalFormat(this.FormatAz);
                DecimalFormat f_el = new DecimalFormat(this.FormatEl);
                DecimalFormat f_dist = new DecimalFormat(this.FormatDs);

                if (Math.round(Ds) == 0)
                        return f_az.format(Az) + " " + f_el.format(El);
                else
                        return f_az.format(Az) + " " + f_el.format(El)
                                        + " " + f_dist.format(Ds);
        }

        /**
         * @param o
         *            Another TupleContainer to compare.
         * @return true if the two objects deviate within the set
         *         limit, otherwise false.
         */
        public boolean almostEquals(TupleContainer o) {
                return (Math.abs(El - o.El) <= equalsThreshold)
                        && (Math.abs(Az - o.Az) <= equalsThreshold);
        }
}
```

Listing B.24: Velocity (Velocity.java)

```
/*
 * The Velocity Object
 * Copyright (C) 2005 Allan Nielsen, Martin Seebach, Thilo Bangert
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA  02110-1301, USA
 */
package groundStation.satellite;

/**
 * This contains velocity parameters for a celestial object.
```

```
 *
 * @version 1.0
 * @author allan, bangert, seebach
 *
 * <pre>
 * $Id: Velocity.java 450 2005-06-23 18:19:39Z martin $
 * </pre>
 */
public class Velocity extends TupleContainer {

        /**
         * Initialisation method. Sets threshold and formatting strings
         * in the parent class.
         */
        private void init() {
                this.equalsThreshold = 0.001;
                this.FormatAz = "+0.0000\u00B0Az;-0.0000\u00B0Az";
                this.FormatEl = "+0.0000\u00B0El;-0.0000\u00B0El";
                this.FormatDs = "+0.0000 km;-0.0000 km";
        }

        /**
         * Constructs a Velocity object with only Azimuth and Elevation
         * parameters.
         * @param Az
         *              Degrees azimuth pr. second.
         * @param El
         *              Degrees elevation pr. second.
         */
        public Velocity(double Az, double El) {
                this.El = El;
                this.Az = Az;
                this.Ds = 0;
                this.init();
        }

        /**
         * Constructs a Velocity object with all parameters.
         *
         * @param Az
         *              Degrees azimuth pr. second.
         * @param El
         *              Degrees elevation pr. second.
         * @param Ds
         *              Kilometer distance pr. second.
         */
        public Velocity(double Az, double El, double Ds) {
                this.El = El;
                this.Az = Az;
                this.Ds = Ds;
                this.init();
        }

        /**
         * Constructs a Velocity object from two distinct positions.
         *
         * @param p1
         *              The first position
         * @param p2
         *              The second position
         * @param deltaT
         *              The time between p1 and p2 in milliseconds
         */
        public Velocity(Position p1, Position p2, int deltaT) {
                this.Az = (p2.Az - p1.Az) * (1000. / deltaT);
```

```
                this.El = (p2.El - p1.El) * (1000. / deltaT);
                this.Ds = (p2.Ds - p1.Ds) * (1000. / deltaT);

                this.init();
        }
}
```

B.2.5 Session

Listing B.25: Chained Exception (ChainedException.java)

```
package groundStation.session;

/*
 * Created on Jul 31, 2005 @author Yu
 */

public class ChainedException extends Exception {

        private Throwable cause = null;

        public ChainedException() {
                super();
        }

        public ChainedException(String message) {
                super(message);
        }

        public ChainedException(String message, Throwable cause) {
                super(message);
                this.cause = cause;
        }

        public void printStackTrace() {
                super.printStackTrace();
                if (cause != null) {
                        System.out.println("Caused by: ");
                        cause.printStackTrace();
                }
        }

        public void printStackTrace(java.io.PrintStream ps) {
                super.printStackTrace(ps);
                if (cause != null) {
                        ps.println("Caused by: ");
                        cause.printStackTrace(ps);
                }
        }

        public void printStackTrace(java.io.PrintWriter pw) {
                super.printStackTrace(pw);
                if (cause != null) {
                        pw.println("Caused by: ");
                        cause.printStackTrace(pw);
                }
        }

}
```

Listing B.26: Session Interface (ISession.java)

```
package groundStation.session;

import groundStation.log.ILog;
import groundStation.protocol.Protocol;
import groundStation.tracker.ITracker;

import java.util.*;

public interface ISession {

        public void stopSession(Thread session);

        public Thread startSession(Session session);

        public void getEnv(ISessionController ctrl, ITracker tracker,
                        Protocol protocol, ILog logger);

        public ArrayList getStatus();

}
```

Listing B.27: Session Controller Interface (ISessionController.java)

```
package groundStation.session;

/*
 * Created on Jul 30, 2005 @author Yu
 */

public interface ISessionController {

        public void notifyController(Object ctrl);

}
```

Listing B.28: Session (Session.java)

```
/*
 * Created on Jun 5, 2005 @author Yu
 */

package groundStation.session;

import groundStation.log.ILog;
import groundStation.protocol.Protocol;
import groundStation.tracker.ITracker;
import groundStation.satellite.Position;
import java.util.*;

import java.io.*;

public class Session implements ISession, Serializable, Runnable {

        private static final long serialVersionUID = 1L;

        private int sessionID;

        private String sessionName;

        private Calendar startTime;
```

```
private Calendar endTime;

private String protocolType;

private String status;

private Position expectedPos;

private ISessionController ctrl;

private ITracker tracker;

private Protocol protocol;

private ILog logger;

private int mod = 2;

private int level = 0;

private int interval = 5; //in seconds

public Session() {
        protocolType = "AX25";
}

public int getSessionID() {
        return sessionID;
}

public void setSessionID(int ID) {
        sessionID = ID;
}

public String getSessionName() {
        return sessionName;
}

public void setSessionName(String name) {
        sessionName = name;
}

public Calendar getStartTime() {
        return startTime;
}

public void setStartTime(Calendar start) {
        startTime = start;
}

public Calendar getEndTime() {
        return endTime;
}

public String getProtocoltype() {
        return protocolType;
}

public void setEndTime(Calendar end) {
        endTime = end;
}

public ArrayList getStatus() {
        ArrayList tmpList;
```

```
        tmpList = null;
        return tmpList;
}

public void setStatus(String stat) {
        status = stat;
}

public void getEnv(ISessionController ctrl, ITracker tracker,
                Protocol protocol, ILog logger) {

        this.ctrl = ctrl;
        this.tracker = tracker;
        this.protocol = protocol;
        this.logger = logger;
}

public Thread startSession(Session session) {

        Thread sessionThread = new Thread(session);
        sessionThread.start();
        return sessionThread;
}

public void stopSession(Thread session) {
        try {
                logger.logMessage(mod, level,
        "Session(stopSession): Session has been interrupted.");
                session.interrupt();
        } catch (Exception ex) {
                logger.logMessage(mod, level,
                                "Session(stopSession): "
                                + ex.getMessage());
        }

}

public synchronized void run() {

        int mod = 2;
        int level = 0;
        Position expectedPos;

        logger
        .logMessage(mod, level,
        "Session: The Session thread is running!");

        try {

                synchronized (tracker) {

                        Thread protocolThread =
                                new Thread(protocol);
                        protocolThread.start();

                        Thread.sleep(100 * 1000);
                        logger.logMessage(mod, level,
                                        "setting satellite");
                        tracker.initSatellite("DTUSAT", 250);

                        logger.logMessage(mod, level,
                                        "starting initTrack");
                        tracker.initTrack(startTime);
```

```
                                Thread.sleep(30 * 1000);
                                expectedPos = tracker.getExpectedPos();
                                logger.logMessage(mod, level,
                                                expectedPos.toString());
                                if (tracker.checkAntennaPosition
                                                (expectedPos)) {
                                        tracker.startTrack();
                                }

                        }

                        synchronized (ctrl) {
                                ctrl.notifyController(ctrl);
                        }

                } catch (Exception ex) {
                        logger.logMessage(mod, level, "Session: "
                                        + ex.getMessage());
                }

                logger
                .logMessage(mod, level,
                "Session: The Session thread terminated!");
        }
}
```

Listing B.29: Session Controller (SessionController.java)

```
/*
 * Created on Jun 5, 2005 @author Yu
 */

package groundStation.session;

import groundStation.gsManager.GSManager;
import groundStation.log.ILog;
import groundStation.protocol.Protocol;
import groundStation.tracker.ITracker;

import java.util.*;

public class SessionController extends Thread implements
                                        ISessionController {

        private Session session;

        private ITracker tracker;

        private ILog logger;

        private GSManager gsMgr;

        private Protocol protocol;

        private Calendar startTime;

        private Calendar endTime;

        private int mod = 1;

        private int level = 0;

        private boolean waitFlag = false;
```

```
        private Thread sessionThread;

        public SessionController(Session session, ITracker tracker,
                        ILog logger, GSManager gsMgr,
                        Protocol protocol) {

                this.session = session;
                this.tracker = tracker;
                this.logger = logger;
                this.gsMgr = gsMgr;
                this.protocol = protocol;
                startTime = session.getStartTime();
                endTime = session.getEndTime();

                if (checkTime()) {
                        if (checkProtocolType()) {

                        } else {

                        }
                } else {

                }

        }

        private boolean checkProtocolType() {

                return true;
        }

        private boolean checkTime() {

                boolean flag;

                Calendar cal =
                Calendar.getInstance(TimeZone.getDefault());
                flag = cal.after(startTime);
                return true;
        }

        public void notifyController(Object ctrl) {
                try {
                        ctrl.notify();
                        waitFlag = true;
                } catch (Exception ex) {
                        logger.logMessage(mod, level,
                                        "Session Controller: "
                                        + ex.getMessage());
                }
        }

        public void killSession() {
                session.stopSession(sessionThread);
        }

        public void getStatus() {

        }

        public boolean stopSessionController(SessionController ctrl) {

                try {
                        ctrl.interrupt();
                        return true;
```

```
                } catch (Exception ex) {
                        logger.logMessage(mod, level,
                        "Session Controller: The thread is terminated!");
                        return false;
                }

        }

        public synchronized void run() {

                long duration;

                waitFlag = false;
                session.getEnv(this, tracker, protocol, logger);
                sessionThread = session.startSession(session);

                duration = endTime.getTimeInMillis()
                        - startTime.getTimeInMillis();
                logger.logMessage(mod, level, "the session period: "
                                + Long.toString(duration));

                try {
                        logger.logMessage(mod, level,
                        "Session Controller: The thread is waiting!");
                        this.wait(duration);

                        if (!waitFlag)
                                session.stopSession(sessionThread);

                } catch (InterruptedException ex) {
                        logger.logMessage(mod, level,
                                        "Session Controller: "
                                        + ex.getMessage());
                } finally {
                        gsMgr.notifyGSManager();
                }

                logger.logMessage(mod, level,
                "Session Controller: The thread is terminated!");
        }
}
```

Listing B.30: Session Exception (SessionException.java)

```
package groundStation.session;

/*
 * Created on Jul 31, 2005 @author Yu
 */

public class SessionException extends ChainedException {

        public SessionException(String message) {
                super(message);
        }

        public SessionException(String message, Throwable cause) {
                super(message, cause);
        }

}
```

Listing B.31: Session Queue (SessionQueue.java)

```
/*
 * Created on Jul 26, 2005 @author Yu
 */

package groundStation.session;

import java.util.*;

public class SessionQueue {

        private ArrayList sessionQueue;

        public SessionQueue() {
                sessionQueue = new ArrayList();
        }

        public ArrayList getSessionQueue() {
                return sessionQueue;
        }

        public boolean insertQueue(Session session) throws
                                        SessionQueueException {
                int i;
                Calendar startTime;
                Calendar endTime;
                Calendar tmpST;
                Calendar tmpET;
                Calendar currDate;
                Session tmpSession;
                int count;
                String message = "";
                String name;
                int interval;

                try {
                        message = "";
                        startTime = session.getStartTime();
                        endTime = session.getEndTime();
                        name = session.getSessionName();
                        interval = 5; // the minutes

                        if ((name == null) || (name == "")) {
                                message = "The session name is empty.";
                                throw new SessionQueueException(message);
                        }

                        if (startTime.after(endTime)) {
                                message =
                        "The session start time is after end time";
                                throw new SessionQueueException(message);
                        }

                        count = sessionQueue.size();

                        currDate = new
                        GregorianCalendar(TimeZone.getDefault());
                        currDate.set(Calendar.MONTH,
                                (currDate.get(Calendar.MONTH) + 1));
                        currDate.set(Calendar.MINUTE,
                        (currDate.get(Calendar.MINUTE) + interval));

                        if (currDate.after(startTime)) {
                                message =
```

```
                    "The start time of new session is outdate";
                            throw new SessionQueueException(message);
                    }

                    for (i = 0; i < count; i++) {
                            tmpSession = (Session)
                            (sessionQueue.get(i));
                            if (name.equalsIgnoreCase
                            (tmpSession.getSessionName())) {
                                    message =
    "The Session name is duplicated with a existing session.";
                                    throw new SessionQueueException
                                    (message);
                            }
                    }

                    synchronized (sessionQueue) {
                      for (i = 0; i < count; i++) {
                        tmpSession = (Session) (sessionQueue.get(i));
                        tmpST = tmpSession.getStartTime();
                        tmpET = tmpSession.getEndTime();
                        if ((tmpET.after(startTime))) {
                          if (tmpST.before(startTime)) {
                            message =
            "The new session overlaps with some existing sessions";
                            throw new
                            SessionQueueException(message);
                            //return false;
                          } else {
                            if (tmpST.before(endTime)) {
                              message =
            "The new session overlaps with some existing sessions";
                              throw new
                              SessionQueueException(message);
                              //return false;
                            } else {
                              sessionQueue.add(i, session);
                              return true;
                            }
                          }
                        }
                    }
                            sessionQueue.add(i, session);
                            return true;
                    }
            } catch (Exception e) {
                    throw new SessionQueueException
                                    ("Add new session failed: "
                                    + message, e);
            }
    }

    public boolean removeSession(String sessionName)
                    throws SessionQueueException {
            int count, i;
            String name;
            Session tmpSession;
            String message = "";

            try {
                    message = "";
                    count = sessionQueue.size();

                    synchronized (sessionQueue) {
                            for (i = 0; i < count; i++) {
```

```
                                        tmpSession = (Session)
                                        sessionQueue.get(i);
                                        name =
                                        tmpSession.getSessionName();
                                        if (name.equalsIgnoreCase
                                                (sessionName)) {
                                                System.out.println
                                        ("Removed session successfully:"
                                                                + name);
                                                sessionQueue.remove(i);
                                                return true;
                                        }
                                }
                        }

                        if (i == count) {
                                message =
                "The Session name does not exist in session queue";
                                throw new SessionQueueException("");
                        }
                        return true;
                } catch (Exception e) {
                        throw new SessionQueueException
                                        ("Remove Sesssion failed: "
                                        + message, e);
                }
        }

        public void deleteSession() throws SessionQueueException {
                try {
                        synchronized (sessionQueue) {
                                sessionQueue.remove(0);
                        }
                } catch (Exception e) {
                        throw new SessionQueueException
                        ("Delete Session failed", e);
                }
        }

        public void printSessionQueue() {
                int count;
                int i;
                Session tmpSession;

                count = sessionQueue.size();
                for (i = 0; i < count; i++) {
                        tmpSession = (Session) sessionQueue.get(i);
                        System.out.println("Still in Session Queue: "
                                        + tmpSession.getSessionName());
                }
        }
}
```

Listing B.32: Session Queue Dispatcher (SessionQueueDispatcher.java)

```
/*
 * Created on Aug 1, 2005 @author Yu
 */

package groundStation.session;

import groundStation.gsManager.GSManager;

import java.util.*;
```

```
public class SessionQueueDispatcher extends Thread {

        private SessionQueue sQueue;

        private Session session;

        private GSManager gsMgr;

        private ArrayList sessionList;

        private int interval; //in minute

        private int sleepTime; // in minute

        private int times;

        public SessionQueueDispatcher(SessionQueue sQueue,
                        GSManager gsMgr, int interval) {

                this.sQueue = sQueue;
                this.gsMgr = gsMgr;
                this.interval = interval;
                sleepTime = 5;
                times = 0;
        }

        private boolean checkSessionQueue() {
                int count;
                Calendar currTime;
                Calendar startTime;
                Session tmpSession;

                sessionList = sQueue.getSessionQueue();
                count = sessionList.size();
                System.out.println("The Session Queue has " + count
                                + " sessions waiting");

                if (count != 0) {
                        tmpSession = (Session) sessionList.get(0);
                        startTime = tmpSession.getStartTime();
                        currTime = new
                        GregorianCalendar(TimeZone.getDefault());
                        currTime.set(Calendar.MONTH,
                        (currTime.get(Calendar.MONTH) + 1));
                        currTime.set(Calendar.MINUTE,
                        (currTime.get(Calendar.MINUTE) + interval));
                        if (currTime.after(startTime)) {
                                session = tmpSession;
                                return true;
                        }
                }
                return false;
        }

        public synchronized void run() {

                try {
                        while (true) {
                                System.out.println
                                ("The Dispatcher is checking "
                                + times + " times");
                                sleep(10000);
                                if (!gsMgr.sessionFlag) {
                                        if (checkSessionQueue()) {
```

```
                                                    gsMgr
                                                    .startSessionController
                                                    (session);
                                            }
                                    }
                                    times++;
                            }
                    } catch (Exception e) {

                    }
            }
}
```

Listing B.33: Session Queue Exception (SessionQueueException.java)

```
package groundStation.session;

/*
 * Created on Jul 31, 2005 @author Yu
 */

public class SessionQueueException extends ChainedException {

        public SessionQueueException(String message) {
                super(message);
        }

        public SessionQueueException(String message, Throwable cause) {
                super(message, cause);
        }

}
```

B.2.6 Tracker

Listing B.34: Tracker Interface (ITracker.java)

```
/*
 * Created on Jun 5, 2005 @author Yu
 */

package groundStation.tracker;

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

import groundStation.satellite.Position;
import groundStation.satellite.Satellite;

public interface ITracker {

        public void startTrack();

        public Calendar initTrack(Calendar time);

        public void stopTrack();

        public ArrayList getStatus();
```

```
        public boolean checkAntennaPosition(Position expectedPos)
                        throws TrackerException;

        public Position getExpectedPos();

        public void initSatellite(String satelliteName, int rxFreq);

}
```

Listing B.35: Tracker (Tracker.java)

```
/*
 * The Tracker Object
 * Copyright (C) 2005 Allan Nielsen, Martin Seebach, Thilo Bangert
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
 * 02110-1301, USA
 */
package groundStation.tracker;

import groundStation.satellite.Position;
import groundStation.satellite.PositionVelocity;
import groundStation.satellite.Predict;
import groundStation.satellite.PredictInterface;
import groundStation.satellite.Satellite;
import groundStation.satellite.SatelliteReader;
import groundStation.satellite.Velocity;
import groundStation.log.ILog;

import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.Vector;
import java.util.TimeZone;

//import javax.swing.JOptionPane;
//import javax.swing.Timer;
import java.util.Timer;
import java.util.TimerTask;

import common.parameterVal.StrParaVal;
import common.path.Path;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

//import dk.dtu.sat.PredictSimulator;
//import groundStation.tracker.TrackerListener;
//import dk.dtu.sat.tracker.TrackerStatus;
```

```
import groundStation.tracker.motor.MotorDriver;
import groundStation.tracker.motor.GS232a;
import groundStation.tracker.motor.SimuMotor;
import groundStation.tracker.motor.PositionTransformer;
import groundStation.tracker.motor.GS232aPositionTransformer;
//import dk.dtu.sat.tracker.motor.DummyPositionTransformer;
import groundStation.tracker.radio.RadioDriver;
import groundStation.tracker.radio.FT847;

/**
 * Tracker - can track Satellites using data from Predict to control a
 * MotorDriver (to position an antenna) and a RadioDriver to do Doppler
 * correction.
 *
 *
 * @author bangert, martin, allan
 * @version 1.0
 *
 * <pre>
 *
 *  $Id: Tracker.java 473 2005-06-24 21:33:43Z allan $
 *
 * </pre>
 */
public class Tracker implements ITracker {

        private Satellite sat;

        private MotorDriver motor;

        private RadioDriver radio;

        private PredictInterface predict;

        private PositionTransformer positionTransformer;

        private Vector trackerListeners = new Vector();

        //private Position nextPos;
        private Position currentPos;

        private Position expectedPos;

        Position nextPos2;

        Position currentPos2;

        Position expectedPos2;

        //the start of the next scheduled pass
        private Calendar nextPassDate;

        //the end of the next scheduled pass
        private Calendar endPassDate;

        private Timer timer = null;

        //number of milisecs between an update
        private final int period = 200;

        //the speed of light in km/s
        private final double speedOfLight = 299792.4580;

        private ILog logger;
```

```
	private int mod = 4;

	private int level = 0;

	/**
	 * Tracker - supervises a pair of motors for antenna control
	 * and a radio for doppler correction
	 *
	 * @since 1.0
	 */
	public Tracker(ILog logger) {

		this.logger = logger;
		//timer = new Timer();
		motor = new SimuMotor();
		//motor = new GS232a(period);
		radio = new FT847();
		currentPos = motor.getPosition();

		expectedPos = new Position(0, 0);
	}

	/**
	 * Set the satellite that should be tracked.
	 *
	 * @param s -
	 *            the satellite.
	 * @since 1.0
	 */
	public void setSatellite(Satellite s) {
		//TODO - handle already existing satellite
		sat = s;
		//predict = new PredictSimulator( sat.getTLE() );
		predict = new Predict(sat.getTLE());
		motor.open();
		radio.open();

		currentPos = motor.getPosition();
		expectedPos =
		predict.getHorizontalPosition(Calendar.getInstance());
	}

	public void initSatellite(String satelliteName, int rxFreq) {

		if (satelliteName.length() == 0)
			satelliteName = "CUTE-1";
		//Get Satellite object
		try {
			SatelliteReader satr = new SatelliteReader(
						new FileReader(
						"./data/sats.txt"));
			sat = satr.getSatellite(satelliteName);
		} catch (IOException e) {
			logger.logMessage(mod, level, e.toString());
		}

		if (sat == null) {
			logger.logMessage
			(mod, level, "No such satellite!");
		} else {
			if (sat.getCommonName().equals("CUTE-1")) {
				//CUTESAT...
				//436,8375 MHz
				sat.setRXFrequency(436837500);
			}
```

```
                    if (sat.getCommonName().equals("DTUSAT")) {
                            //DTUSAT...
                            //????? MHz
                            sat.setRXFrequency(0);
                    }
                    if (sat.getCommonName().equals("QUAKESAT")) {
                            //QUAKESAT...
                            //436.675 MHz
                            sat.setRXFrequency(436675000);
                    }
                    if (sat.getCommonName().equals("KO-25")) {
                            sat.setRXFrequency(436500000);
                    }
                    if (sat.getCommonName().equals("KO-23")) {
                            sat.setRXFrequency(435175000);
                    }
                    if (sat.getCommonName().equals("CUBESAT XI-IV"))
                    {
                            sat.setRXFrequency(436847500);
                    }

                    if (sat.getRXFrequency() == 0) {
                            sat.setRXFrequency(rxFreq);
                    }
            }

            setSatellite(sat);
            logger.logMessage(mod, level, "Satellite initialized");

    }

    /**
     * Notifiy the given TrackerListener to events from the Tracker.
     *
     * @param a
     *              the object implementing the TrackerListener
     *               interface.
     * @since 1.0
     */
    public void addTrackerListener(TrackerListener a) {
            if (trackerListeners.contains(a))
                    return;
            trackerListeners.addElement(a);
    }

    /**
     * Returns the current status.
     *
     * @return the current status.
     * @since 1.0
     */
    public ArrayList getStatus() {
            ArrayList tmpList;
            StrParaVal val1;
            StrParaVal val2;
            Position currPos;

            tmpList = null;
            val1 = null;
            val2 = null;

            try {
                    tmpList = new ArrayList();
                    currPos = motor.getPosition();
```

```
                    val1 = new StrParaVal();
                    val1.timeStamp =
                    new Date(System.currentTimeMillis());
                    val1.path = new Path("Tracker.CurrentPosition");
                    val1.strValue = currPos.toString();

                    val2 = new StrParaVal();
                    val2.timeStamp =
                    new Date(System.currentTimeMillis());
                    val2.path = new Path("Tracker.ExpectedPosition");
                    val2.strValue = expectedPos.toString();

            } catch (Exception e) {

            }

            tmpList.add(val1);
            tmpList.add(val2);
            return tmpList;
            //return new TrackerStatus(currentPos,expectedPos);
    }

    /**
     * Inform all Listeners about status updates.
     *
     */
    private void updateStatusListeners() {
            TrackerStatus status = new
            TrackerStatus(currentPos, expectedPos);
            TrackerListener target = null;

            for (int x = 0; x < trackerListeners.size(); x++) {
                    target = (TrackerListener)
                    trackerListeners.elementAt(x);
                    target.trackerUpdate(status);
            }
    }

    /**
     * Tell all Listeners that the antenna is in the startposition.
     * Tracking can start now.
     *
     */
    private void inPositionNotify() {
            TrackerListener target = null;
            for (int x = 0; x < trackerListeners.size(); x++) {
                    target = (TrackerListener)
                    trackerListeners.elementAt(x);
                    target.antennaInPosition();
            }
    }

    /**
     * Tell all Listeners that tracking has finished (possibly
     * because it was canceled).
     *
     */
    private void doneTrackingNotify() {
            TrackerListener target = null;
            for (int x = 0; x < trackerListeners.size(); x++) {
                    target = (TrackerListener)
                    trackerListeners.elementAt(x);
                    target.doneTracking();
            }
    }
```

```
/**
 * Initializes a track.
 *
 * If the satellite is currently visiable, the start position
 * is NOW.
 * @param time
 *            The point in time at which the track should start
 *            the earliest. If the satellite is visible at that
 *            time - start then!
 * @return a Calendar object representing the time when the
 *         tracking will start. If the satellite is currently
 *         visible, the start time is now!
 * @since 1.0
 */
public Calendar initTrack(Calendar time) {
        boolean trackUnderway = false;
        //has the track we are initiating
        // already started

        Calendar nowDate = Calendar.getInstance(TimeZone
                        .getTimeZone("Europe/Copenhagen"));

        nextPassDate = predict.getNextPass(time);

        logger.logMessage(mod, level, "nextPassDate: "
                        + nextPassDate.getTime().toString());

        if (nextPassDate.before(nowDate)) {
                logger.logMessage(mod, level,
                                "Track is underway!");
                trackUnderway = true;
                nextPassDate = nowDate;
        }
        expectedPos =
        predict.getHorizontalPosition(nextPassDate);
        logger.logMessage(mod, level, "expectedPos: "
                        + expectedPos.toString());

        currentPos = motor.getPosition();
        logger.logMessage(mod, level, "currentPos: "
                        + currentPos.toString());

        Position stopPos = predict.getHorizontalPosition(predict
                        .getPassEnd(nextPassDate));
        positionTransformer = new GS232aPositionTransformer
                        (expectedPos, stopPos,
                        predict.getHorizontalPositionVelocity
                        (nextPassDate, 1000).velocity.Az > 0,
                        new Position(0, 0));
        motor.go(expectedPos);

        endPassDate = (Calendar) nextPassDate.clone();
        endPassDate.add(Calendar.MILLISECOND, (int) predict
                        .getPassDuration(nextPassDate));

        // start timer to check the antenna position
        timer = new Timer();
        TimerTask initChecker = new InitChecker();
        timer.scheduleAtFixedRate(initChecker, 1000, 1000);

        return nextPassDate;
}

/**
```

```
     * Bring the antenna into the start position for the next
     * track from now on. If the satellite is currently visible,
     * this track is assumed.
     * @return a Calendar object representing the time when the
     *         tracking will start. If the satellite is currently
     *         visible, the start time is now!
     * @since 1.0
     * @see #initTrack(Calendar)
     */
    public Calendar initTrack() {
            return initTrack(Calendar.getInstance(TimeZone
                            .getTimeZone("Europe/Copenhagen")));
    }

    /**
     * Start to track. This function waits for the satellite to
     * become visible and then tracks it. The velocities given
     * to the motordriver are updated periodically - likewise
     * is the antennas current position polled from the motordrive.
     * Doppler Correction is done and the calculated frequency sent
     * to the RadioDriver.
     * @since 1.0
     */
    public void startTrack() {

            TimerTask trackRunner = new TrackRunner();
            timer = new Timer();

            int initDelay = (int) (nextPassDate.getTimeInMillis()
                            - System.currentTimeMillis());
            logger.logMessage(mod, level, "init delay: "
                            + initDelay);
            if (initDelay < 0)
                    initDelay = 0;

            timer.scheduleAtFixedRate(trackRunner, initDelay,
                            period);

    }

    /**
     * Stop the current track. All registered TrackerListeners will
     * be notified using the doneTracking() callback.
     *
     * @since 1.0
     */
    public void stopTrack() {
            motor.close();
            radio.close();
            timer.cancel();
    }

    /**
     * Abort the current track. Call this to make sure all
     * connections are closed - fx. when abruptly quitting
     * the application... This is the same as cancelTrack()
     * but without notifying the TrackerListeners
     * @since 1.0
     * @see #cancelTrack()
     */
    public void abort() {
            motor.close();
            radio.close();
            if (timer != null)
                    //timer.stop();
```

```
                timer.cancel();
        }

        private Velocity getNewMotorVelocity(Position currentPos,
                        long nextTimerInterrupt, Position newPos) {
                double nextTimerInterruptSec =
                                (double) nextTimerInterrupt
                                / (double) 1000;
                return new Velocity((SubtractDegrees(newPos.Az,
                                        currentPos.Az))
                                / nextTimerInterruptSec,
                                (SubtractDegrees(newPos.El,
                                currentPos.El))
                                / nextTimerInterruptSec);
        }

        /**
         *
         */
        private Velocity getNewMotorVelocity2(Position currentPos,
                        long nextTimerInterrupt, Position newPos) {
                double nextTimerInterruptSec =
                                (double) nextTimerInterrupt
                                / (double) 1000;
                return new Velocity(
                (newPos.Az - currentPos.Az) / nextTimerInterruptSec,
                (newPos.El - currentPos.El) / nextTimerInterruptSec);
        }

        /**
         *
         * @param a
         * @param b
         * @return
         */
        private double SubtractDegrees(double a, double b) {
                double tmp1 = 360 - Math.abs(a - b);
                double tmp2 = Math.abs(a - b);
                return ((a - b) > 0 ? 1 : -1) *
                ((tmp1) < (tmp2) ? tmp1 : tmp2);
        }

        public boolean checkAntennaPosition(Position expectedPos)
                        throws TrackerException {
                Position currPos;

                currPos = motor.getPosition();
                logger.logMessage(mod, level, "currentPos: "
                                + currPos.toString());
                if (currPos.almostEquals(expectedPos)) {
                        return true;
                } else {
                        throw new TrackerException(
                "The antenna is not in the right position!");
                }
        }

        public Position getExpectedPos() {
                return expectedPos;
        }

        class InitChecker extends TimerTask {

                public void run() {
                        currentPos = motor.getPosition();
```

```
                if (currentPos.almostEquals(expectedPos)) {
                        logger.logMessage(mod, level,
        "Current postion is almost equal to expected position");
                        motor.stop();
                        timer.cancel();
                }
        }
}

class TrackRunner extends TimerTask {

        public void run() {
                if (endPassDate.after(Calendar.getInstance())) {
                        //we can still see the satellite
                        Calendar curDate =
                                Calendar.getInstance();
                        Calendar newDate =
                                (Calendar) curDate.clone();
                        newDate.add(Calendar.MILLISECOND,
                                        period);

                        PositionVelocity pv = predict
                        .getHorizontalPositionVelocity(
                                        newDate, 1000);
                        expectedPos = pv.position;
                        Velocity satVelocity = pv.velocity;

                        Position nextPos = positionTransformer
                                        .transform(predict
                                        .getHorizontalPosition
                                        (newDate));
                        expectedPos = positionTransformer
                                        .transform(predict
                                        .getHorizontalPosition
                                        (curDate));
                        expectedPos2 = predict
                        .getHorizontalPosition(curDate);
                        currentPos = motor.getPosition();
                        Velocity v = getNewMotorVelocity
                        (currentPos, period, nextPos);
                        Velocity v2 = getNewMotorVelocity2
                        (currentPos, period, nextPos);

                        System.out.println("Trans    Exp:"
                                        + expectedPos
                                        + " velocity:"
                                        + v);
                        System.out.println("NoTrans  Exp:"
                                        + expectedPos2
                                        + " velocity:" + v2);

                        System.out.println(v);
                        motor.goAzEl(v);

                        double recvFreq = sat.getRXFrequency();
                        double xmitFreq = sat.getTXFrequency();

                        // speedOfLight;
                        double dopplerFactor = satVelocity.Ds
                                        / speedOfLight;

                        if (recvFreq != 0) {
                                double tmp = recvFreq -
                                (dopplerFactor * recvFreq);
                                radio.setRXFrequency(tmp);
```

```
                                        System.out.println(tmp + " Hz");
                                }
                                if (xmitFreq != 0)
                                        radio.setTXFrequency(xmitFreq
                                        + (dopplerFactor * xmitFreq));

                                updateStatusListeners();
                        } else {
                                //we can't see the satellite anymore
                                stopTrack();
                        }
                }
        }
}
```

B.3 Mission Control Center

Listing B.36: Mission control Center (MCCClient.java)

```
/*
 * Created on Jul 26, 2005 @author s030972
 */

package mccClient;

import groundStation.session.Session;
import groundStation.gsManager.IMcc;

import java.rmi.*;
import java.rmi.registry.*;
import java.util.*;
import java.net.*;
import java.io.*;

public class MCCClient {

        public static void main(String[] args) {

                IMcc GSmgr;
                Registry registry;
                Session session;
                Calendar start;
                Calendar end;

                int port;
                ServerSocket server_socket;
                BufferedReader input;

                String serverAddress = "192.38.79.147";
                int serverPort = 4545;

                try {

                        if (System.getSecurityManager() == null) {
                                System.setSecurityManager
                                        (new RMISecurityManager());
                        }

                        registry = LocateRegistry
```

```
.getRegistry(serverAddress, serverPort);
System.out.println("Sending session to "
                + serverAddress + ":"
                + Integer.toString(serverPort));
// look up the remote object
GSmgr = (IMcc) (registry.lookup("GSManager"));

//setup the new session object
session = new Session();
session.setSessionName("test1");
session.setSessionID(1234);
start = Calendar.getInstance();
start.set(2005, 9, 9, 18, 25);
session.setStartTime(start);
end = Calendar.getInstance();
end.set(2005, 9, 9, 18, 50);
session.setEndTime(end);

//call the remote method
GSmgr.addSession(session);

// setup tcp connection
try {
        port = 1500;
        server_socket = new ServerSocket(port);
        System.out.println(
        "MCC Server waiting for client on port "
        + server_socket.getLocalPort());

        // MCC server infinite loop
        while (true) {
                Socket socket =
                        server_socket.accept();
                System.out.println(
                "New connection accepted "
                + socket.getInetAddress()
                + ":" + socket.getPort());
                input = new BufferedReader(new
                InputStreamReader(socket
                .getInputStream()));

                // print received data
                try {
                        while (true) {
                          String message =
                          input.readLine();
                          if (message == null)
                                  break;
                          System.out.println
                          (message);
                        }
                } catch (IOException e) {
                        System.out.println(
                        e.toString());
                }

                // connection closed by client
                try {
                        socket.close();
                        System.out.println(
                "Connection closed by client");
                } catch (IOException e) {
                        System.out.println(
                        e.toString());
                }
```

```
                        }

                    } catch (IOException e) {
                        System.out.println(e.toString());
                    }

                } catch (RemoteException e) {
                    e.printStackTrace();
                } catch (NotBoundException e) {
                    e.printStackTrace();
                } catch (Exception e) {
                    System.out.println("Error: " + e.toString());
                }
        }
}
```

B.4 Manual Operation

Listing B.37: Manual Operation (Operator.java)

```
/*
 * Created on Aug 9, 2005 @author s030972
 */

package operator;

import javax.swing.*;
import javax.swing.border.*;

import common.parameterVal.BoolParaVal;
import common.parameterVal.IntParaVal;
import common.parameterVal.NumParaVal;
import common.parameterVal.ParameterVal;
import common.parameterVal.StrParaVal;

import groundStation.gsManager.IOperator;
import groundStation.gsManager.GSManagerException;

import groundStation.log.ConsoleLog;
import groundStation.log.ILog;

//import javax.swing.border.TitledBorder;
import java.awt.*;
import java.awt.event.*;
import java.rmi.*;
import java.rmi.registry.*;
import java.util.*;

public class Operator extends JFrame implements ActionListener {

        private JButton startProtocol, startTracker;

        private JButton stopProtocol, stopTracker;

        private JButton killSession;

        private JButton getProStatus;

        private JButton getTracStatus;
```

```
private JLabel statusBar;

private TextArea statusArea;

private IOperator operator;

private String status;

private ILog logger;

private int mod;

private int level;

public Operator() {

        logger = (ILog) new ConsoleLog(mod, level);
        mod = logger.OPERATOR;
        level = logger.DEBUG;

        status = "";
        setTitle("Manual Control Clinet");
        Container container = getContentPane();
        container.setLayout(new BorderLayout());

        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(4, 2, 20, 20));

        p1.add(startProtocol = new JButton());
        p1.add(startTracker = new JButton());
        p1.add(stopProtocol = new JButton());
        p1.add(stopTracker = new JButton());
        p1.add(killSession = new JButton());
        p1.add(getProStatus = new JButton());
        p1.add(getTracStatus = new JButton());

        p1.setBorder(new TitledBorder("Manual Control Buttons"));

        JPanel p2 = new JPanel();
        p2.setLayout(new BorderLayout());
        p2.add(p1, BorderLayout.CENTER);

        container.add(p2, BorderLayout.EAST);
        statusArea = new TextArea("Status displayed here.");
        container.add(statusArea, BorderLayout.CENTER);
        statusBar = new JLabel("Status Bar....");
        container.add(statusBar, BorderLayout.SOUTH);

        //set button text
        startProtocol.setText("Start Protocol");
        startTracker.setText("Start Tracker");
        stopProtocol.setText("Stop Protocol");
        stopTracker.setText("Stop Tracker");
        killSession.setText("Kill Session");
        getProStatus.setText("Get Protocol Status");
        getTracStatus.setText("Get Tracker Status");

        startProtocol.addActionListener(this);
        startTracker.addActionListener(this);
        stopProtocol.addActionListener(this);
        stopTracker.addActionListener(this);
        killSession.addActionListener(this);
        getProStatus.addActionListener(this);
        getTracStatus.addActionListener(this);
```

```
        setSize(600, 500);

        Dimension screenSize = Toolkit.getDefaultToolkit()
                                .getScreenSize();
        int screenWidth = screenSize.width;
        int screenHeight = screenSize.height;

        Dimension frameSize = getSize();
        int x = (screenWidth - frameSize.width) / 2;
        int y = (screenHeight - frameSize.height) / 2;

        setLocation(x, y);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

    private void startProcButton() {

        System.out.println("Start Protocol!");
        try {
                operator.startProtocol();
        } catch (RemoteException e) {
                logger.logMessage(mod, level, e.toString());
        }

    }

    private void stopProcButton() {

        System.out.println("Stop Protocol!");
        try {
                operator.stopProtocol();
        } catch (RemoteException e) {
                logger.logMessage(mod, level, e.toString());
        }
    }

    private void startTracButton() {

        System.out.println("Start Tracker!");
        try {
                operator.startTracker();
        } catch (RemoteException e) {
                logger.logMessage(mod, level, e.toString());
        }
    }

    private void stopTracButton() {

        System.out.println("Stop Tracker!");
        try {
                operator.stopTracker();
        } catch (RemoteException e) {
                logger.logMessage(mod, level, e.toString());
        }
    }

    public void killSessionButton() {
        System.out.println("kill session");
        try {
                operator.killCurrSession();
        } catch (RemoteException e) {
                logger.logMessage(mod, level, e.toString());
        } catch (GSManagerException e) {
```

```
                statusBar.setText(e.toString());
                logger.logMessage(mod, level, e.toString());
        }
}

public void getProStatusButton() {
        ArrayList tmpList;
        int i;
        ParameterVal tmpVal;
        StrParaVal strVal;
        NumParaVal numVal;
        IntParaVal intVal;
        BoolParaVal boolVal;
        //String status = "";

        System.out.println("Get Protocol Status");
        try {
                tmpList = operator.getProtocolStatus();
                for (i = 0; i < tmpList.size(); i++) {
                        tmpVal = (ParameterVal)
                                (tmpList.get(i));
                        if (tmpVal instanceof StrParaVal) {
                                strVal = (StrParaVal)
                                        (tmpList.get(i));
                                status = status + "\n" + strVal
                                        .timeStamp.toString()
                                        + " : " + strVal.path
                                        .strPath + " : "
                                        + strVal.strValue;
                        }

                        if (tmpVal instanceof NumParaVal) {
                                numVal = (NumParaVal)
                                        (tmpList.get(i));
                                status = status + "\n" + numVal.
                                        timeStamp.toString()
                                        + " : " + numVal.path
                                        .strPath + " : "
                                        + numVal.numValue
                                        + " " + numVal.unit
                                        + " : Accuracy +- "
                                        + numVal.accuracy
                                        + " ";
                        }

                        if (tmpVal instanceof IntParaVal) {
                                intVal = (IntParaVal)
                                        (tmpList.get(i));
                                status = status + "\n" + intVal.
                                        timeStamp.toString()
                                        + " : " + intVal.path
                                        .strPath + " : "
                                        + intVal.intValue + " "
                                        + intVal.unit;
                        }

                        if (tmpVal instanceof BoolParaVal) {
                                boolVal = (BoolParaVal)
                                        (tmpList.get(i));
                                status = status + "\n" + boolVal
                                        .timeStamp.toString()
                                        + " : " + boolVal.path
                                        .strPath + " : "
                                        + boolVal.boolValue;
                        }
```

```
                }

        } catch (RemoteException e) {
                logger.logMessage(mod, level, e.toString());
        }
}

public void refreshStatus() {

        status = "";
        getProStatusButton();
        status = status + "\n";
        getTracStatusButton();
        statusArea.setText(status);

}

public void getTracStatusButton() {
        ArrayList tmpList;
        int i;
        ParameterVal tmpVal;
        StrParaVal strVal;
        NumParaVal numVal;
        IntParaVal intVal;
        BoolParaVal boolVal;

        System.out.println("Get Tracker Status");
        try {
                tmpList = operator.getTrackerStatus();

                for (i = 0; i < tmpList.size(); i++) {

                        tmpVal = (ParameterVal) (tmpList.get(i));
                        if (tmpVal instanceof StrParaVal) {
                                strVal = (StrParaVal)
                                        (tmpList.get(i));
                                status = status + "\n" + strVal
                                        .timeStamp.toString()
                                        + " : " + strVal.path
                                        .strPath + " : "
                                        + strVal.strValue;
                        }

                        if (tmpVal instanceof NumParaVal) {
                                numVal = (NumParaVal)
                                        (tmpList.get(i));
                                status = status + "\n" + numVal
                                        .timeStamp.toString()
                                        + " : " + numVal.path
                                        .strPath + " : "
                                        + numVal.numValue + " "
                                        + numVal.unit
                                        + " : Accuracy +- "
                                        + numVal.accuracy + " ";
                        }

                        if (tmpVal instanceof IntParaVal) {
                                intVal = (IntParaVal)
                                        (tmpList.get(i));
                                status = status + "\n" + intVal
                                        .timeStamp.toString()
                                        + " : " + intVal.path
                                        .strPath + " : "
                                        + intVal.intValue + " "
                                        + intVal.unit;
```

```
                        }

                        if (tmpVal instanceof BoolParaVal) {
                                boolVal = (BoolParaVal)
                                        (tmpList.get(i));
                                status = status + "\n" + boolVal
                                        .timeStamp.toString()
                                        + " : " + boolVal.path
                                        .strPath + " : "
                                        + boolVal.boolValue;
                        }
                }

        } catch (RemoteException e) {
                logger.logMessage(mod, level, e.toString());
        }
}

public void actionPerformed(ActionEvent e) {

        if (e.getSource() == startProtocol) {
                startProcButton();
        }

        else if (e.getSource() == stopProtocol) {
                stopProcButton();
        }

        else if (e.getSource() == startTracker) {
                startTracButton();
        }

        else if (e.getSource() == startTracker) {
                stopTracButton();
        }

        else if (e.getSource() == getProStatus) {
                getProStatusButton();
        }

        else if (e.getSource() == getTracStatus) {
                getTracStatusButton();
        }

        else if (e.getSource() == killSession) {
                killSessionButton();
        }

}

private void getRegistry() {

        Registry registry;

        String serverAddress = "192.38.79.147";
        int serverPort = 4545;

        try {

                if (System.getSecurityManager() == null) {
                        System.setSecurityManager(new
                                RMISecurityManager());
                }

                registry = LocateRegistry.getRegistry
```

```
                        (serverAddress, serverPort);
                System.out.println("Sending session to "
                        + serverAddress + ":"
                        + Integer.toString(serverPort));

                operator = (IOperator)
                        (registry.lookup("GSManager"));

            } catch (RemoteException e) {
                logger.logMessage(mod, level, e.toString());
            } catch (NotBoundException e) {
                logger.logMessage(mod, level, e.toString());
            } catch (Exception e) {
                logger.logMessage(mod, level, e.toString());
            }
        }

        public static void main(String[] args) {

            StatusMonitor statusMon;

            System.out.println("Starting...");
            Operator opr = new Operator();
            opr.pack();
            opr.setVisible(true);
            opr.getRegistry();

            statusMon = new StatusMonitor(opr);
            statusMon.run();

        }
}
```

Appendix C

Testing

C.1 Test Cases of Unit Testing

C.1.1 Session Queue

Insert Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	session start time earlier than current time. start time = 2005, 10, 3, 9, 30 current time = Mon Oct 03 10:04:58 CEST 2005	The insert session operation should fail.	groundStation. session. SessionQueueException: Add new session failed: The start time of new session is outdate	Match, the operation failed.

Table C.1: Test Case 1 for Insert Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	session start time is not earlier than current time but close to it. The interval is 5 min. start time = 2005, 10, 3, 10, 14 current time = Oct 03 10:09:45	The insert session operation should fail.	groundStation. session. SessionQueueException: Add new session failed: The start time of new session is outdate	Match, the operation failed.

Table C.2: Test Case 2 for Insert Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	overlap with current sessions in the queue session in the queue: start = 2005, 10, 3, 13, 14, end = 2005, 10, 3, 14, 50 new session: start = 2005, 10, 3, 12, 14, end = 2005, 10, 3, 14, 14,	The insert session operation should fail.	groundStation. session. SessionQueueException: Add new session failed: The new session is overlap with some existing sessions	Match, the operation failed.

Table C.3: Test Case 3 for Insert Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	overlap with current sessions in the queue session in the queue: start = 2005, 10, 3, 13, 14, end = 2005, 10, 3, 14, 50 new session: start = 2005, 10, 3, 14, 14 end = 2005, 10, 3, 15, 50	The insert session operation should fail.	groundStation. session. SessionQueueException: Add new session failed: The new session is overlap with some existing sessions	Match, the operation failed.

Table C.4: Test Case 4 for Insert Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	overlap with current sessions in the queue session in the queue: start = 2005, 10, 3, 13, 14, end = 2005, 10, 3, 14, 50 new session: start = 2005, 10, 3, 13, 30 end = 2005, 10, 3, 14, 30	The insert session operation should fail.	groundStation. session. SessionQueueException: Add new session failed: The new session is overlap with some existing sessions	Match, the operation failed.

Table C.5: Test Case 5 for Insert Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	overlap with current sessions in the queue session in the queue: start = 2005, 10, 3, 13, 14, end = 2005, 10, 3, 14, 50 new session: start = 2005, 10, 3, 12, 14 end = 2005, 10, 3, 15, 50	The insert session operation should fail.	groundStation. session. SessionQueueException: Add new session failed: The new session is overlap with some existing sessions	Match, the operation failed.

Table C.6: Test Case 6 for Insert Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	no session in the queue current time = Oct 03 10:29:45 CEST 2005 starttime = 2005, 10, 3, 12, 14 end = 2005, 10, 3, 15, 50	The insert session operation should succeed.	Mon Oct 03 10:29:45 CEST 2005 : New Session id is test1	Match, the operation succeed.

Table C.7: Test Case 7 for Insert Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	session “test1” in the queue: start = 2005, 10, 3, 13, 14, end = 2005, 10, 3, 14, 50 insert session “test2” start = 2005, 10, 3, 15, 14, end = 2005, 10, 3, 16, 50	The insert session operation should succeed.	Mon Oct 03 10:37:01 CEST 2005 : New Session id is test2 Still in Session Queue: test1 Still in Session Queue: test2	Match, the operation succeeded.

Table C.8: Test Case 8 for Insert Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	session name is same. session name in the queue = test1. new session name = test1	The insert session operation should fail.	groundStation. session. SessionQueueException: Add new session failed: The Session name is duplicated with a existing session.	Match, the operation failed.

Table C.9: Test Case 9 for Insert Session

Remove Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	remove session: session name is not in the queue. session name in the queue: test1. remove session test2	The remove session operation should fail.	groundStation. session. SessionQueueException: Remove Sesssion failed: The Session name does not exist in session queue	Match, the operation failed.

Table C.10: Test Case 1 for Remove Session

Module Overview	Input	Expected Output	Real Output	Result
Session Queue	remove session: the name is in the queue. session name in the queue: test1. remove session test1	The remove session operation should succeed.	Removed session successfully: test1	Match, the operation succeeded.

Table C.11: Test Case 2 for Remove Session

C.1.2 Ground Station Manager

Register GSM and Start Session Dispatcher

Module Overview	Input	Expected Output	Real Output	Result
Ground Station Manager	Start up ground station manager server.	The GSM services should be registered. The session Dispatcher should start.	GSManager is running! Mon Oct 03 10:36:42 CEST 2005 : this address = heidi / 192.38.79.147, this port = 4545 Mon Oct 03 10:36:42 CEST 2005 : Dispatcher is running. The Dispatcher is checking 0 times	Match, the operation succeeded.

Table C.12: Test Case 1 for Ground Station Manager

Add Session

Module Overview	Input	Expected Output	Real Output	Result
Ground Station Manager	Add Session	The new session should be added into session queue.	This function has been tested in the unit testing of Session Queue.	Match, the operation succeeded.

Table C.13: Test Case 2 for Ground Station Manager

Remove Session

Module Overview	Input	Expected Output	Real Output	Result
Ground Station Manager	Remove Session	The session should be removed from session queue.	This function has been tested in the unit testing of Session Queue.	Match, the operation succeeded.

Table C.14: Test Case 3 for Ground Station Manager

Start Session Controller

Module Overview	Input	Expected Output	Real Output	Result
Ground Station Manager	Start session controller	The session controller should be started by GSM.	start session controller. session in queue: start = 2005, 10, 3, 10, 50 end = 2005, 10, 3, 11, 50 current time = Oct 03 10:44:49 CEST 2005 Session Controller: The session controller thread is running!	Match, the operation succeeded.

Table C.15: Test Case 4 for Ground Station Manager

C.1.3 Protocol

The Connection with MCC

Module Overview	Input	Expected Output	Real Output	Result
Protocol	Connect with MCC	The protocol thread should setup the TCP connection with MCC	connection with MCC. Protocol: Connected with MCC server heidi.heidi.com / 192.38.79.147:1500	Match, the operation succeeded.

Table C.16: Test Case 1 for Protocol

Get The Status of Protocol

Module Overview	Input	Expected Output	Real Output	Result
Protocol	Get status of protocol	The protocol thread should return its current status.	Protocol Status: in step 1.	Match, the operation succeeded.

Table C.17: Test Case 2 for Protocol

Read File

Module Overview	Input	Expected Output	Real Output	Result
Protocol	Read file	The protocol thread should read the file content and return it to MCC.	this is from read Serial simulation. send the message to MCC client. MCC client: the first ax25 data, the second ax25 data	Match, the operation succeeded.

Table C.18: Test Case 3 for Protocol

Write File

Module Overview	Input	Expected Output	Real Output	Result
Protocol	Write file	The protocol thread should write the messages to the file.	this is from write Serial simulation. write a message into file "ax25.write". print file "ax25.write". It is from protocol message	Match, the operation succeeded.

Table C.19: Test Case 4 for Protocol

C.1.4 Session Controller

Start Session

Module Overview	Input	Expected Output	Real Output	Result
Session Controller	Start Session. start time (2005, 10, 3, 11, 10), end time(2005, 10, 3, 11, 50),	The session thread should start by session controller.	Oct 03 10:56:09 CEST 2005 : Session: The Session thread is running.	Match, the operation succeeded.

Table C.20: Test Case 1 for Session Controller

C.1.5 Session

Start Tracker

Module Overview	Input	Expected Output	Real Output	Result
Session	Start tracker: session start time (2005, 10, 3, 11, 10); end time(2005, 10, 3, 11, 50);	The Tracker should start.	Mon Oct 03 11:01:31 CEST 2005 : setting satellite. Mon Oct 03 11:01:31 CEST 2005 : Satellite initialized. Mon Oct 03 11:01:31 CEST 2005 : starting initTrack	Match, the operation succeeded.

Table C.21: Test Case 1 for Session

Start Protocol

Module Overview	Input	Expected Output	Real Output	Result
Session	Start Protocol: session start time (2005, 10, 3, 11, 10); end time(2005, 10, 3, 11, 50);	The protocol should start.	Mon Oct 03 11:01:21 CEST 2005 : starting protocol	Match, the operation succeeded.

Table C.22: Test Case 2 for Session

C.1.6 Mission Control Center

Add Session

Module Overview	Input	Expected Output	Real Output	Result
Mission Control Center	Add Session	The session should send to GSM.	Sending session to 192.38.79.147:4545	Match, the operation succeeded.

Table C.23: Test Case 1 for Mission Control Center

TCP Server

Module Overview	Input	Expected Output	Real Output	Result
Mission Control Center	TCP server.	The mcc should setup the TCP server and accept connections from clients.	MCC Server waiting for client on port 1500. New connection accepted /192.38.79.147 : 32973	Match, the operation succeeded.

Table C.24: Test Case 2 for Mission Control Center

C.1.7 Manual Operation

Kill Session

Module Overview	Input	Expected Output	Real Output	Result
Manual Operation	Kill session if session is running.	The session should be killed.	Mon Oct 03 11:08:26 CEST 2005 : Session (stopSession): Session has been interrupted. Mon Oct 03 11:08:26 CEST 2005 : Session: sleep interrupted. Mon Oct 03 11:08:26 CEST 2005 : Session: The Session thread terminated!	Match, the operation succeeded.

Table C.25: Test Case 1 for Manual Operation

Module Overview	Input	Expected Output	Real Output	Result
Manual Operation	Kill session if session is not running.	The error should occur.	kill session. Mon Oct 03 11:09:46 CEST 2005 : groundStation. gsManager. GSManagerException: No session found!	Match, the operation succeeded.

Table C.26: Test Case 2 for Manual Operation

Get Protocol Status

Module Overview	Input	Expected Output	Real Output	Result
Manual Operation	Get protocol status	The manual operation should get the status of protocol.	Protocol Status. Oct 03 11:13:25 CEST 2005 : Protocol.Type : AX25. Mon Oct 03 11:13:25 CEST 2005 : Protocol.StatusMsg : null	Match, the operation succeeded.

Table C.27: Test Case 3 for Manual Operation

Get Tracker Status

Module Overview	Input	Expected Output	Real Output	Result
Manual Operation	Get tracker status	The manual operation should get the status of tracker.	Tracker Status. Oct 03 11:13:25 CEST 2005 : Tracker. CurrentPosition : 015.00^{o}Az 00.00^{o}El. Mon Oct 03 11:13:25 CEST 2005 : Tracker. ExpectedPosition : 017.46^{o}Az -00.00^{o}El 03375.9km	Match, the operation succeeded.

Table C.28: Test Case 4 for Manual Operation

C.2 Test Cases of Integration Testing

C.2.1 Automation Session Testing

Test Case 1

- The mission control center sends a new session into session queue and removes it from session queue afterwards.

  The client side:
  heidi src # ./runMCC
  Sending session to 192.38.79.147:4545
  MCC Server waiting for client on port 1500

  The server side:
  heidi src # ./runGSControl
  GSManager is running!
  Sun Oct 09 12:56:19 CEST 2005 : this address = heidi/192.38.79.147, this port = 4545

  Sun Oct 09 12:56:19 CEST 2005 : Dispatcher is running.

  The Dispatcher is checking 0 times
  Sun Oct 09 12:56:24 CEST 2005 : New Session id is test1

  Still in Session Queue: test1
  Removed session successfully: test1

Test Case 2

- The mission control center sends a new session into session queue. The GSM starts up the session controller. The session controller start up the session thread. The session can not finish its tasks in its duration. The session controller wakes up and force to stop the session thread.

  The client side:
  heidi src # ./runMCC
  Sending session to 192.38.79.147:4545
  MCC Server waiting for client on port 1500

New connection accepted /192.38.79.147:32788
the first ax25 data
the second ax25 data
Connection closed by client

The server side:
heidi src # ./runGSControl
GSManager is running!
Sun Oct 09 12:58:25 CEST 2005 : this address = heidi/192.38.79.147, this port = 4545

Sun Oct 09 12:58:25 CEST 2005 : Dispatcher is running.

The Dispatcher is checking 0 times
Sun Oct 09 12:58:29 CEST 2005 : New Session id is test1

Still in Session Queue: test1
The Session Queue has 1 sessions waiting
The Dispatcher is checking 1 times
Sun Oct 09 12:58:35 CEST 2005 : Session Controller: The session controller thread is running!

Sun Oct 09 12:58:35 CEST 2005 : the session period: 5100001

Sun Oct 09 12:58:35 CEST 2005 : Session Controller: The thread is waiting!

Sun Oct 09 12:58:35 CEST 2005 : Session: The Session thread is running!

Sun Oct 09 12:58:35 CEST 2005 : starting protocol

Protocol: Connected with MCC server heidi.heidi.com/192.38.79.147:1500
this is from write Serial simulation
this is from read Serial simulation
Sun Oct 09 12:58:39 CEST 2005 : Session(stopSession): Session has been interrupted.

Sun Oct 09 12:58:39 CEST 2005 : Session Controller: The thread is terminated!

Sun Oct 09 12:58:39 CEST 2005 : Session: sleep interrupted

Sun Oct 09 12:58:39 CEST 2005 : Session: The Session thread terminated!

Test Case 3

- The mission control center sends a new session into session queue. The GSM starts up the session controller. The session controller starts up the session threads. The session finishes all tasks in its duration. The session wakes up the session controller and finishes this pass.

The client side:
heidi src # ./runMCC

Sending session to 192.38.79.147:4545

MCC Server waiting for client on port 1500

New connection accepted /192.38.79.147:32796

the first ax25 data

the second ax25 data

Connection closed by client

The server side:
heidi src # ./runGSControl

GSManager is running!

Sun Oct 09 13:00:52 CEST 2005 : this address = heidi/192.38.79.147, this port = 4545

Sun Oct 09 13:00:52 CEST 2005 : Dispatcher is running.

The Dispatcher is checking 0 times

Sun Oct 09 13:00:54 CEST 2005 : New Session id is test1

Still in Session Queue: test1

The Session Queue has 1 sessions waiting

The Dispatcher is checking 1 times

Sun Oct 09 13:01:02 CEST 2005 : Session Controller: The session controller thread is running!

Sun Oct 09 13:01:02 CEST 2005 : the session period: 5100001

Sun Oct 09 13:01:02 CEST 2005 : Session Controller: The thread is waiting!

Sun Oct 09 13:01:02 CEST 2005 : Session: The Session thread is running!

Sun Oct 09 13:01:02 CEST 2005 : starting protocol

Protocol: Connected with MCC server heidi.heidi.com/192.38.79.147:1500

this is from write Serial simulation

this is from read Serial simulation

The Dispatcher is checking 2 times

Sun Oct 09 13:01:12 CEST 2005 : Session: The Session thread terminated!

Sun Oct 09 13:01:12 CEST 2005 : Session Controller: The thread is terminated!

Test Case 4

- The mission control center sends a new session into session queue. The GSM starts up the session controller. The session controller starts up the session threads. The operator kills the current running session.

  The client side:
  heidi src # ./runMCC

  Sending session to 192.38.79.147:4545

  MCC Server waiting for client on port 1500

  heidi src # ./runOperator

  Starting...

  Sending command to 192.38.79.147:4545

  kill session

  The server side:
  heidi src # ./runGSControl

  GSManager is running!

  Sun Oct 09 13:02:56 CEST 2005 : this address = heidi/192.38.79.147, this port =4545

  Sun Oct 09 13:02:57 CEST 2005 : Dispatcher is running.

  The Dispatcher is checking 0 times

  Sun Oct 09 13:03:02 CEST 2005 : New Session id is test1

  Still in Session Queue: test1

  The Session Queue has 1 sessions waiting

  The Dispatcher is checking 1 times

  Sun Oct 09 13:03:07 CEST 2005 : Session Controller: The session controller thread is running!

  Sun Oct 09 13:03:07 CEST 2005 : the session period: 5100000

  Sun Oct 09 13:03:07 CEST 2005 : Session Controller: The thread is waiting!

  Sun Oct 09 13:03:07 CEST 2005 : Session: The Session thread is running!

  Sun Oct 09 13:03:07 CEST 2005 : starting protocol

  this is from write Serial simulation

  this is from read Serial simulation

  Sun Oct 09 13:03:12 CEST 2005 : Session(stopSession): Session has been interrupted.

  Sun Oct 09 13:03:12 CEST 2005 : Session: sleep interrupted

  Sun Oct 09 13:03:12 CEST 2005 : Session: The Session thread terminated!

  The Dispatcher is checking 2 times

- The mission control center sends a new session into session queue. The GSM starts up the session controller. The session controller starts up the session threads. The session starts the tracking system.

  The client side:
  heidi src # ./runMCC

  Sending session to 192.38.79.147:4545

  MCC Server waiting for client on port 1500

  New connection accepted /192.38.79.147:32816

  the first ax25 data

  the second ax25 data

  Connection closed by client

  The server side:
  Sun Oct 09 13:04:36 CEST 2005 :

  setting satellite

  Sun Oct 09 13:04:37 CEST 2005 : Satellite initialized

  Sun Oct 09 13:04:37 CEST 2005 : starting initTrack

  Sun Oct 09 13:04:37 CEST 2005 : nextPassDate: Wed Nov 09 14:31:39 CET 2005

  Sun Oct 09 13:04:37 CEST 2005 : expectedPos: 082.89^{o}Az 00.00^{o}El 03361.3km

  Sun Oct 09 13:04:37 CEST 2005 : currentPos: 000.00^{o}Az 00.00^{o}El

  Type: 2

  Running speed: 5.1429

  The Dispatcher is checking 3 times

  Sun Oct 09 13:04:53 CEST 2005 : Current postion is almost equal to expected position

  The Dispatcher is checking 4 times

  The Dispatcher is checking 5 times

  Sun Oct 09 13:05:07 CEST 2005 : 082.89^{o}Az 00.00^{o}El 03361.3km

  Sun Oct 09 13:05:07 CEST 2005 : currentPos: 082.00^{o}Az 00.00^{o}El

  Sun Oct 09 13:05:07 CEST 2005 : init delay: -1607775580

  Sun Oct 09 13:05:07 CEST 2005 : Session Controller: The thread is terminated!

Sun Oct 09 13:05:07 CEST 2005 : Session: The Session thread terminated!

Trans Exp:267.47^{o}Az -41.03^{o}El velocity:+872.6697^{o}Az -205.1625^{o}El
NoTrans Exp:267.47^{o}Az -41.03^{o}El 09552.6km velocity:+927.3303^{o}Az -205.1625^{o}El
+872.6697^{o}Az -205.1625^{o}El
Running speed: 5.1429
249.9961035920312 Hz

Trans Exp:267.46^{o}Az -41.04^{o}El velocity:+872.7155^{o}Az -205.2278^{o}El
NoTrans Exp:267.46^{o}Az -41.04^{o}El 09554.5km velocity:+927.2845^{o}Az -205.2278^{o}El
+872.7155^{o}Az -205.2278^{o}El
Running speed: 5.1429
249.99610436802598 Hz

Trans Exp:267.46^{o}Az -41.04^{o}El velocity:+872.7158^{o}Az -205.2283^{o}El
NoTrans Exp:267.46^{o}Az -41.04^{o}El 09554.5km velocity:+927.2842^{o}Az -205.2283^{o}El
+872.7158^{o}Az -205.2283^{o}El
Running speed: 5.1429
249.9961043739207 Hz

C.2.2 Manual Control Testing

The integrated testing of the manual control should concentrate to test all the functionalities which are provide by the manual operation GUI interface. The table C.29 shows the all possible test cases which have to be tested in the manual control.

	If session is not running	If session is running
Start Protocol	Test case 1	Test case 8
Stop Protocol	Test case 2	Test case 9
Start Tracker	Test case 3	Test case 10
Stop Tracker	Test case 4	Test case 11
Get Protocol Status	Test case 5	Test case 12
Get Tracker Status	Test case 6	Test case 13
Kill Session	Test case 7	Test case 14

Table C.29: Manual Control Testing

Test Case 1

Start the protocol when the session is not running.

- **The client side**:
  heidi src # ./runMCC

Sending session to 192.38.79.147:4545

MCC Server waiting for client on port 1500

New connection accepted /192.38.79.147:32790

the first ax25 data

the second ax25 data

Connection closed by client

heidi src # ./runOperator

Starting...

Sending session to 192.38.79.147:4545

Start Protocol!

The server side:
heidi src # ./runGSControl

GSManager is running!

Mon Oct 10 21:44:24 CEST 2005 : this address = heidi/192.38.79.147, this port = 4545

Mon Oct 10 21:44:25 CEST 2005 : Dispatcher is running.

The Dispatcher is checking 0 times

Mon Oct 10 21:44:34 CEST 2005 : Starting Protocol!

Protocol from server.

Protocol: Connected with MCC server heidi.heidi.com/192.38.79.147:1500

this is from write Serial simulation

this is from read Serial simulation

Test Case 2

Stop the protocol when the session is not running.

- **The client side**:
  heidi src # ./runOperator

  Starting...

  Sending session to 192.38.79.147:4545

  Start Protocol!

  Stop Protocol!

  The server side:
  heidi src # ./runGSControl

GSManager is running!

Mon Oct 10 22:00:02 CEST 2005 : this address = heidi/192.38.79.147, this port = 4545

Mon Oct 10 22:00:02 CEST 2005 : Dispatcher is running.

The Dispatcher is checking 0 times

The Session Queue has 0 sessions waiting

The Dispatcher is checking 1 times

Mon Oct 10 22:00:16 CEST 2005 : Starting Protocol!

Protocol from server.

Protocol: Connected with MCC server heidi.heidi.com/192.38.79.147:1500

this is from write Serial simulation

this is from read Serial simulation

Mon Oct 10 22:00:19 CEST 2005 : Protocol(stopProtocol): Protocol has been stopped.

Test Case 3

Start the tracker when the session is not running.

- **The client side**:
  heidi src # ./runOperator

  Starting...

  Sending session to 192.38.79.147:4545

  Start Tracker!

  The server side:
  heidi src # ./runGSControl

  GSManager is running!

  Mon Oct 10 22:33:50 CEST 2005 : this address = heidi/192.38.79.147, this port = 4545

  Mon Oct 10 22:33:52 CEST 2005 : Dispatcher is running.

  The Dispatcher is checking 0 times

  The Session Queue has 0 sessions waiting

  Mon Oct 10 22:34:34 CEST 2005 : setting satellite

  Mon Oct 10 22:34:35 CEST 2005 : Satellite initialized

  Mon Oct 10 22:34:35 CEST 2005 : starting initTrack

Mon Oct 10 22:34:35 CEST 2005 : nextPassDate: Fri Nov 11 03:59:35 CET 2005

Mon Oct 10 22:34:36 CEST 2005 : expectedPos: 042.19^{o}Az -00.00^{o}El 03383.7km

Mon Oct 10 22:34:36 CEST 2005 : currentPos: 000.00^{o}Az 00.00^{o}El

Type: 0 Running speed: 5.1429

Mon Oct 10 22:34:44 CEST 2005 : Current postion is almost equal to expected position

Mon Oct 10 22:34:46 CEST 2005 : 042.19^{o}Az -00.00^{o}El 03383.7km

Mon Oct 10 22:34:46 CEST 2005 : currentPos: 042.00^{o}Az 00.00^{o}El

Mon Oct 10 22:34:46 CEST 2005 : init delay: -1593478313

Trans Exp:015.85^{o}Az -22.75^{o}El 06645.1km velocity:-130.6988^{o}Az -113.7623^{o}El
NoTrans Exp:015.85^{o}Az -22.75^{o}El 06645.1km velocity:-130.6988^{o}Az -113.7623^{o}El
-130.6988^{o}Az -113.7623^{o}El
Running speed: 0.0
249.996192213392 Hz

Trans Exp:015.97^{o}Az -22.82^{o}El 06656.4km velocity:-130.1204^{o}Az -114.1055^{o}El
NoTrans Exp:015.97^{o}Az -22.82^{o}El 06656.4km velocity:-130.1204^{o}Az -114.1055^{o}El
-130.1204^{o}Az -114.1055^{o}El
Running speed: 0.0
249.99619158949133 Hz

Trans Exp:015.97^{o}Az -22.82^{o}El 06656.6km velocity:-130.1075^{o}Az -114.1131^{o}El
NoTrans Exp:015.97^{o}Az -22.82^{o}El 06656.6km velocity:-130.1075^{o}Az -114.1131^{o}El
-130.1075^{o}Az -114.1131^{o}El
Running speed: 0.0
249.9961915758471 Hz

Test Case 4

Stop the tracker when the session is not running.

- **The client side**:
  heidi src # ./runOperator

  Starting...

  Sending session to 192.38.79.147:4545

  Start Tracker!

  Stop Tracker!

The server side:
heidi src # ./runGSControl

GSManager is running!

Mon Oct 10 22:46:21 CEST 2005 : this address = heidi/192.38.79.147, this port = 4545

Mon Oct 10 22:46:21 CEST 2005 : Dispatcher is running.

The Dispatcher is checking 0 times

The Session Queue has 0 sessions waiting

Mon Oct 10 22:46:32 CEST 2005 : setting satellite

Mon Oct 10 22:46:33 CEST 2005 : Satellite initialized

Mon Oct 10 22:46:33 CEST 2005 : starting initTrack

Mon Oct 10 22:46:33 CEST 2005 : nextPassDate: Fri Nov 11 03:59:35 CET 2005

Mon Oct 10 22:46:33 CEST 2005 : expectedPos: 042.19^{o}Az -00.00^{o}El 03383.7km

Mon Oct 10 22:46:33 CEST 2005 : currentPos: 000.00^{o}Az 00.00^{o}El

Type: 0

Running speed: 5.1429

Mon Oct 10 22:46:41 CEST 2005 : Current postion is almost equal to expected position

Mon Oct 10 22:46:43 CEST 2005 : 042.19^{o}Az -00.00^{o}El 03383.7km

Mon Oct 10 22:46:43 CEST 2005 : currentPos: 041.00^{o}Az 00.00^{o}El

Mon Oct 10 22:46:43 CEST 2005 : init delay: -1594195004

Trans Exp:044.23^{o}Az -42.32^{o}El 09755.7km velocity:+16.1839^{o}Az -211.6019^{o}El
NoTrans Exp:044.23^{o}Az -42.32^{o}El 09755.7km velocity:+16.1839^{o}Az -211.6019^{o}El
+16.1839^{o}Az -211.6019^{o}El
Running speed: 5.1429
249.99677808940183 Hz

Trans Exp:044.24^{o}Az -42.32^{o}El 09756.3km velocity:+16.2137^{o}Az -211.6231^{o}El
NoTrans Exp:044.24^{o}Az -42.32^{o}El 09756.3km velocity:+16.2137^{o}Az -211.6231^{o}El
+16.2137^{o}Az -211.6231^{o}El
Running speed: 5.1429
249.9967783376981 Hz

Mon Oct 10 22:46:44 CEST 2005 : Try to stop Tracker

stoptrack() in Tracker module.

Mon Oct 10 22:46:44 CEST 2005 : Tracker has been stopped!

Test Case 5 and Test Case 6

Get the status of the protocol and the tracker when the session is not running.

- **The client side**:
  heidi src # ./runOperator

  Starting...
  Sending session to 192.38.79.147:4545
  Get Protocol Status
  Get Tracker Status

  Wed Oct 12 16:16:00 CEST 2005 : Protocol.Type : AX25
  Wed Oct 12 16:16:00 CEST 2005 : Protocol.StatusMsg : The protocol thread is in the first step!

  Wed Oct 12 16:16:00 CEST 2005 : Tracker.CurrentPosition : 050.00^{o}Az 00.00^{o}El
  Wed Oct 12 16:16:00 CEST 2005 : Tracker.ExpectedPosition : 147.59^{o}Az 00.00^{o}El 03345.5km

  Get Protocol Status
  Get Tracker Status

  Wed Oct 12 16:16:11 CEST 2005 : Protocol.Type : AX25
  Wed Oct 12 16:16:11 CEST 2005 : Protocol.StatusMsg : The protocol thread is in the first step!

  Wed Oct 12 16:16:11 CEST 2005 : Tracker.CurrentPosition : 104.00^{o}Az 00.00^{o}El
  Wed Oct 12 16:16:11 CEST 2005 : Tracker.ExpectedPosition : 147.59^{o}Az 00.00^{o}El 03345.5km

  Get Protocol Status
  Get Tracker Status

  Wed Oct 12 16:16:21 CEST 2005 : Protocol.Type : AX25
  Wed Oct 12 16:16:21 CEST 2005 : Protocol.StatusMsg : The protocol thread is in the second step!

  Wed Oct 12 16:16:21 CEST 2005 : Tracker.CurrentPosition : 148.00^{o}Az 00.00^{o}El
  Wed Oct 12 16:16:21 CEST 2005 : Tracker.ExpectedPosition : 311.00^{o}Az 210.77^{o}El

Test Case 7 and Test Case 14

These test cases have been tested in the unit testing of the manual operation(Section C.1.7).

Test Case 8

Start the protocol when the session is running.

- **The client side**:
  heidi src # ./runOperator

  Starting...

  Sending session to 192.38.79.147:4545

  Start Protocol!

  Wed Oct 12 16:35:44 CEST 2005 : groundStation.gsManager.GSManagerException: The session is running! start protocol forbidden.

Test Case 9

Stop the protocol when the session is running.

- **The client side**:
  heidi src # ./runOperator

  Starting...

  Sending session to 192.38.79.147:4545

  Stop Protocol!

  Wed Oct 12 16:38:35 CEST 2005 : groundStation.gsManager.GSManagerException: The session is running! stop protocol forbidden.

Test Case 10

Start the tracker when the session is running.

- **The client side**:
  heidi src # ./runOperator

  Starting...

  Sending session to 192.38.79.147:4545

  Start Tracker!

  Wed Oct 12 16:39:03 CEST 2005 : groundStation.gsManager.GSManagerException: The session is running! start tracker forbidden.

Test Case 11

Stop the tracker when the session is running.

- **The client side**:
  heidi src # ./runOperator

  Starting...

  Sending session to 192.38.79.147:4545

  Stop Tracker!

  Wed Oct 12 16:39:33 CEST 2005 : groundStation.gsManager.GSManagerException: The session is running! stop tracker forbidden.

Test Case 12 and Test Case 13

Get the status of the protocol and the tracker when the session is running.

- **The client side**:
  heidi src # ./runOperator
  Starting...
  Sending session to 192.38.79.147:4545
  Get Protocol Status
  Get Tracker Status

  Wed Oct 12 16:40:11 CEST 2005 : Protocol.Type : AX25
  Wed Oct 12 16:40:11 CEST 2005 : Protocol.StatusMsg : The protocol thread is in the second step!

  Wed Oct 12 16:40:11 CEST 2005 : Tracker.CurrentPosition : 185.00^{o}Az 132.00^{o}El
  Wed Oct 12 16:40:11 CEST 2005 : Tracker.ExpectedPosition : 178.97^{o}Az 172.29^{o}El

  Get Protocol Status
  Get Tracker Status

  Wed Oct 12 16:40:21 CEST 2005 : Protocol.Type : AX25
  Wed Oct 12 16:40:21 CEST 2005 : Protocol.StatusMsg : The protocol thread is in the second step!

  Wed Oct 12 16:40:21 CEST 2005 : Tracker.CurrentPosition : 185.00^{o}Az 155.00^{o}El

Wed Oct 12 16:40:21 CEST 2005 : Tracker.ExpectedPosition : 177.91^{o}Az 172.91^{o}El

Get Protocol Status
Get Tracker Status

Wed Oct 12 16:40:31 CEST 2005 : Protocol.Type : AX25
Wed Oct 12 16:40:31 CEST 2005 : Protocol.StatusMsg : The protocol thread is in the second step!

Wed Oct 12 16:40:31 CEST 2005 : Tracker.CurrentPosition : 185.00^{o}Az 174.00^{o}El
Wed Oct 12 16:40:31 CEST 2005 : Tracker.ExpectedPosition : 176.91^{o}Az 173.51^{o}El

C.3 Test Cases of On Site Testing

C.3.1 Tracking CUTE-1

AOS time : 16:00:17 Loc Sun Oct 16 2005
LOS time : 16:12:45 Loc Sun Oct 16 2005
Duration : 00:12:27
AOS Az. : 98^o
Max El. : 15^o
LOS Az. : 349^o
Orbit # : 11,907
Frequency : 436837500

Test Case for Tracking CUTE-1

Start to track CUTE-1 in the ground station control system.

- **The control system side**: Sun Oct 16 15:57:08 CEST 2005 : Session Controller: The session controller thread is running!

  Sun Oct 16 15:57:08 CEST 2005 : the session period: 900001

  Sun Oct 16 15:57:08 CEST 2005 : Session Controller: The thread is waiting!

  Sun Oct 16 15:57:08 CEST 2005 : Session: The Session thread is running!

  Sun Oct 16 15:57:08 CEST 2005 : starting protocol

Protocol: Connected with MCC server /127.0.0.1:1500

Sun Oct 16 15:57:18 CEST 2005 : setting satellite

15:57:19 CEST 2005 : Satellite initialized

Sun Oct 16 15:57:19 CEST 2005 : starting initTrack

Sun Oct 16 15:57:19 CEST 2005 : nextPassDate: Sun Oct 16 16:00:18 CEST 2005

Sun Oct 16 15:57:19 CEST 2005 : expectedPos: 097.91^{o}Az 00.00^{o}El 03345.9km

Sun Oct 16 15:57:19 CEST 2005 : currentPos: 146.00^{o}Az 166.00^{o}El

Sun Oct 16 15:58:31 CEST 2005 : Current postion is almost equal to expected position

Sun Oct 16 15:58:59 CEST 2005 : 097.91^{o}Az 00.00^{o}El 03345.9km

Sun Oct 16 15:58:59 CEST 2005 : currentPos: 098.00^{o}Az 01.00^{o}El

Sun Oct 16 15:58:59 CEST 2005 : starting tracking

Sun Oct 16 15:58:59 CEST 2005 : init delay: 78808

Trans Exp:277.91^{o}Az 180.00^{o}El velocity:+899.4935^{o}Az +894.9505^{o}El
NoTrans Exp:097.91^{o}Az 00.00^{o}El 03345.9km velocity:+899.4935^{o}Az +894.9505^{o}El
+899.4935^{o}Az +894.9505^{o}El
4.36845364186539E8 Hz

Trans Exp:277.90^{o}Az 179.99^{o}El velocity:+895.5823^{o}Az +894.9018^{o}El
NoTrans Exp:097.90^{o}Az 00.01^{o}El 03344.8km velocity:+904.4177^{o}Az +894.9018^{o}El
+895.5823^{o}Az +894.9018^{o}El
4.368453627200235E8 Hz

Trans Exp:277.88^{o}Az 179.98^{o}El velocity:+894.3418^{o}Az +894.8531^{o}El
NoTrans Exp:097.88^{o}Az 00.02^{o}El 03343.7km velocity:+894.3418^{o}Az +894.8531^{o}El
+894.3418^{o}Az +894.8531^{o}El
4.368453612517588E8 Hz

.....
.....
.....

we can't see the satellite anymore!

The Session Queue has 0 sessions waiting

C.3.2 Tracking CUBESAT XI-IV

AOS time : 16:23:11 Loc Sun Oct 16 2005
LOS time : 16:36:34 Loc Sun Oct 16 2005
Duration : 00:13:23
AOS Az. : 110^o
Max El. : 20^o
LOS Az. : 348^o
Orbit # : 11,906
Frequency : 436847500

Test Case for Tracking CUBESAT XI-IV

Start to track CUBESAT XI-IV in the ground station control system.

- **The control system side**:

  Sun Oct 16 16:20:57 CEST 2005 : Session Controller: The session controller thread is running!

  Sun Oct 16 16:20:57 CEST 2005 : the session period: 960001

  Sun Oct 16 16:20:57 CEST 2005 : Session Controller: The thread is waiting!

  Sun Oct 16 16:20:57 CEST 2005 : Session: The Session thread is running!

  Sun Oct 16 16:20:57 CEST 2005 : starting protocol

  Protocol: Connected with MCC server /127.0.0.1:1500

  Sun Oct 16 16:21:07 CEST 2005 : setting satellite

  Sun Oct 16 16:21:07 CEST 2005 : Satellite initialized

  Sun Oct 16 16:21:07 CEST 2005 : starting initTrack

  Sun Oct 16 16:21:07 CEST 2005 : nextPassDate: Sun Oct 16 16:22:59 CEST 2005

  Sun Oct 16 16:21:07 CEST 2005 : expectedPos: 109.86^oAz -00.00^oEl 03344.5km

  Sun Oct 16 16:21:07 CEST 2005 : currentPos: 349.00^oAz 01.00^oEl

  Type: 1

  Oct 16 16:21:54 CEST 2005 : Current postion is almost equal to expected position

  Sun Oct 16 16:22:48 CEST 2005 : 109.86^oAz -00.00^oEl 03344.5km

Sun Oct 16 16:22:48 CEST 2005 : currentPos: 110.00^{o}Az 00.00^{o}El

Sun Oct 16 16:22:48 CEST 2005 : starting tracking

Sun Oct 16 16:22:48 CEST 2005 : init delay: 11691

Trans Exp:289.86^{o}Az 180.00^{o}El velocity:+899.2217^{o}Az +899.9474^{o}El
NoTrans Exp:109.86^{o}Az 00.00^{o}El 03344.5km velocity:+899.2217^{o}Az +899.9474^{o}El
+899.2217^{o}Az +899.9474^{o}El
4.368559725268067E8 Hz

Trans Exp:289.84^{o}Az 179.99^{o}El velocity:+895.8419^{o}Az +894.8947^{o}El
NoTrans Exp:109.84^{o}Az 00.01^{o}El 03343.4km velocity:+904.1581^{o}Az +894.8947^{o}El
+895.8419^{o}Az +894.8947^{o}El
4.368559714878731E8 Hz

Trans Exp:289.83^{o}Az 179.98^{o}El velocity:+894.0950^{o}Az +894.8425^{o}El
NoTrans Exp:109.83^{o}Az 00.02^{o}El 03342.2km velocity:+894.0950^{o}Az +894.8425^{o}El
+894.0950^{o}Az +894.8425^{o}El
4.3685597045779777E8 Hz

.....
.....
.....

we can't see the satellite anymore!

Bibliography

[1] Scott J. Landis, John E. Mulholland, *Low Cost Satellite Ground Control Facility Design*, IEEE AES Systems Magazine, June 1993

[2] RM Barry, F Nel, *University Of Stellenbosch Satellite Ground Station Operation And Future Plans*, IEEE Africon 2002

[3] James W. Cutler, Christopher A. Kitts, *Mercury: A Satellite Ground Station Control System* Space System Development Laboratory, Standford University

[4] James Culter, Peder Linder, Armando Fox, *A Federated Ground Station Network* Space System Development Laboratory, Standford University

[5] James Culter *Ground Station Virtualization* Standford University

[6] Charles Cooper, Ronald Fevig, Jason Patel, *The CubeSat Ground Station At The University Of Arizona*, The University Of Arizona, Tucson, AZ 85721

[7] Dr. U. Haring, Dr. R. Kozlowski *Introduction Of Pattern Oriented Software Concepts In Ground Stations*

[8] *Introduction To Satellite Communication Technology For Nren* http://www.nas.nasa.gov/News/Techreports/2004/PDF/nas-04-009.pdf

[9] Forest Fisher, Mark L. James, Barbara Engelhardt, *An Architecture For An Autonomous Ground Station Controller*, 2001 IEEE

[10] Tara Estlin, Forest Fisher, Darren Mutz, Steve Chien, *Automated Generation Of Antenna Tracking Plans For A Deep Space Communications Station*, Jet Propulsion Laboratory, California Institute of Technology

[11] Heiko Damerow, Joachim Schwarz, *Satellite Data Reception System At Multimission Ground Station*, German Aerospace Center

[12] Y. Daniel Liang, *Introduction To Java Programming*, Third Edition, Prentice Hall, 2001

[13] Bruce Eckel, *Thinking in Java*, Prentice Hall PTR; 3 edition, December 6, 2002

[14] Peter Fortescue (Editor), John Stark (Editor), Graham Swinerd (Editor), *Spacecraft Systems Engineering*, 3rd Edition, Wiley, March 2003

[15] Mercury Ground Station Software
http://mgsn.sourceforge.net/index.php

[16] JStation - Java Satellite Ground Station
http://www.qsl.net/n6lyt/download.html

[17] DTUsat
http://dtusat.dtu.dk/

[18] Jeff Tranter, *Linux Amateur Radio AX.25 HOWTO*
http://www.tldp.org/HOWTO/AX25-HOWTO/

[19] NORAD
http://www.norad.mil/index.cfm?fuseaction=home.welcome

[20] AX.25 Amateur Packet-Radio Link-Layer Protocol
http://www.tapr.org/pub_ax25.html

[21] Java Remote Method Inovation (Java RMI)
http://java.sun.com/products/jdk/rmi/

[22] Java Technology
http://java.sun.com/

[23] The Linux Home Page
http://www.linux.org/

[24] Introduction To TCP/IP
http://www.yale.edu/pclt/COMM/TCPIP.HTM

[25] Sputnik
http://www.hq.nasa.gov/office/pao/History/sputnik/

[26] JGuru: Remote Method Invocation(RMI)
http://java.sun.com/developer/onlineTraining/rmi/RMI.html

[27] Software Testing
http://en.wikipedia.org/wiki/Software_testing

[28] CUTE 1, 1.7
http://space.skyrocket.de/doc_sdat/cute-1.htm

[29] Satellite Tracking Software
http://www.nlsa.com/

[30] Default Policy Implementation and Policy File Syntax
http://java.sun.com/j2se/1.3/docs/guide/security/PolicyFiles.html

[31] Informatics and Mathematical Modelling
http://www.imm.dtu.dk

[32] Web Services
http://www.w3.org/2002/ws/

[33] Two Line Elements
http://en.wikipedia.org/wiki/TLE

[34] Concurrent Programming Using the Java Language
https://www.cs.drexel.edu/s̃hartley/ConcProgJava/monitors.html